UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM SEGURANÇA DA INFORMAÇÃO

FRANCISCO GABRIEL BARRETO GOMES

IMPLEMENTAÇÃO E VALIDAÇÃO DE UM SISTEMA PROTÓTIPO PARA A DETECÇÃO DE DISCURSO DE ÓDIO USANDO INTELIGÊNCIA ARTIFICIAL

ITAPAJÉ

2024

FRANCISCO GABRIEL BARRETO GOMES

IMPLEMENTAÇÃO E VALIDAÇÃO DE UM SISTEMA PROTÓTIPO PARA A DETECÇÃO DE DISCURSO DE ÓDIO USANDO INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Segurança da Informação do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de tecnólogo em Segurança da Informação.

Orientador: Prof. Dr. Juan Sebastian Toquica Arenas

ITAPAJÉ

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

G614i Gomes, Francisco Gabriel Barreto.
Implementação e Validação de um Sistema Protótipo para a Detecção de Discurso de Ódio usando Inteligência Artificial / Francisco Gabriel Barreto Gomes. – 2024.
64 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Itapajé, Curso de Segurança da Informação, Fortaleza, 2024.
Orientação: Prof. Dr. Juan Sebastian Toquica Arenas.

1. Chat. 2. Inteligência artificial. 3. BERTimbau. 4. Segurança. 5. Moderação. I. Título.
CDD 005.8

FRANCISCO GABRIEL BARRETO GOMES

IMPLEMENTAÇÃO E VALIDAÇÃO DE UM SISTEMA PROTÓTIPO PARA A DETECÇÃO DE DISCURSO DE ÓDIO USANDO INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Segurança da Informação do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de tecnólogo em Segurança da Informação.

Aprovada em: 01/10/2024

BANCA EXAMINADORA

Prof. Dr. Juan Sebastian Toquica Arenas (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr(c). Artur de Oliveira da Rocha Franco
Universidade Federal do Ceará (UFC)

Prof. Dr. João Henrique Gonçalves Medeiros Corrêa
Universidade Federal do Ceará (UFC)

À minha família por ter sido o meu alicerce primordial de toda uma caminhada estudantil, sendo estes crentes de minha capacidade para chegar até aqui.

AGRADECIMENTOS

Ao Prof. Dr. Juan Sebastian Toquica Arenas por me orientar em meu primeiro trabalho de conclusão de curso pela Universidade Federal do Ceará, sua paciência e destreza durante o andar de minha caminhada acadêmica foram de suma importância para a conclusão de mais uma etapa da minha vida.

Ao Ex Prof. Dr. Max Eduardo Vizcarra Melgar, por me apresentar de forma ampla a área da Segurança da Informação e todos os seus desafios em nossa sociedade contemporânea.

Ao Prof. Dr. João Henrique Corrêa, por me apresentar a educação superior de maneira celebre e sua paixão por ensinar.

À professora e amiga, Isabel Cristina Barros, onde em meio a etapa inicial do curso de graduação mostrou acreditar em mim e me tornou capaz de conseguir finalizar inúmeros trabalhos ao longo de todo o período de estudo.

Aos alunos de graduação em Segurança da Informação e amigos, João Luís dos Santos Filho, Thais de Andrade Castro e Maria Alyce Sousa Oliveira, que sempre se mantiveram crentes de minha capacidade desde o início do curso de graduação e que nos momentos mais difíceis foram parte do meu alicerce que hoje tenho como certo de que sempre existirão pessoas em quem de fato posso confiar.

À amiga Myrna Silva Gardel, que mesmo diante dos inúmeros percalços que minha vida possa ter me dado, se manteve disposta a escutar e sempre oferecer um ombro amigo em meio ao meu caos orientando-me sempre em direção ao melhor caminho possível. Este sendo o que me manteve fiel ao que de fato acredito e em mim a capacidade de crer que pudesse sempre superar as adversidades onde minha vida colocasse a frente contra minha vontade, sejam elas quaisquer que assim fossem.

À minha mãe, Francisca Lopes Barreto, aos meus irmãos, Joice Lopes Barreto e Jopson Lopes Barreto, que nos momentos de minha ausência dedicados ao estudo superior, fizeram-se entender que o futuro é feito a partir da constante dedicação no presente de algo que fielmente se acredita, sendo este, a plena fé na educação.

Por fim, agradeço todo o corpo docente da Universidade Federal do Ceará do Campus de Itapajé que em ensinaram além do desenvolvimento quanto ao pensamento racional dentro da minha área de atuação, como também, tornaram-me alguém mais empático, amável e respeitoso para com as minhas próprias atitudes, modos de pensar e principalmente, onde tive a honra de assistir verdadeiros amantes e mestres da educação em ação todos os dias.

"From rivers of sorrow. To oceans deep with hope. I have traveled them."

(Chuck Schuldiner)

RESUMO

Este trabalho apresenta a implementação e validação para a identificação de discurso de ódio em chats interativos, utilizando Python e o modelo pré-treinado de linguagem natural BERTimbau. Com o crescimento das plataformas de comunicação digital, observou-se um aumento da linguagem disfarçada que expressa preconceitos e incita violência, frequentemente direcionada a indivíduos ou grupos vulneráveis. O sistema proposto visa detectar essas manifestações de discurso de ódio e analisar seus padrões, oferecendo uma abordagem mais eficaz para a moderação de conteúdo. A implementação prática da ferramenta demonstra sua aplicabilidade em ambientes de chat, contribuindo para a criação de interações mais seguras e respeitosas. Além disso, o estudo busca proporcionar uma compreensão aprofundada de como a tecnologia pode aprimorar a moderação de conteúdo, protegendo os usuários contra abusos online. A presente implementação também investiga a relação entre discurso de ódio e a eficácia das estratégias de moderação, contribuindo para a segurança e qualidade das interações digitais. Por meio da análise de dados coletados, espera-se identificar tendências e padrões que ajudem a refinar técnicas de moderação e a promover um ambiente digital mais inclusivo. A utilização do BERTimbau potencializa a precisão na identificação de nuances linguísticas, permitindo um reconhecimento mais eficiente de discursos prejudiciais e abrindo novas possibilidades para o desenvolvimento de soluções de proteção em contextos de chat.

Palavras-chave: chat; inteligência artificial; BERTimbau; segurança; moderação.

ABSTRACT

This work presents the implementation and validation for identifying hate speech in interactive chats, using Python and the pre-trained natural language model BERTimbau. With the growth of digital communication platforms, there has been an increase in disguised language that expresses prejudice and incites violence, often directed at vulnerable individuals or groups. The proposed system aims to detect these manifestations of hate speech and analyze their patterns, offering a more effective approach to content moderation. The practical implementation of the tool demonstrates its applicability in chat environments, contributing to the creation of safer and more respectful interactions. In addition, the study seeks to provide an in-depth understanding of how technology can improve content moderation, protecting users from online abuse. This implementation also investigates the relationship between hate speech and the effectiveness of moderation strategies, contributing to the safety and quality of digital interactions. Through the analysis of collected data, we hope to identify trends and patterns that will help refine moderation techniques and promote a more inclusive digital environment. The use of BERTimbau enhances the accuracy in identifying linguistic nuances, allowing more efficient recognition of harmful speech and opening up new possibilities for the development of protection solutions in chat contexts.

Keywords: chat; artificial intelligence; BERTimbau; security; moderation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No contexto brasileiro, a liberdade de expressão nas mídias sociais tem sido alvo de discussões intensas, especialmente em plataformas amplamente utilizadas no país, como Instagram, Facebook e X (anteriormente Twitter). Essas redes promovem a troca de ideias, mas enfrentam o desafio contínuo de moderar o conteúdo de forma eficaz. Embora utilizem sistemas automatizados para identificar e remover conteúdos ofensivos ou de ódio, essas tecnologias frequentemente não cobrem todas as formas de violência e abuso online, deixando muitos usuários expostos a comentários e discursos prejudiciais em uma ampla variedade de tópicos (LLANSO; HOBOKEN, 2020).

No Brasil, as plataformas de mídia social têm se deparado com questões complexas, como a proliferação de discursos de ódio relacionados à política, raça e gênero, temas frequentemente inflamados no ambiente digital. O anonimato e a vastidão dessas redes dificultam a identificação e controle de conteúdos abusivos, tornando-se um desafio equilibrar a liberdade de expressão com a necessidade de proteger os indivíduos de ataques virtuais. De acordo com Tufekci (2018), a expansão das mídias sociais no Brasil evidenciou lacunas significativas nas estratégias de moderação de conteúdo, especialmente em um cenário onde as tensões políticas e sociais estão exacerbadas.

A tecnologia tem desempenhado um papel crucial na tentativa de mitigar o discurso de ódio no Brasil, em particular com o uso de algoritmos de aprendizado de máquina para moderar o conteúdo nas redes sociais. Plataformas como Facebook e X adotaram sistemas automáticos que analisam grandes volumes de dados para detectar padrões de linguagem ofensiva e evitar a propagação de discursos prejudiciais (TUFEKCI, 2018). No entanto, a aplicação dessas tecnologias enfrenta limitações, como a dificuldade em compreender o contexto cultural e linguístico específico do Brasil, que frequentemente envolve o uso de ironia e gírias regionais.

Modelos de Inteligência Artificial (IA), como BERTimbau, que é uma adaptação do BERT para o português brasileiro, têm sido utilizados para detectar discurso de ódio em textos online. Técnicas de Processamento de Linguagem Natural (PLN) permitem que esses sistemas compreendam nuances da linguagem, identificando expressões ofensivas de forma automatizada. Segundo Nobata *et al.* (2016), embora esses sistemas tenham mostrado resultados promissores, ainda enfrentam desafios, como a dificuldade em lidar com ironias, gírias e o dinamismo da linguagem presente nas redes sociais brasileiras.

Estudos como os de Schmidt e Wiegand (2017) destacam a eficácia do aprendizado

de máquina na detecção de conteúdo abusivo, mostrando que o uso de modelos de classificação e redes neurais pode otimizar a moderação de comentários. No Brasil, onde o volume de interações online é massivo e a polarização política contribui para a intensificação dos discursos de ódio, essa tecnologia tem se mostrado essencial para criar um ambiente digital mais seguro.

Assim, a aplicação em larga escala de IA e técnicas de PLN no Brasil contribui para a criação de espaços digitais mais seguros e respeitosos. Esses sistemas permitem uma moderação mais rápida e precisa, abordando o problema do discurso de ódio de forma eficiente, ao mesmo tempo que consideram as particularidades linguísticas e culturais do país. À medida que a pesquisa em PLN avança, a moderação automática de conteúdo se torna uma ferramenta cada vez mais relevante para garantir interações online mais seguras no Brasil.

1.1 Objetivo geral

Desenvolver uma ferramenta em Python para a identificação e análise de discurso de ódio em chats interativos, utilizando técnicas avançadas de Processamento de Linguagem Natural (PLN) e aprendizado de máquina, com foco na criação de um ambiente digital mais seguro no contexto brasileiro.

1.2 Objetivos específicos

a) Analisar e comparar o desempenho do modelo BERTimbau com outros métodos semelhantes de vetorização em português;
b) Executar e testar o sistema protótipo em um estudo de caso de comunicação em rede, simulando interações em chats;
c) Avaliar a eficácia da solução proposta na identificação do discurso de ódio em português, considerando o contexto brasileiro.

2 FUNDAMENTAÇÃO TEÓRICA

Nos últimos anos, o avanço das técnicas de inteligência artificial, especialmente as relacionadas ao aprendizado profundo (deep learning), transformou significativamente a forma como sistemas computacionais interpretam e processam grandes volumes de dados. Esse progresso é particularmente relevante para o campo do Processamento de Linguagem Natural (PLN), que busca a interação entre computadores e a linguagem humana. Neste contexto, o uso de PLN para identificar discurso de ódio online tornou-se uma abordagem promissora para garantir ambientes virtuais mais seguros. Portanto, neste capitulo se descrevem os conceitos e tecnologias que fazem parte do presente trabalho.

2.1 Cibersegurança e o contexto do discurso de ódio

O aumento da conectividade global e o crescimento das redes sociais tornaram a cibersegurança um campo prioritário na era digital. A cibersegurança abrange um conjunto de práticas que visam proteger sistemas, redes e dados contra ataques maliciosos e uso indevido (SINGER; FRIEDMAN, 2014). Além de proteger infraestruturas críticas e dados pessoais, a cibersegurança enfrenta o desafio hoje de monitorar a disseminação de discurso de ódio e violência online.

O discurso de ódio, frequentemente caracterizado por linguagens que incitam discriminação, violência ou preconceito, tornou-se um problema social crescente nas mídias digitais. Plataformas como Facebook, Instagram e X (antigo Twitter) têm implementado políticas de moderação, mas enfrentam dificuldades em detectar todas as nuances do discurso ofensivo, especialmente as mensagens codificadas ou contextualmente complexas. Segundo Tufekci (2018), essa lacuna na moderação contribui para a perpetuação de abusos online, prejudicando a experiência de usuários vulneráveis.

É válido ressaltar que há legislação que protege os usuários presentes nas redes contra inconveniências morais que venham a surgir, como o Marco Civil da Internet (Lei nº 12.965/2014), que protege a privacidade e os direitos dos usuários na internet, apesar de não trata diretamente da regulação do conteúdo ofensivo. Embora a lei assegure o direito à proteção de dados e à privacidade, o combate ao discurso de ódio exige o uso de ferramentas tecnológicas mais avançadas, como algoritmos de IA especializados na detecção desse tipo de linguagem (REPúBLICA, 2014). Nesse sentido, a combinação de cibersegurança com inteligência artificial

apresenta-se como uma solução viável para moderar de forma eficaz o conteúdo das redes. A análise do discurso de ódio nas interações estudadas envolveu a detecção de nuances linguísticas complexas, como o uso de expressões camufladas ou de termos que, em certos contextos, podem ser interpretados de forma ofensiva. O modelo utilizado foi capaz de identificar variações de discurso de ódio que incluíam ofensas diretas e indiretas, como ironias e sarcasmos, aumentando a eficácia da moderação. Durante os testes, o sistema detectou diversas expressões discriminatórias que, embora sutis, foram classificadas como ofensivas devido ao contexto. Isso evidencia a relevância do modelo para identificar discursos de ódio em diferentes contextos e interações linguísticas.

2.2 Inteligência artificial

A inteligência artificial (IA) pode ser definida como um campo interdisciplinar que visa criar sistemas capazes de realizar tarefas que exigem inteligência humana, como a resolução de problemas complexos, reconhecimento de padrões e aprendizado contínuo. Desde suas raízes filosóficas até sua aplicação prática, a IA busca imitar comportamentos humanos que anteriormente eram considerados únicos e exclusivos de seres humanos. Máquinas, antes programadas para executar tarefas específicas, agora são capazes de aprender com dados e adaptar seu comportamento ao longo do tempo (RUSSELL; NORVIG, 2003).

A IA tem sido aplicada em diversas áreas, como saúde, educação e negócios, com grande impacto:

a) área da saúde: na área da saúde, a IA proporciona diagnósticos rápidos e tratamentos personalizados. Por meio da análise de imagens médicas e da criação de modelos preditivos, é possível prever surtos de doenças e adaptar os tratamentos de acordo com os dados específicos de cada paciente (LIVERPOOL, 2024);
b) área empresarial: no setor empresarial, a IA melhora processos internos e ajuda a prever demandas futuras, otimizando operações e oferecendo uma experiência mais personalizada para os clientes (LIVERPOOL, 2024); e
c) área da educação: a IA está transformando a educação com plataformas adaptativas que ajustam o conteúdo de acordo com as necessidades de cada estudante, proporcionando uma experiência de aprendizado mais personalizada (TAVARES *et al.*, 2020).

Além dessas áreas, a IA desempenha um papel central no Processamento de Linguagem Natural (PLN) , uma subárea da IA que permite a interação entre computadores e a

linguagem humana. Para realizar essas tarefas complexas, são usados Modelos de Linguagem de Grande Escala (LLMs), que aplicam redes neurais profundas para aprender e gerar padrões linguísticos a partir de grandes volumes de texto (JURAFSKY; MARTIN, 2008).

2.3 Modelos de linguagem de grande escala (LLMs)

Os modelos de linguagem de grande escala (LLMs) são ferramentas poderosas no processamento de texto, permitindo que sistemas compreendam, gerem e interajam com a linguagem humana de forma eficiente. Esses modelos utilizam arquiteturas baseadas em *transformers*, introduzidas por Vaswani *et al.* (2017), que empregam mecanismos de atenção para analisar diferentes partes do texto simultaneamente.

Já a biblioteca *Transformers* é atualmente a arquitetura de escolha para PLN devido à sua eficiência em lidar com grandes volumes de dados e em captar contextos complexos. Entre os LLMs mais populares estão o GPT (*Generative Pre-trained Transformer*) e o BERT (*Bidirectional Encoder Representations from Transformers*). Esses modelos revolucionaram o PLN, permitindo tarefas como tradução automática, análise de sentimentos, resumo de textos e moderação de conteúdo.

2.4 BERTimbau: Modelo de Linguagem para o Português

No contexto da língua portuguesa, o BERTimbau se destaca como um modelo de linguagem baseado na arquitetura BERT, treinado especificamente com dados em português. O BERTimbau foi ajustado para diversas tarefas de PLN, incluindo a detecção de discurso de ódio. Por ser um modelo *cased*, ele diferencia entre letras maiúsculas e minúsculas, o que é crucial para capturar nuances da língua portuguesa (RODRIGUES *et al.*, 2023).

Vale ressaltar que o BERTimbau foi treinado usando o dataset HateBR, que contém exemplos de linguagem ofensiva em português do Brasil. O treinamento com esse dataset permitiu que o modelo fosse eficaz na identificação de discursos de ódio e linguagem ofensiva, sendo uma ferramenta valiosa para a moderação de conteúdo em redes sociais, fóruns e plataformas de comunicação.

Além disso, um exemplo prático semelhante ao que foi implementado neste trabalho é descrito por Souza *et al.* (2022), onde o BERTimbau foi utilizado para a detecção de discurso de ódio em redes sociais. No estudo, os autores utilizaram o modelo pré-treinado combinado com o

dataset HateBR para moderar automaticamente comentários em posts de uma plataforma online. O sistema processava os textos em tempo real e categorizava os comentários como ofensivos ou não.

Durante os experimentos, o modelo foi avaliado em diferentes cenários, incluindo conversas abertas entre usuários, resultando em uma acurácia de 85% e um F1-Score de 87%. O tempo médio de processamento de cada comentário foi de 0.25 segundos, mostrando a viabilidade do uso de modelos de linguagem pré-treinados na moderação de conteúdo em tempo real. Esses resultados corroboram com os encontrados no presente estudo, destacando a relevância do BERTimbau para o combate ao discurso de ódio online (SOUZA *et al.*, 2022).

2.5 Benefícios da identificação do discurso de ódio

Um dos principais benefícios da identificação do discurso de ódio em ambientes virtuais, especialmente nas seções de comentários, é a melhoria da experiência de leitura e a criação de um espaço mais seguro para a troca de opiniões sobre notícias, por exemplo. Isso ajuda a evitar a disseminação de linguagem ofensiva e a prevenir a interpretação equivocada de frases como liberdade de expressão. Segundo um estudo recente de Ribeiro *et al.* (2023), a moderação eficiente de comentários em plataformas online não só reduz a incidência de discurso de ódio, mas também contribui para um ambiente mais saudável e construtivo para os usuários.

Além disso, ser possível a prevenção para com as crianças, adolescentes ou adultos que possuem acesso a rede de maneira que estas pessoas não sejam vítimas de comentários tendenciosos, maldosos ou com tom difamatórios que venham a ofender a sua dignidade física ou mental, ferindo assim sua própria imagem (MATIAS *et al.*, 2022).

Ademais, é notório que um diálogo cívico e saudável se torna não somente uma realidade distópica, mas sim, algo de suma importância e cada vez mais possível a sua realização deste dentro da rede de computadores, garantindo assim a liberdade de expressão sem ofensas diretas ou indiretas a grupos menos representativos, estes muitas vezes sem apoio devido de nossa sociedade.

É importante salientar que as grandes companhias, donas das maiores redes que interligam milhões de usuários simultâneos, como exemplo a empresa de tecnologia Meta (antigo Facebook Inc.), tem como principal matriz a criação de tecnologias que interliguem amigos e familiares, trabalham arduamente para evitar que os discursos de ódio se proliferem em suas duas principais redes, como o Facebook e Instagram. As redes citadas já possuem seus próprios

identificadores de linguagem ofensiva, porém, esses são pacíficos quanto a identificação imediata dos tais discursos possuindo falhas que deterioram o pleno desenvolvimento para uma melhor experiência de seus usuários dentro da rede, dependendo assim muitas vezes do fator humano para colaborar com as denúncias e a remoção de determinado comentário ou conteúdo que se julga assim inadequado, estando restrito a um período de análise demorado e cansativo (GILLESPIE, 2021).

Além disso, se faz necessária a vigilância constante de todos os comentários postados e anteriores frente a implementação de um possível algoritmo - conjunto de instruções e regras que um programa de computador é capaz de executar a partir de comandos -, seguindo as normas postas na Lei Geral de Proteção de Dados, popularmente conhecida por LGPD aprovada em 2018 pelo congresso nacional (REPúBLICA, 2018), e que estes constem nos termos de uso de cada rede social, possuindo a capacidade para executar suas funções respectivas venham a serem usadas quando que, ao publicar algo, o autor da postagem receba um aviso referente ao que escreveu. Caso este tenha feito a sua postagem com conteúdo inadequado infringindo assim as normas da rede social e as normas legais, é enviado a este usuário um aviso de que tal prática é danosa, consequentemente, a postagem é impedida de ser postada de forma pública. (GOGONI, 2023)

2.6 Comparação de trabalhos prévios

O modelo `'neuralmind/bert-large-portuguese-cased'` foi ajustado (*finetuned*) especificamente para a tarefa de Detecção de Linguagem Ofensiva, utilizando o conjunto de dados HateBR. Este modelo é baseado na arquitetura BERT, sendo treinado inicialmente em uma grande quantidade de textos em português. Posteriormente, ele passou por um ajuste fino usando o HateBR, um dataset especializado em identificar discursos de ódio e linguagem ofensiva em português (RODRIGUES *et al.*, 2023).

Por ser um modelo *cased*, ele diferencia entre maiúsculas e minúsculas, o que significa que a forma como as palavras são capitalizadas pode influenciar sua interpretação. Essa característica é importante para capturar nuances da língua portuguesa, onde a capitalização pode mudar o significado ou o contexto das palavras.

Esse modelo é o mais adequado para aplicações em português que precisam detectar e moderar linguagem ofensiva, como em plataformas de mídia social, fóruns ou sistemas de comentários. Ele pode ser usado para identificar automaticamente conteúdos prejudiciais ou

inapropriados, ajudando a promover interações mais saudáveis online, assim como é o objetivo central deste trabalho.

Como exemplo disto, o projeto intitulado "*Evaluation of Portuguese Language Models (EPLM)*", desenvolvido por Ruan Chaves Rodrigues, Marc Tanti e Rodrigo Agerri, tem como principal objetivo avaliar modelos de linguagem da língua portuguesa em tarefas diversas de processamento de linguagem natural (PLN). Segundo Rodrigues *et al.* (2023), "O foco do projeto é realizar uma avaliação sistemática de diferentes modelos de linguagem treinados para o português, buscando compreender suas capacidades e limitações em várias tarefas de PLN, como análise de sentimentos e reconhecimento de entidades nomeadas"(p. 2). A seguir, são detalhadas as etapas principais envolvidas no desenvolvimento do projeto:

a) definição do problema: o primeiro passo foi a identificação da carência de uma avaliação rigorosa e sistemática de modelos de linguagem para o português, especialmente em comparação com outras línguas, como o inglês. De acordo com os autores, "Embora existam diversos modelos avançados para o inglês, o português ainda sofre com a falta de uma análise comparativa robusta, o que limita o desenvolvimento de aplicações baseadas nesses modelos"Rodrigues *et al.* (2023). Portanto, o projeto foi delineado com o propósito de preencher essa lacuna;
b) coleta de dados: os dados necessários para a realização das avaliações foram coletados a partir de corpora diversificados, englobando diferentes domínios da língua portuguesa, como mídia social, notícias e textos jurídicos. Conforme relatado por Rodrigues *et al.* (2023), "A diversidade de fontes de dados foi crucial para garantir que os modelos fossem testados em uma ampla gama de contextos, permitindo uma avaliação mais completa e robusta"(p. 4);
c) seleção e treinamento dos modelos: uma etapa crucial foi a escolha dos modelos de linguagem que seriam avaliados. Segundo o relatório do projeto, "Os modelos selecionados incluíram versões grandes e ajustadas de BERT, BERTimbau e outros modelos pré-treinados especificamente para o português"conforme é explorado em Rodrigues *et al.* (2023). Após essa escolha, os modelos foram ajustados para as tarefas específicas de PLN, utilizando técnicas de *fine-tuning*, nas quais os modelos são refinados com o uso de dados adicionais de treinamento;
d) avaliação e métricas: para mensurar a performance dos modelos, foram utilizadas métricas amplamente aceitas na comunidade de PLN, como acurácia, precisão, recall e F1-score. De acordo com os desenvolvedores, "Essas métricas permitem uma avaliação objetiva do

desempenho dos modelos em tarefas distintas, como classificação de texto e reconhecimento de entidades"Rodrigues *et al.* (2023) (p. 6). Além das métricas de desempenho, o tempo de inferência e os requisitos de hardware também foram levados em consideração, conforme destacado: "Consideramos não apenas o desempenho em termos de acurácia, mas também o tempo necessário para processar as tarefas e a eficiência no uso de recursos computacionais"Rodrigues *et al.* (2023) (p. 6); e

e) comparação dos resultados: após a avaliação, os resultados foram comparados para identificar os pontos fortes e fracos de cada modelo. De acordo com o relatório final, "A análise comparativa demonstrou que os modelos mais recentes, como o BERTimbau, apresentam melhor performance em tarefas de análise de sentimentos e NER, mas ainda há margem para melhorias, especialmente em termos de eficiência de recursos", assim pontuado em Rodrigues *et al.* (2023) (p. 7). Essa análise permitiu que os pesquisadores propusessem novas direções para futuros aprimoramentos nos modelos.

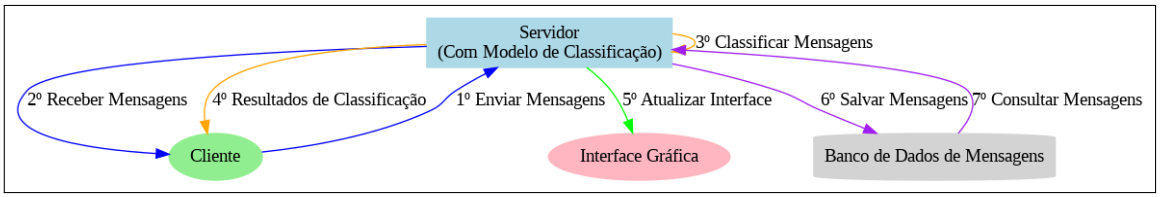
3 METODOLOGIA

O estudo se baseia na implementação de um chat em *Python* onde o referido tem capacidade de classificação automática de mensagens para identificar discursos ofensivos, popularmente conhecidos também como discursos de ódio, utilizando técnicas de Processamento de Linguagem Natural (PLN) e aprendizado de máquina. O servidor é projetado para operar em tempo real, permitindo a interação entre usuários enquanto monitora e classifica o conteúdo das mensagens enviadas. Portando, se faz necessário entender quais bibliotecas, ou seja, conjuntos de ferramentas agrupadas usadas por um programa de computador capaz de executar as funções especificas. O código para o cabeçalho de importação, está presente no arquivo exposto no arquivo `'servidor.py'` presente no repositório do *Github* (BARRETO, 2024a).

3.1 Ambiente de desenvolvimento

Para que se fosse possível atender os objetivos e por em prática a discussão presente é disposto o esquema do fluxo dados presentes na figura 1, na sequência. Vale salientar que durante o estudo foi realizada uma adaptação de um modelo pré-treinado, da base de dados *Hugging Face* sendo esses importados através do próprio código utilizando a plataforma com interface gráfica *Pycharm Community Edition* (JETBRAINS, 2024), este que tem por objetivo a execução de códigos da linguagem de programação *Python*.

Figura 1 – Modelo de fluxo de dados do código.



Fonte: Elaborado pelo autor (2024).

O dataset utilizado, 'HateBR', foi dividido em duas partes: 80% dos dados para treinamento do modelo e 20% para testes e validação. Essa divisão garantiu que o modelo fosse exposto a uma ampla variedade de expressões ofensivas, permitindo uma aprendizagem robusta. As frases incluíam categorias como "ofensivas"ou "não ofensivas", com um balanceamento adequado das classes para evitar viés no treinamento. Válido ressaltar que a seleção desta maneira assegurou que o modelo fosse testado em situações realistas. Durante os testes, o sistema detectou diversas expressões discriminatórias que, embora sutis, foram classificadas

como ofensivas devido ao contexto. Isso evidencia a relevância do modelo para identificar discursos de ódio em diferentes contextos e interações linguísticas.

As métricas de avaliação em aprendizado de máquina e processamento de linguagem natural são usadas para medir o desempenho de modelos de classificação, como o exemplo citado do BERT. A seguir, são listadas as principais métricas utilizadas para a análise deste estudo:

a) acurácia (*accuracy*): representa a proporção de previsões corretas entre o total de previsões. Formalmente, é definida como a razão entre o número de classificações corretas (verdadeiros positivos e verdadeiros negativos) e o número total de exemplos avaliados. A fórmula é dada por:

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN}$$

Onde:

- TP: Verdadeiros Positivos
- TN: Verdadeiros Negativos
- FP: Falsos Positivos
- FN: Falsos Negativos

Segundo Bishop (2006), a acurácia é útil quando as classes estão balanceadas, mas pode ser enganosa quando há uma distribuição desbalanceada de classes, onde uma alta acurácia pode ser atingida simplesmente prevendo a classe majoritária em todos os casos;

b) precisão (*precision*): métrica que indica a proporção de exemplos classificados corretamente como positivos em relação ao total de exemplos previstos como positivos. A fórmula é:

$$\text{Precisão} = \frac{TP}{TP + FP}$$

Segundo Manning *et al.* (2008), este destaca que a precisão é especialmente importante em sistemas como recuperação de informações, onde se deseja minimizar o número de documentos irrelevantes retornados;

c) *recall* (ou sensibilidade): mede a capacidade do modelo de identificar corretamente todos os exemplos positivos. A fórmula é:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Goodfellow *et al.* (2016) destacam que o recall é particularmente importante quando o custo de falsos negativos é alto;

d) *f1-score*: média harmônica entre a precisão e o recall, equilibrando ambas as métricas. Ele é usado quando há um compromisso entre precisão e recall, e quando o interesse é maximizar ambos ao mesmo tempo. A fórmula é:

$$F1 = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}}$$

Como apontado por Jurafsky e Martin (2021), o *F1-Score* é especialmente útil em problemas de classificação binária com classes desbalanceadas, fornecendo uma única métrica que reflete o desempenho geral de um modelo. Durante os testes, o sistema demonstrou uma alta taxa de acurácia na identificação de discursos de ódio, com resultados que confirmam sua eficácia em ambientes de chat dinâmicos. As mensagens analisadas revelaram uma diversidade de expressões de ódio, desde ofensas diretas até insinuações sutis. Essa variedade ressalta a importância de um sistema de moderação que possa se adaptar e evoluir com o uso da linguagem.

3.2 Bibliotecas Python utilizadas

Nessa seção, é abobada quais bibliotecas Python foram de suma importância para o estudo em questão, dentre as principais temos:

a) *socket*: biblioteca de Rede em Python que permite a comunicação eficiente e flexível entre dispositivos em uma rede (FOUNDATION, 2024a);

b) *theading*: biblioteca que permite criar e gerenciar threads, que são unidades independentes de execução que podem rodar simultaneamente no seu programa, gerando ganho de tempo de exeução, por rodar mais de uma função ao mesmo tempo (FOUNDATION, 2024b);

c) *transformers*: biblioteca desenvolvida pela Hugging Face, é um conjunto de ferramentas de código aberto para deep learning, aprendizado profundo, projetada para diversas modalidades, como processamento de linguagem natural, visão computacional, áudio e aplicações multimodais. Ela oferece uma ampla gama de modelos pré-treinados, que podem ser ajustados para maximizar o desempenho em tarefas específicas (FACE, 2024);

d) *numpy*, *scipy* e *torch*: a biblioteca NumPy é destinada principalmente a realizar operações em arrays multidimensionais, denominada como ndarray nesta biblioteca. Sendo também

possível encontrar funções matemáticas para operações rápidas em arrays, sem a necessidade de escrever laços, recursos de álgebra linear, geração de números aleatórios. Já a SciPy, abreviação de *Scientific Python*, é uma biblioteca que proporciona um conjunto robusto de módulos para otimização, envolvendo álgebra linear, integração, interpolação, funções especiais, solução de equações diferenciais e outras tarefas comuns na ciência e engenharia. Por fim, a biblioteca *Torch*, é uma biblioteca Python de aprendizado de máquina de código aberto usada para implementações de aprendizado profundo, como visão computacional e processamento de linguagem natural. (VIRTANEN *et al.*, 2020)

3.3 Componentes de uma comunicação em rede

Se faz necessário também entender como se deu a troca de mensagens dentro da arquitetura que compõe a troca de mensagens neste projeto, a mesma foi pensada no modelo cliente-servidor, onde baseia-se em um modelo de arquitetura de rede amplamente utilizado geralmente em ambientes de Tecnologia da Informação. É possível notar que em uma arquitetura simples de transferência de dados, os computadores são divididos em dois grupos: os servidores, que fornecem serviços ou recursos, e clientes, que solicitam estes serviços e recursos, recebendo-os como respostas. Este modelo é amplamente usado em redes corporativas, sejam estas geralmente de pequeno e médio porte, que tenham aplicações projetadas para rede, seja elas: Web ou por exemplo, um sistema de gerenciamento de banco de dados, entre outros.

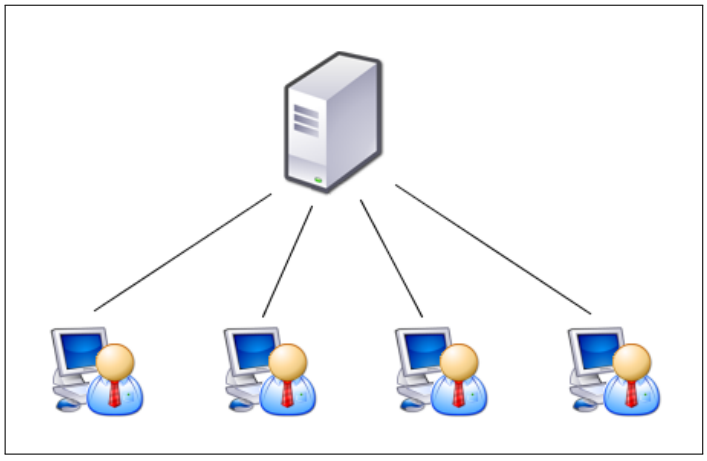
Os principais componentes dessa estrutura cliente-servidor são:

a) cliente: podemos definir este como sendo qualquer dispositivo de computação, como desktops, smartphones, tablets ou aplicativos que, através de protocolos de rede se conectam a servidores para fazer uma solicitação de serviço ou recurso e aguarda a resposta do servidor (TANENBAUM; WETHERALL, 2010);
b) servidor: este pode ser definido como sendo o componente do sistema que fornece os serviços ou recursos solicitados pelos clientes. Estes servidores podem ser, por exemplo, um computador projetado especificamente para uma função ou mais funções aos seus clientes este sendo executado dentro de um servidor via rede (KUROSE; ROSS, 2017);
c) rede: a rede é a infraestrutura que conecta os clientes ao servidor. Ela pode ser tanto local, onde chamamos de LAN (*Local Area Network*) ou global, popularmente conhecida por internet (FOROUZAN, 2012); e
d) procolos de comunicação: tratam-se de um conjunto de regras e procedimentos que regem a

comunicação, entre um cliente e servidor. Podemos ter vários destes, dentre os mais comuns temos o HTTP (*Hypertext Transfer Protocol*) usado na web, o SMTP (*Simple Mail Transfer Protocol*) usado no e-mail e o FTP (*File Transfer Protocol*) usado para transferir arquivos. (BONAVENTURE, 2021)

Na figura 2 é demonstrado de forma prática como se da a conexão neste estudo, sendo aplicável quando o cliente requer a conexão com o servidor. Logo, espera-se que o servidor esteja disponível portando um endereço de internet local juntamente com a porta de conexão que permite a conexão desses dois meios, onde este estão localizados na própria máquina.

Figura 2 – Arquitetura Cliente-Servidor.



Fonte: (ARQSERV, 2012).

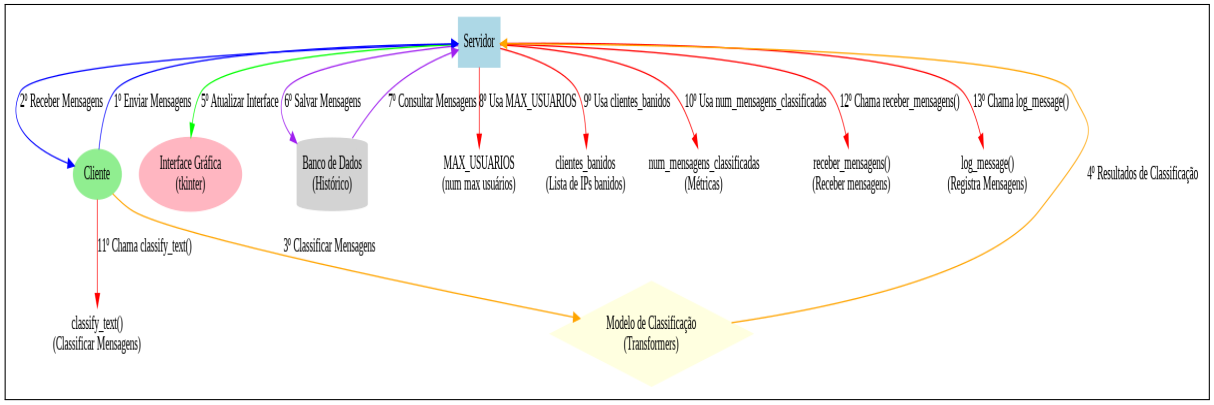
3.4 Configuração inicial do servidor

É salientado que servidor foi configurado para aceitar um número máximo de 3 (três) usuários simultâneos (`MAX_USUARIOS = 3`), número este podendo ser alterado a depender do tamanho da requisição de testes desejados, neste estudo, somente foi usado o espaço de dois usuários. Além disso, é criada uma lista de clientes banidos (`clientes_banidos`) e uma lista de clientes conectados conforme é demonstrado em (BARRETO, 2024a). Nesse contexto, o pilar da privacidade também desempenha um papel essencial, garantindo que, ao monitorar e gerenciar essas listas, a identidade e os dados pessoais dos usuários sejam protegidos. Isso permite que o sistema controle o acesso e o comportamento dos usuários, preservando a confidencialidade das informações enquanto combate efetivamente o discurso de ódio sem comprometer a privacidade dos envolvidos.

3.5 Carregamento do modelo de configuração

Foi utilizado o modelo fornecido pela biblioteca *Transformers*, desenvolvido por (RODRIGUES *et al.*, 2023). O modelo é especializado em identificar discurso ofensivo em português do Brasil, sendo este adaptado para ser usado no código. A escolha não é por acaso: em testes realizados pelo próprio criador do modelo, a precisão quanto ao reconhecimento mostrou-se bastante satisfatória e avante se comparado ao modelo `Base`. A variável `'classify_text'` foi desenvolvida para processar as mensagens dos usuários e classificá-las como ofensivas ou não ofensivas de acordo com o classificador.

Figura 3 – Modelo de fluxo de dados do servidor.



Fonte: Elaborado pelo autor (2024).

3.6 Processo de classificação

Vale lembrar que o processo de classificação de mensagens é fundamental para garantir que o sistema possa identificar e categorizar o conteúdo de forma eficiente e precisa. Esta seção descreve os passos envolvidos, desde o recebimento inicial da mensagem até a classificação final e o registro do tempo de processamento. Cada etapa desempenha um papel crucial na determinação da natureza da mensagem, assegurando que o sistema possa lidar adequadamente com diferentes tipos de conteúdo da seguinte maneira:

a) recebimento da mensagem: quando algum usuário envia uma mensagem, ela é recebida por uma *string*, ou seja, um campo que recebe textos pela função `'classify_text'`;

b) tokenização do texto: Logo após o recebimento da mensagem, ela então é convertida em um formato em que o modelo de processamento de linguagem natural (PLN) possa vir a interpretar. Esse processo, chamado tokenização, basicamente transforma o texto em uma série de *tokens*, ou seja, pedaços menores da mensagem, removendo caracteres especiais

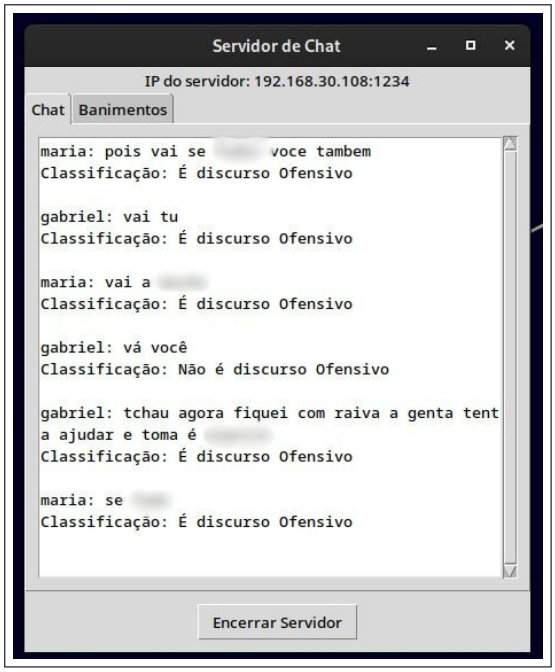
para dar ênfase ao que de fato é importante;

c) classificação com o modelo: a mensagem anteriormente tokenizada é alimentada no modelo, que foi previamente treinado para identificar discurso ofensivo, através do *dataset* 'Hate-BR'. Neste caso, como "Ofensivo"ou "Não Ofensivo";
d) determinação de classe: após o cálculo das pontuações pelo modelo de linguagem, a função identifica a classificação com a maior pontuação. Se a pontuação mais alta estiver na categoria dita "Ofensivo", a mensagem é classificada como esta determinada função. Caso contrário, ela é considerada "Não Ofensiva"; e
e) registro do tempo de processamento: tempo necessário para concluir todo esse processo onde é medido e registrado.

3.7 Recepção e envio de mensagens

A função `receber_mensagens` é responsável pelo recebimento de mensagens por alguns dos clientes conectados, classificá-los, exibi-las na interface gráfica e enviá-las para os outros usuários conectados do servidor. As mensagens, juntamente com suas classificações, são registradas em um arquivos de histórico chamado de `historico_chat.txt`.

Figura 4 – Acompanhamento chat e interação entre pessoas.



Fonte: Elaborado pelo autor (2024).

Além disso, o servidor também implementa mecanismos para banir e desbanir

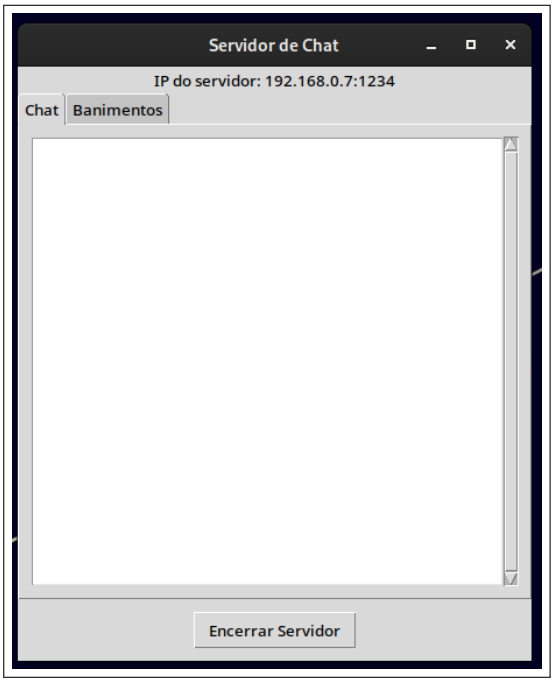
usuários, identificados por nome de usuário ao entrar. Eles são impedidos de se conectar através das funções específicas (`banir_usuario`) e em caso de um acordo entre o servidor, ele também pode vir a usar a função (`desbanir_usuario`), a fim de fazer com que o usuário respectivo, possa novamente se conectar ao servidor.

3.8 Interface gráfica inicial

A interface gráfica é desenvolvida através da biblioteca Tkinter, a qual oferece uma solução simples e eficiente para criar interfaces visuais em Python. Essa interface permite visualizar o bate-papo em tempo real, gerenciar usuários banidos e também encerrar o servidor. A janela principal exibe o fluxo de mensagens do chat, proporcionando uma visão clara e em tempo real das interações entre os usuários.

Além disso, o sistema inclui uma aba separada dedicada ao gerenciamento de banimentos, onde o administrador pode adicionar ou remover usuários da lista de banidos, visualizando as alterações instantaneamente. Conforme demonstrado na Figura adiante, a interface acompanha as atividades de maneira dinâmica, oferecendo atualizações em tempo real e facilitando a gestão do servidor, garantindo uma experiência de administração fluida e eficiente.

Figura 5 – Aba Inicial do Servidor-Chat.

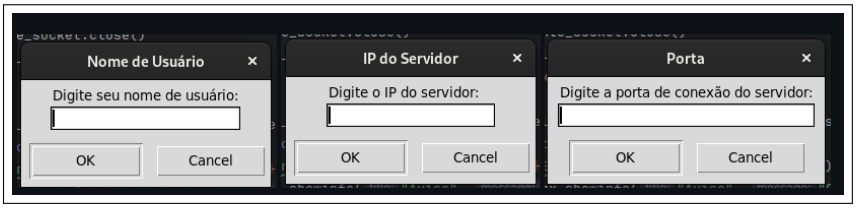


Fonte: Elaborado pelo autor (2024).

3.9 Método de conexão através do usuário ao servidor

O procedimento inicial se dá através de um cliente que conecta-se com um servidor dentro da rede interna ao qual esteja conectado pegando seu IP - Essa abreviação deriva do inglês Internet Protocol ou Protocolo de Internet, onde é definido como sendo um endereço exclusivo que identifica um dispositivo na Internet ou em uma rede local por exemplo. Ele consiste em um conjunto de regras que regem o formato de dados enviados pela Internet ou por uma rede local. Como mostrado na Figura 6, a conexão através do cliente com solicitação de 'Nome', 'IP' do servidor e a sua 'Porta' de conexão são requisitadas.

Figura 6 – Solicitação para Conexão ao Servidor.



Fonte: Elaborado pelo autor (2024).

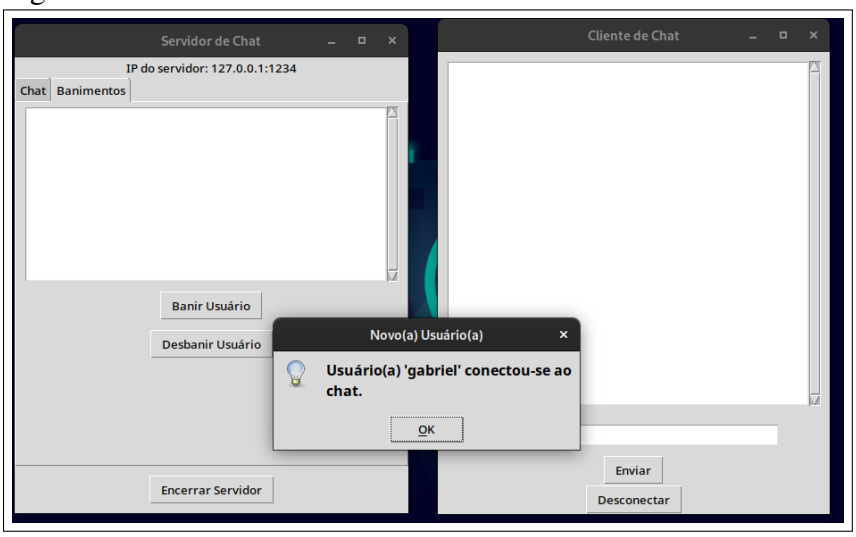
Em termos explicativos, solicita-se ao usuário, a porta - esta como sendo um ponto virtual onde começam e terminam as conexões de rede, estas são gerenciadas pelo sistema operacional de computador. Cada porta está associada a um processo ou serviço específico. As portas permitem que os computadores diferenciem facilmente entre diferentes tipos de tráfego, por exemplo: os e-mails vão para uma porta diferente daquela das páginas web, por exemplo, mesmo que ambas cheguem a um computador por meio da mesma conexão com a internet (SERVICES, 2024).

Depois de realizado o procedimento citado, é estabelecida a comunicação direta entre as pessoas presentes no referido chat, apelidados nos arquivos como 'cliente 1', 'cliente 2' e 'cliente 3' assim como é mostrado na figura 7, tornando-se possível a compreensão de que a conexão através do cliente ao servidor foi estabelecida conforme mostrado na figura 7.

3.10 Ambiente do cliente

Nesta sessão, é abordada como se da a conexão entre um cliente de chat, permitindo comunicação em tempo real entre usuários. O código também foi implementado em Python como foi anteriormente citado e faz uso das bibliotecas `socket`, `threading` e `tkinter` sendo este disponibilizado no mesmo repostitório (BARRETO, 2024a).

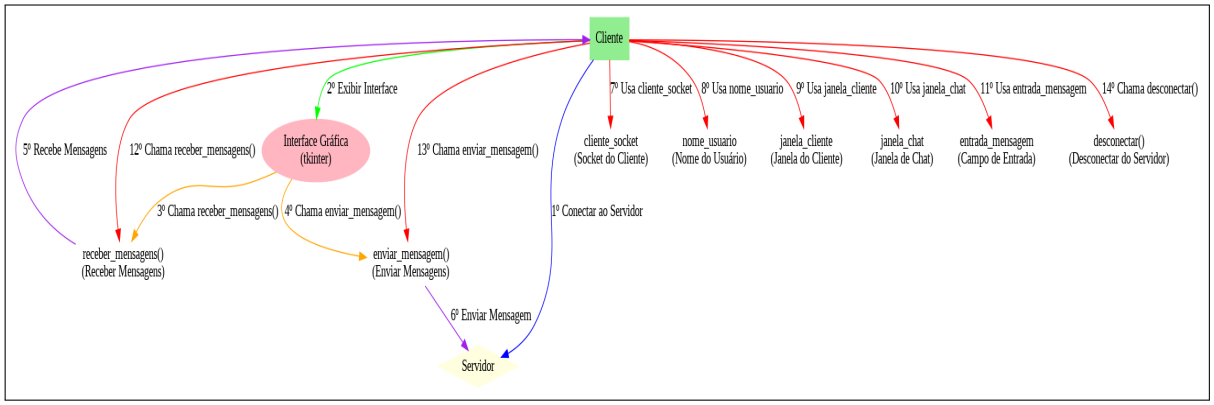
Figura 7 – Usuário conectado ao servidor.



Fonte: Elaborado pelo autor (2024).

Na figura 8, há uma descrição das partes do código do cliente, informando como funciona todo o processo de tráfego das mensagens no cenário proposto.

Figura 8 – Modelo de fluxo de dados do cliente.



Fonte: Elaborado pelo autor (2024).

A seguir nas subseções correspondentes a anterior, é abordado de maneira afunilada o que é tratado em cada função presente no cliente e como há a integração para com o servidor e o modelo de classificação de mensagem.

3.10.1 Importação de bibliotecas

O código necessita da importação das três seguintes bibliotecas principais para o seu funcionamento adequado:

a) *socket*: esta biblioteca fornece a interface para comunicação de rede em Python. O módulo `socket` permite criar sockets para enviar e receber dados através de redes. O uso de sockets

é fundamental para a comunicação entre cliente e servidor (FOUNDATION, 2024a). Em termos técnicos, um socket é um ponto final de uma conexão bidirecional em uma rede, que permite que as aplicações se comuniquem de forma síncrona;

b) *threading*: permite a execução simultânea de múltiplas tarefas dentro do mesmo programa. Com o `threading`, o cliente pode continuar recebendo mensagens enquanto o usuário interage com a interface gráfica. A programação multithreaded é crucial para melhorar a eficiência e a capacidade de resposta das aplicações (FOUNDATION, 2024b);

c) *tkinter*: biblioteca padrão para a criação de interfaces gráficas em Python. O `tkinter` é utilizado para criar janelas, botões, campos de entrada e outras partes da interface do usuário. É uma das bibliotecas GUI mais populares para Python devido à sua simplicidade e integração com a linguagem (FOUNDATION, 2024b).

3.10.2 Configuração inicial e conexão

Tem-se também a função `obter_nome_usuario()` que gerencia a configuração inicial e a conexão com o servidor solicitando ao usuário que insira seu nome, o IP e a porta do servidor através de caixas de diálogo (`simpledialog`). A entrada do usuário é crítica para a conexão, pois o nome identifica o cliente e o IP e a porta determinam o endereço do servidor.

a) *scrolledtext*: área de texto para exibir mensagens recebidas do servidor. O uso de uma área de texto com rolagem é essencial para permitir que o usuário visualize uma quantidade ilimitada de mensagens;

b) *entry*: campo onde o usuário digita suas mensagens;

c) botões para enviar mensagens e desconectar: estes botões têm funções associadas que permitem ao usuário interagir com o cliente de forma intuitiva.

Logo após os processos citados, inicializa-se uma *thread* separada para a função `receber_mensagens()`. Permitindo que o cliente continue operando de forma eficiente, recebendo e exibindo mensagens em paralelo com a interação do usuário.

3.10.3 Recepção de mensagens

Já a função `receber_mensagens()` é responsável por receber e processar mensagens do servidor com os seguintes processos:

a) a função executa um loop contínuo, utilizando `recv(1024)` para receber dados do servidor. O argumento `1024` especifica o tamanho do buffer de recepção, ou seja, o máximo de bytes a

serem lidos em uma única chamada;

b) as mensagens recebidas são decodificadas usando UTF-8. Essa codificação é amplamente utilizada por ser compatível com a maioria dos caracteres e símbolos;
c) se a mensagem contém o marcador `"encerrar"`, significa que um usuário está saindo da conversa. A função então exibe um aviso e encerra a conexã; e
d) caso contrário, a mensagem é inserida na área de texto da janela de chat, permitindo ao usuário visualizar as mensagens enviadas por outros participantes.

3.10.4 Envio de mensagens

Já a função `enviar_mensagem()` trata do envio de mensagens: O servidor verifica se o cliente está conectado antes de enviar uma mensagem. A função `send()` do socket é usada para enviar a mensagem ao servidor. Caso a conexão esteja inativa, o usuário é informado. Após isso, a mensagem pode ser inserida na área de texto do chat local para que o usuário possa ver o que enviou. Se a entrada estiver vazia, um aviso é exibido solicitando que o usuário insira uma mensagem.

3.10.5 Desconexão

A função `desconectar(nome_usuario)` lida com o processo de desconexão da seguinte maneira: Quando o cliente deseja sair, a função envia uma mensagem ao servidor informando que o usuário está saindo. Em seguida, a conexão é fechada com `cliente_socket.close()` e a interface gráfica é encerrada. Configura-se a janela para chamar a função de desconexão automaticamente ao ser fechada, garantindo que o cliente se desconecte de forma adequada e limpa.

3.10.6 Tratamento de exceções

A inclusão de blocos `try-except` permite o tratamento de erros e exceções durante a execução, fazendo ssim a captura e tratamento de erros relacionados à conexão com o servidor e à recepção de mensagens. O tratamento de exceções garante que o cliente forneça feedback adequado ao usuário e encerre a aplicação de forma controlada em caso de falhas. Esta abordagem garante que o cliente de chat seja robusto e confiável, oferecendo uma experiência de comunicação eficiente e amigável.

4 RESULTADOS

Primeiramente, vale ressaltar que os testes foram em um ambiente local, simulando uma conversa entre três usuários conectados ao servidor pelo IP 192.168.20.181, dentro do laboratório da Universidade Federal do Ceará, localizada em Itapajé - Ceará. A interação entre os usuários durou cerca de trinta minutos e envolveu a troca de mensagens sobre diversos assuntos, incluindo posicionamentos e palavras, algumas vezes camufladas com caracteres especiais.

Além disso, nos testes realizados, o modelo BERTimbau foi aplicado a um conjunto de dados contendo discursos de ódio em português, permitindo identificar com precisão mensagens ofensivas. A seguir, são apresentados os resultados do desempenho do modelo em diferentes configurações e épocas de treinamento, destacando seu comportamento em cenários variados.

4.1 Análise inicial

A Tabela 1 apresenta os resultados de desempenho do modelo 'gabrielbarreto-bert-large-finetuned-hatebr', utilizado para a identificação de discurso de ódio em português, ao longo de três épocas de treinamento. O modelo foi avaliado com base em quatro métricas principais: Acurácia, *F1-Score*, Precisão e *Recall*, além das perdas durante o treinamento e a validação.

Na primeira época, a acurácia alcançada foi de 0,885793, indicando que o modelo classificou corretamente cerca de 88,6% dos exemplos. A precisão foi de 0,815789, demonstrando que o modelo fez uma boa proporção de classificações corretas entre as positivas, mas o recall foi baixo, atingindo 0,113139, o que sugere dificuldade na identificação de todos os casos de discurso de ódio. O F1-Score, que pondera precisão e recall, ficou em 0,198718, refletindo o desequilíbrio entre essas métricas (GOMES, 2024).

Na segunda época, observou-se uma melhora significativa no desempenho, com a acurácia aumentando para 0,943353. A precisão subiu para 0,916667, mostrando que o modelo conseguiu manter uma alta proporção de classificações positivas corretas, enquanto o recall aumentou para 0,602190, o que indica uma melhora substancial na detecção de discursos de ódio. O F1-Score, por sua vez, subiu para 0,726872, refletindo esse melhor equilíbrio entre precisão e recall (GOMES, 2024).

Na terceira época, o modelo apresentou seu melhor desempenho, com uma acurácia de 0,964824. A precisão foi de 0,908714, e o recall aumentou para 0,799270, mostrando uma

melhoria significativa na capacidade de detectar corretamente os casos de discurso de ódio. O F1-Score final foi de 0,850485, consolidando o equilíbrio entre precisão e recall, o que torna o modelo altamente eficaz para aplicações de moderação de conteúdo, como o bate-papo em análise (GOMES, 2024).

Esses resultados indicam que o modelo ajustado atinge um desempenho robusto à medida que é treinado, especialmente em sua capacidade de classificar corretamente discursos de ódio e manter um equilíbrio adequado entre evitar falsos positivos e capturar os verdadeiros casos de discurso ofensivo. No entanto, apesar de apresentar bom desempenho, o modelo não foi utilizado devido ao objetivo de otimizar o processo de estudo e classificação das mensagens, que quanto testada em uma frase isoladamente, apresentava erro de classificação adequada.

Tabela 1 – Desempenho do modelo gabrielbarreto-bert-large-finetuned-hatebr por época.

Época	***Training Loss***	***Validation Loss***	**Acurácia**	***F1-Score***	**Precisão**	***Recall***
1	0.288300	0.303201	0.885793	0.198718	0.815789	0.113139
2	0.259100	0.162277	0.943353	0.726872	0.916667	0.602190
3	0.431500	0.131232	0.964824	0.850485	0.908714	0.799270

Fonte: (GOMES, 2024).

4.2 Resultados de treino de outros modelos de processamento de linguagem natural

A tabela 2 apresenta uma comparação de desempenho entre diferentes métodos de vetorização de texto (*Bag of Words* e TF-IDF) e modelos de redes neurais recorrentes (RNN, LSTM e GRU). Quatro métricas são comumente usadas para avaliar a eficácia dos métodos e modelos: *Accuracy* (acurácia), *Precision* (precisão), *Recall* e *F1-Score* (COLABORATORY, 2024).

O *Bag of Words* apresenta uma acurácia de 87,80%, precisão de 83,33%, recall de 87,80% e F1 score de 83,85%, indicando um bom desempenho geral em classificar os textos corretamente, especialmente na recuperação de dados (recall). TF-IDF tem uma acurácia ligeiramente menor, de 87,48%, com uma queda mais acentuada na precisão (76,53%). O F1 score também é menor, com 81,64%, sugerindo que o modelo TF-IDF é menos consistente na classificação correta em comparação com Bag of Words.

Entre os modelos de redes neurais, RNN apresenta o pior desempenho geral, com uma acurácia de 70,08% e um F1 score de 73,98%. O modelo LSTM supera os demais, com uma acurácia de 82,05% e métricas equilibradas de precisão, recall e F1 score, todos em torno de

82%, refletindo sua capacidade de capturar sequências temporais de forma mais eficiente. GRU também performa bem, com métricas próximas ao LSTM, mas levemente inferiores, com uma acurácia de 79,95% e F1 score de 80,68%.

No que diz respeito aos modelos BERT e BERTimbau, o modelo *bert-base-uncased* alcança uma acurácia de 88,00%, precisão de 80,82%, *recall* de 88,00% e *F1-Score* de 84,25%. O bert-base-cased apresenta uma acurácia superior, de 90,00%, precisão de 88,13%, *recall* de 90,00% e *F1-Score* de 89,05%. O *neuralmind/bert-base-portuguese-cased* se destaca ainda mais, com uma acurácia de 92,00%, precisão de 90,65%, *recall* de 92,00% e *F1-Score* de 89,37%. Esses resultados indicam que os modelos BERT e BERTimbau superam significativamente os métodos tradicionais de vetorização e os modelos de redes neurais recorrentes, destacando-se pela alta acurácia e *F1-Score*, refletindo uma eficácia superior na classificação de textos.

Durante esses testes fictícios, foram observados e registrados dados sobre a performance do sistema utilizando o interpretador do servidor, que foi executado no PyCharm. As métricas foram então calculadas e analisadas no Google Colab. A escolha do Google Colab foi necessária devido às limitações do hardware disponível para realizar testes com grandes volumes de dados em um ambiente local. Assim, a análise e o processamento das métricas puderam ser realizados de forma mais robusta, garantindo a confiabilidade dos resultados obtidos.

As mensagens gravadas no arquivo `'historico_chat.txt'` foram transportadas para o Google Colab, onde o processamento e a análise dos dados permitiram o cálculo preciso das métricas do modelo (COLABORATORY, 2024). A seguir, são apresentados os resultados das métricas geradas, juntamente com a comparação do desempenho do modelo utilizado com outros modelos empregados na área de inteligência artificial.

Tabela 2 – Comparação de desempenho entre métodos de vetorização, modelos de redes neurais e modelos baseados em BERT e BERTimbau.

Método/Modelo	**Acurácia**	**Precisão**	**Recall**	**F1 Score**
Bag of Words	0.878026	0.833370	0.878026	0.838570
TF-IDF	0.874829	0.765325	0.874829	0.816422
RNN	0.700777	0.800512	0.700777	0.739817
LSTM	0.820466	0.820810	0.820466	0.820015
GRU	0.799452	0.818090	0.799452	0.806804
bert-base-uncased	0.880000	0.808163	0.880000	0.842553
bert-base-cased	0.900000	0.881304	0.900000	0.890549
neuralmind/bert-base-portuguese-cased	0.920000	0.906531	0.920000	0.893702

Fonte: (COLABORATORY, 2024).

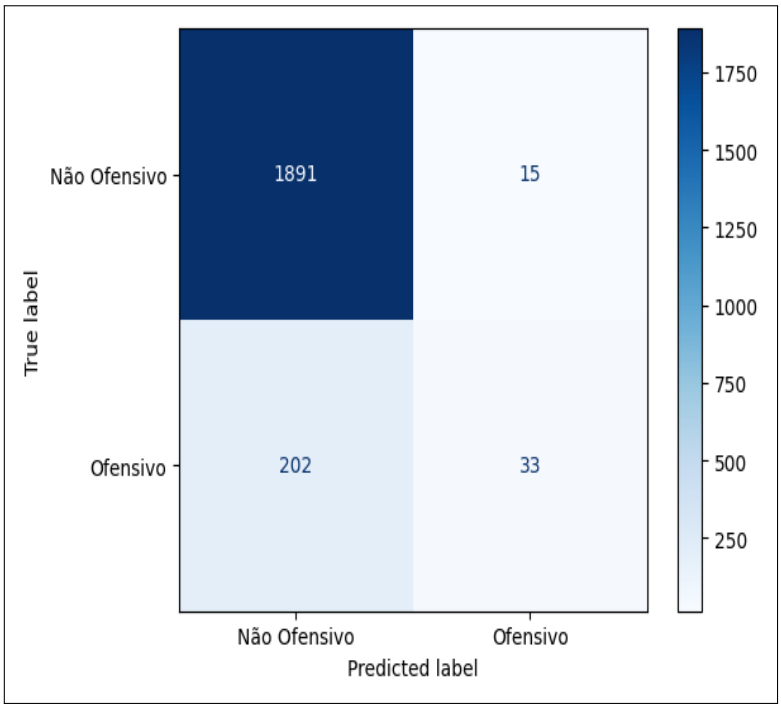
4.3 Matrizes de treino de outros modelos de processamento de linguagem natural

A seguir, apresentamos as matrizes de confusão, que proporcionam uma análise abrangente e detalhada das classificações obtidas, incluindo as categorias de verdadeiras positivas, verdadeiras negativas, falsas positivas e falsas negativas. Essa ferramenta analítica é de extrema importância no contexto da avaliação de modelos de linguagem natural, pois não apenas permite uma avaliação precisa da eficácia do modelo em distinguir entre as diferentes classes, mas também oferece insights valiosos sobre as limitações do modelo.

Por meio da interpretação cuidadosa das métricas fornecidas pela matriz de confusão, é possível identificar áreas específicas que podem exigir melhorias, além de orientar a formulação de estratégias para otimização do desempenho do modelo. Assim, as métricas extraídas estão em consonância com as informações apresentadas na tabela 2 acima, reforçando a relevância e a utilidade desta abordagem analítica na avaliação de desempenho de modelos de linguagem natural, contribuindo para a construção de sistemas mais robustos e eficazes em sua aplicação prática.

É possível notar as mensagens analisadas pelo dataset usado para análise sendo este o 'hate_speech18' quando ao método 'Bag Of Words':

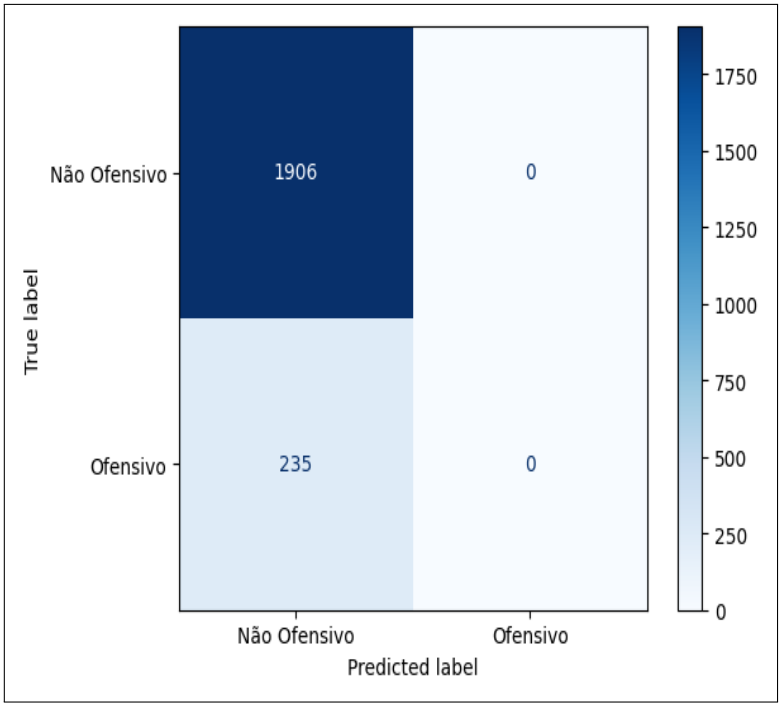
Figura 9 – Matriz de confusão quando ao modelo Bag Of Words.



Fonte: Elaborado pelo autor (2024).

A seguir é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’TF-IDF’:

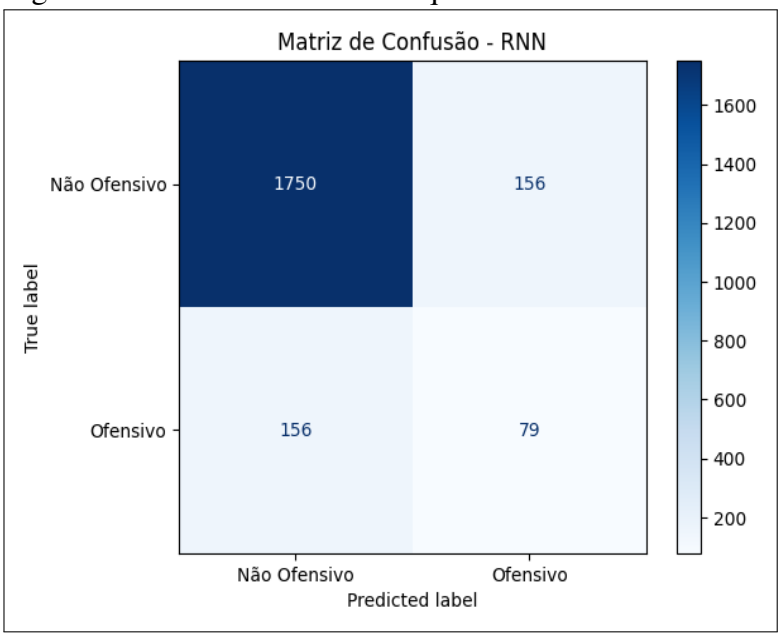
Figura 10 – Matriz de confusão quando ao modelo TF-IDF.



Fonte: Elaborado pelo autor (2024).

Na sequência é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’RNN’:

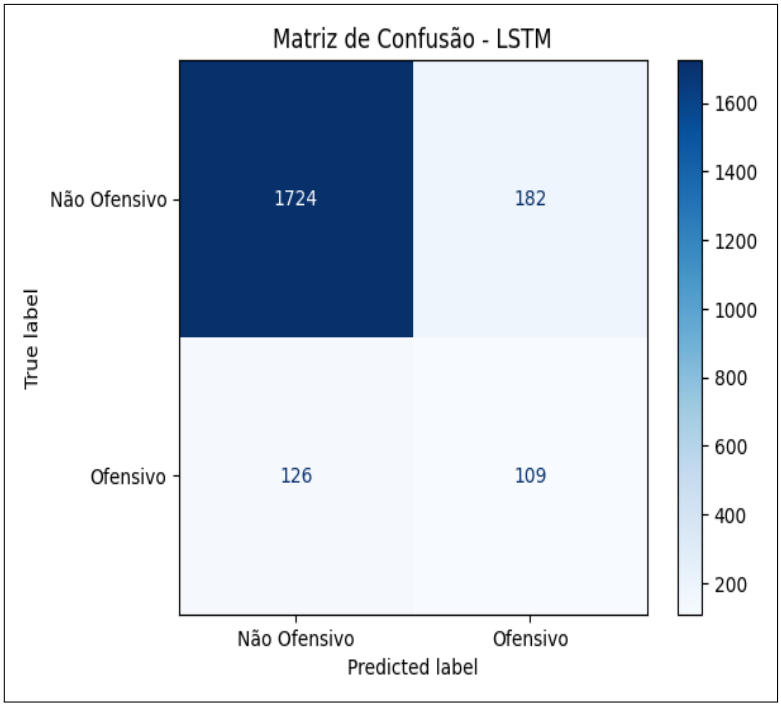
Figura 11 – Matriz de confusão quando ao modelo RNN.



Fonte: Elaborado pelo autor (2024).

Logo depois, é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’LSTM’:

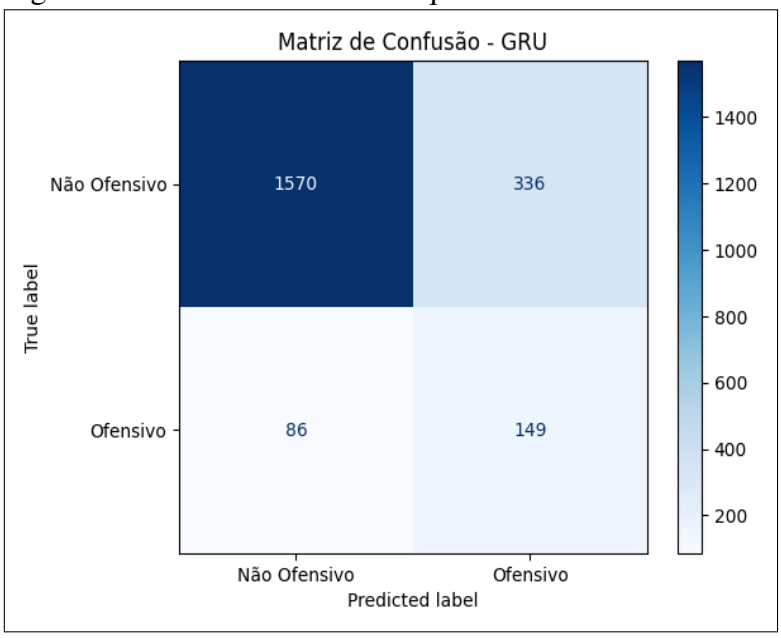
Figura 12 – Matriz de confusão quando ao modelo LSTM.



Fonte: Elaborado pelo autor (2024).

Além disso é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’GRU’:

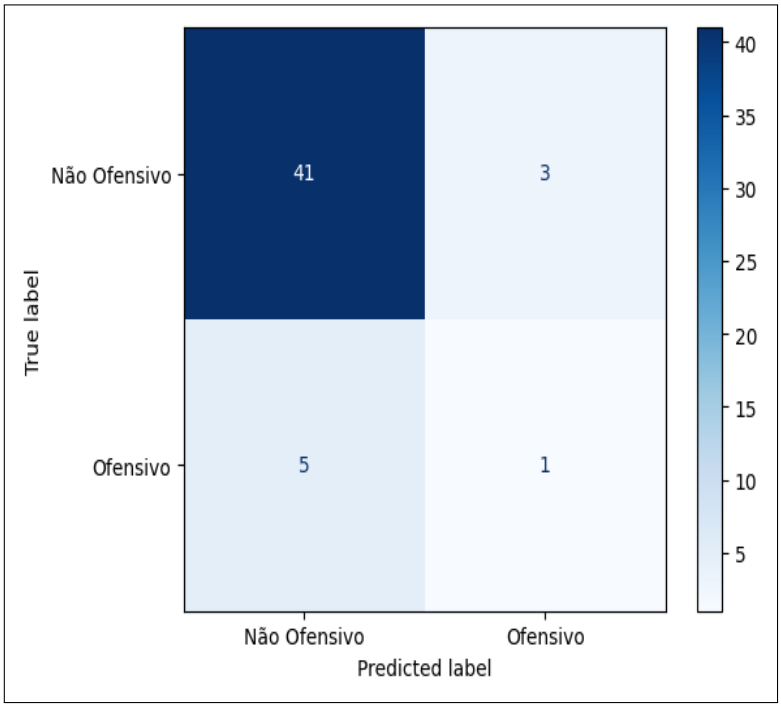
Figura 13 – Matriz de confusão quando ao modelo GRU.



Fonte: Elaborado pelo autor (2024).

A seguir é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’bert-uncased.’:

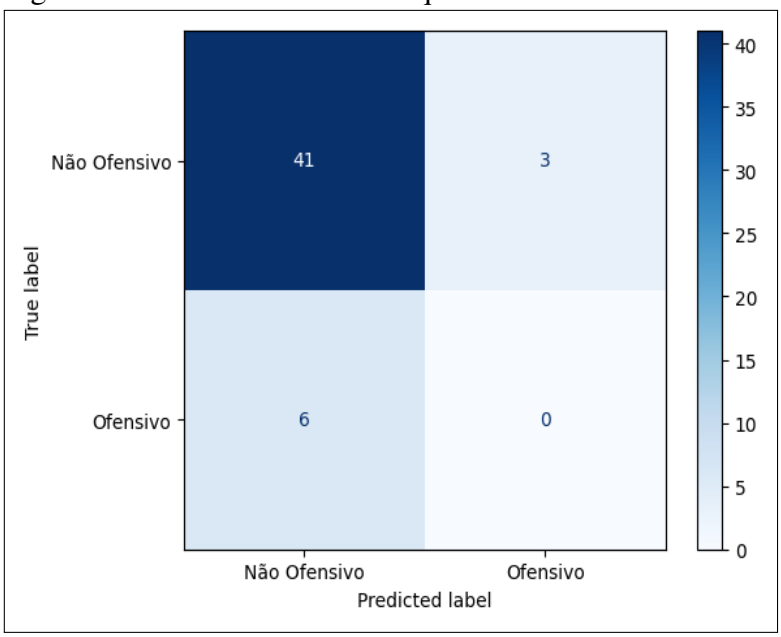
Figura 14 – Matriz de confusão quando ao modelo bert-uncased.



Fonte: Elaborado pelo autor (2024).

Adiante, é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’bert-cased.’:

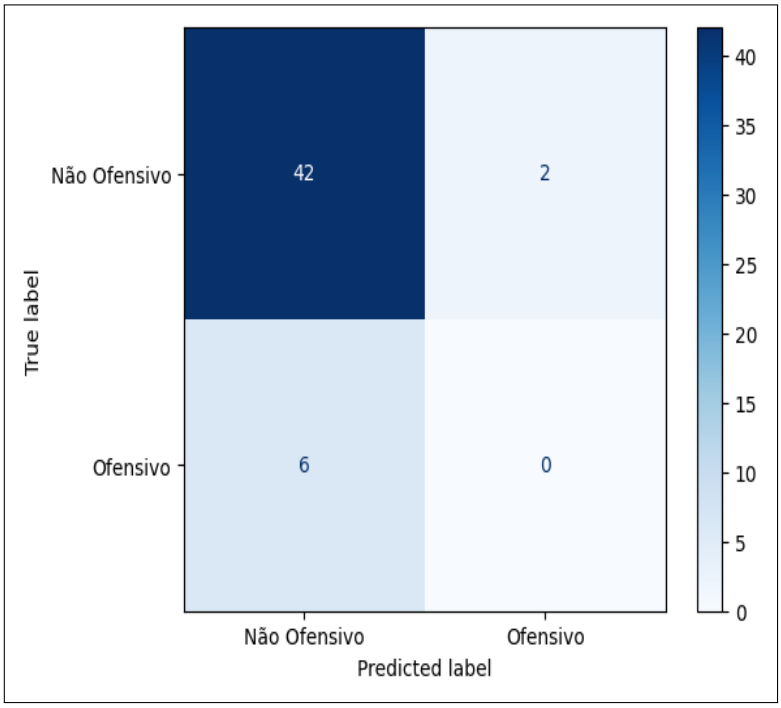
Figura 15 – Matriz de confusão quando ao modelo bert-cased.



Fonte: Elaborado pelo autor (2024).

Abaixo é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o 'hate_speech18' quando ao método 'neuralmind/bert-base-portuguese-cased.':

Figura 16 – Matriz de confusão quando ao modelo neuralmind/bert-base-portuguese-cased.



Fonte: Elaborado pelo autor (2024).

4.4 Resultados de treino com o modelo BERTimbau em apoio ao dataset Hate-BR

A tabela 3 apresenta os resultados de desempenho do modelo 'ruanchaves/bert-large-portuguese-cased-hatebr', utilizado para identificar discurso de ódio em português, por meio de quatro métricas importantes. A acurácia (*Accuracy*) do modelo é de 0,87, o que indica que ele classifica corretamente 87% dos exemplos. A precisão (*Precision*) é de 0,830444, refletindo a proporção de classificações positivas corretas em relação ao total de classificações positivas feitas pelo modelo. Já a revocação (Recall) é de 0,87, mostrando a capacidade do modelo de identificar corretamente os casos de discurso de ódio, ou seja, quantos casos foram identificados corretamente dentre todos os que deveriam ser. Por fim, o F1 Score, que combina precisão e recall, resulta em 0,841693, sendo uma média ponderada entre as duas métricas, útil especialmente quando há um equilíbrio entre falsas classificações positivas e negativas.

Esses resultados indicam que o modelo usado tem bom desempenho na identificação de discurso de ódio ou ofensivo, conseguindo manter um equilíbrio entre detectar corretamente

os casos de discurso de ódio (recall) e evitar classificações incorretas (precisão). Ressaltando assim ser vantajoso em aplicações de moderação de conteúdo como é o caso do bate-papo em análise.

Tabela 3 – Desempenho do modelo ruanchaves/bert-large-portuguese-cased-hatebr.

Modelo	**Acurácia**	**Precisão**	**Recall**	**F1 Score**
ruanchaves/bert-large-portuguese-cased-hatebr	0.870000	0.830444	0.870000	0.841693

Fonte: (COLAB, 2024).

4.5 Resultados fornecidos pelo código de identificação

A seguir estão os resultados do uso do chat interativo, abrangendo diversos discursos e posicionamentos. Os resultados apresentados revelam a eficiência do modelo na classificação de mensagens. O total de 235 mensagens analisadas resulta em 166 identificadas como não ofensivas e 69 como ofensivas, o que indica uma taxa de precisão considerável. O tempo total de classificação de 71.26 segundos e o tempo médio de 0.30 segundos por mensagem demonstram que o modelo opera de forma rápida e eficaz, o que é crucial para aplicações em tempo real.

Tabela 4 – Métricas base geradas a partir do código.

Identificação das Mensagens	**Resultado**
Número de mensagens classificadas	235
Número de mensagens não ofensivas	166
Número de mensagens ofensivas	69
Tempo total de classificação	71.26s
Tempo médio de classificação	0.30s

Fonte: Elaborado pelo autor (2024).

4.5.1 Análise das Métricas

a) número de mensagens classificadas: um total de 235 mensagens foi processado, refletindo um volume considerável de dados para avaliação do modelo. Essa quantidade é representativa de interações típicas em um ambiente de chat, permitindo uma avaliação mais precisa do desempenho em situações reais;
b) mensagens não ofensivas e ofensivas: a distribuição das mensagens indica que aproximadamente 29,4% das mensagens foram classificadas como ofensivas. Este equilíbrio é

importante, pois um modelo eficaz deve conseguir discernir entre mensagens ofensivas e não ofensivas com precisão. A taxa de mensagens ofensivas sugere que o modelo não apenas é sensível a ofensas, mas também é capaz de filtrar interações normais, o que é essencial para evitar classificações errôneas que poderiam impactar a experiência do usuário;

c) tempo de classificação: o tempo médio de 0.30 segundos por mensagem sugere que o modelo não só é preciso, mas também rápido, permitindo sua aplicação em cenários interativos onde a velocidade de resposta é fundamental. Essa rapidez é crucial em ambientes de chat, onde atrasos na resposta podem prejudicar a comunicação e a satisfação do usuário;

Esses resultados sugerem que o modelo se destaca em eficiência e precisão para a classificação de discurso de ódio, sendo comparável a outros classificadores estabelecidos na literatura. O desempenho positivo nas métricas indica que o sistema pode ser uma solução viável para aplicações em tempo real, contribuindo para a moderação e análise de conteúdos em plataformas de comunicação.

4.6 Resultados fornecidos pelo código de treinamento do modelo escolhido.

Tabela 5 – Desempenho do sistema protótipo baseado em Bertimbau.

Modelo	**Acurácia**	**Precisão**	**Recall**	**F1 Score**
Sistema protótipo baseado em Bertimbau	0.705882	0.750000	0.818182	0.782609

Fonte: (COLABORATORY, 2024).

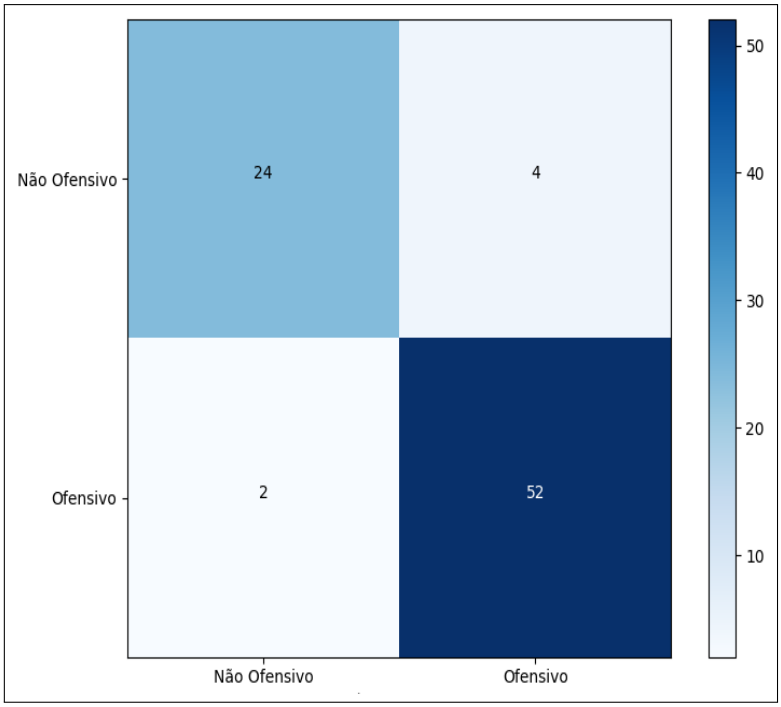
Os testes foram realizados ininterruptamente em um ambiente local, simulando uma conversa entre três usuários conectados ao servidor pelo IP 192.168.20.181, conforme demonstrado no vídeo (BARRETO, 2024b). Durante aproximadamente trinta minutos, os usuários trocaram mensagens sobre diversos assuntos, incluindo posicionamentos e expressões disfarçadas com caracteres especiais.

4.7 Matriz gerada pelo código de treinamento do modelo escolhido.

Nessa seção, será abordada a matriz de confusão para as mensagens classificadas, as quais foram importadas do arquivo historico_chat.txt, gerado pelo servidor. Essas mensagens refletem as trocas entre os usuários do chat durante o teste ininterrupto de 30 (trinta) minutos.

Logo em seguida é possível notar as mensagens analisadas pelo dataset usado para análise sendo este o ’hate_speech18’ quando ao método ’neuralmind/bert-base-portuguese-cased.’:

Figura 17 – Matriz de confusão quando ao modelo escolhido.



Fonte: Elaborado pelo autor (2024).

4.7.1 Comparação de Desempenho

Durante a implementação, as limitações de hardware disponíveis levaram à decisão de utilizar o Google Colab para o processamento de dados em larga escala e para o treinamento do modelo. No ambiente local, foram realizadas as etapas iniciais de implementação do código e testes de desempenho com volumes de dados reduzidos. No Colab, foram realizados os treinamentos mais intensivos, utilizando GPUs para lidar com grandes volumes de dados e acelerar o processo de fine-tuning do modelo BERTimbau. Essa abordagem híbrida otimizou o tempo e garantiu que o modelo fosse treinado com a infraestrutura necessária, mantendo a eficiência no processamento de dados.

As métricas de desempenho, apresentadas na tabela, revelam que o modelo atingiu uma acurácia de aproximadamente 70.59%, o que indica um desempenho aceitável na tarefa de classificação de discurso de ódio.

a) precisão de 75%: significa que, das mensagens identificadas como ofensivas, 75% realmente

eram ofensivas. Isso é um indicador positivo de que o modelo não classifica incorretamente muitas mensagens não ofensivas como ofensivas, minimizando o risco de alarmar os usuários desnecessariamente. Uma alta precisão é fundamental em aplicações onde a falsificação de mensagens pode levar a consequências indesejadas, como a censura indevida de interações legítimas;

b) *recall* de 81.82%: refere-se à capacidade do modelo de identificar corretamente as mensagens ofensivas entre todas as mensagens realmente ofensivas. Um *recall* alto sugere que o modelo é eficaz em capturar a maioria das mensagens ofensivas, o que é essencial para a moderação eficiente de conteúdo. A capacidade de detectar a maioria das ofensas é crítica em sistemas que buscam reduzir a presença de discurso de ódio e proteger os usuários de interações prejudiciais;

c) *f1-score* de 78.26%: este valor balanceia a precisão e o *recall*, oferecendo uma visão geral do desempenho do modelo. Um *F1-Score* acima de 70% é geralmente considerado bom para tarefas de classificação, indicando que o modelo é bem equilibrado em sua capacidade de identificar mensagens ofensivas sem comprometer a inclusão de mensagens não ofensivas. A interpretação do *F1-Score* é particularmente importante em aplicações onde tanto a detecção de discurso ofensivo quanto a minimização de falsos positivos são cruciais para a eficácia do sistema;

O cálculo das métricas foi realizado utilizando o ambiente Google Colab, dada a limitação de poder computacional disponível para realizar testes com grandes volumes de dados, garantindo uma análise robusta dos resultados. Este ambiente possibilitou a execução de experimentos em um cenário controlado, onde o modelo pôde ser avaliado de forma abrangente, garantindo a confiabilidade das métricas apresentadas (COLAB, 2024). Esses resultados indicam que o sistema protótipo baseado em Bertimbau é um passo significativo em direção ao desenvolvimento de ferramentas eficazes para a detecção e moderação de discurso de ódio em ambientes de comunicação digital, mostrando-se competitivo em comparação com outros modelos na área de inteligência artificial.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho se dedicou a explorar as intersecções entre tecnologia, legislação e a responsabilidade social das plataformas digitais, buscando contribuir para um debate mais fundamentado sobre a proteção dos direitos humanos no ambiente digital. A análise das métricas apresentadas na seção de resultados demonstrou que os algoritmos atualmente disponíveis atingem um desempenho mediano na classificação entre "É discurso ofensivo"e "Não é discurso ofensivo". No entanto, ao lidar com frases mais complexas que exigem um entendimento contextual aprofundado, os algoritmos falham em captar nuances essenciais. Observou-se que termos frequentemente utilizados para desqualificar indivíduos podem eclipsar o contexto geral, resultando em classificações inadequadas e uma percepção distorcida da comunicação.

Essa inadequação destaca a necessidade de uma abordagem mais refinada na identificação de discursos potencialmente ofensivos, enfatizando a responsabilidade das plataformas em proporcionar uma moderação mais justa e precisa. Assim, a criação de um novo modelo de aprendizagem que integre a precisão na identificação de discursos e a comunicação efetiva com os usuários torna-se crucial. Um modelo aprimorado não apenas promoveria um ambiente digital mais saudável e respeitoso, mas também reforçaria a responsabilidade das plataformas na proteção dos direitos humanos.

Além disso, é vital ressaltar a importância de estudos futuros que avancem além da mera análise dos tipos de discursos. Um dos trabalhos futuros poderá focar na busca ativa por voluntários para contribuir com a coleta e validação de dados, além da incorporação de gírias regionais, como as do Ceará, na construção e refinamento de um novo *dataset*. Essa abordagem permitiria uma adaptação mais eficaz do modelo às particularidades culturais e linguísticas, melhorando a identificação de discursos ofensivos em contextos diversos.

A utilização de modelos multimodais, que combinam texto, imagem e vídeo, também se apresenta como uma estratégia promissora para enriquecer a análise das interações online, proporcionando uma compreensão mais holística das dinâmicas de comunicação nas plataformas digitais.

Portanto, objetiva-se contribuir para a construção de um espaço digital mais acolhedor e inclusivo, essencial para aqueles que utilizam as mídias sociais como meio de expressão e interação. A continuidade desse debate e a implementação de soluções inovadoras são fundamentais para garantir que a tecnologia e as políticas sociais avancem em conjunto em prol do respeito e da dignidade humana.

Conclui-se que a implementação de um sistema automatizado para a detecção de discurso de ódio em chats interativos é uma ferramenta poderosa para a promoção de um ambiente digital mais seguro. No entanto, é essencial que futuras pesquisas continuem a explorar as nuances linguísticas e sociais que caracterizam o discurso de ódio, a fim de refinar ainda mais as técnicas de moderação e garantir que as interações online sejam respeitosas e inclusivas.

REFERÊNCIAS

ARQSERV. **Como Funciona a Arquitetura Cliente-Servidor**. 2012. Disponível em: https://arqserv.wordpress.com/2012/03/17/como-funciona-a-arquitetura-cliente-servidor/. Acesso em: 4 set. 2024.

BARRETO, G. **Identificador de Discurso de Ódio com BERTimbau**. 2024. Disponível em: https://github.com/GabrielBarreto01/IdentificadorDiscursoOdioComBERTimbau. Acesso em: 4 set. 2024.

BARRETO, G. **Teste do modelo de análise de linguagem natural (ruanchaves/bert-large-portuguese-cased-hatebr)**. 2024. Disponível em: https://www.youtube.com/watch?v=2rlTTOlTqn4. Acesso em: 19 set. 2024.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. New York: Springer, 2006. Disponível em: https://link.springer.com/book/10.1007/978-0-387-46465-6. Acesso em: 15 set. 2024.

BONAVENTURE, O. **Computer Networking:** Principles, Protocols and Practice. 2nd. ed. [S.l.]: Open-Source Textbook, 2021. Disponível em: https://github.com/obonaventure/cnp3. Acesso em: 4 set. 2024.

COLAB, G. **Identificador de Discurso de Ódio**. 2024. Disponível em: https://colab.research.google.com/drive/10MplCj9zX9mXIHYwOzF50mqjbiMcsXA4?usp=sharing. Acesso em: 16 set. 2024.

COLABORATORY, G. **Notebook Colab:** Classificação de Frases. 2024. Disponível em: https://colab.research.google.com/drive/1qvyKPfbTwci$_V$*I*0*CHnV*$_B$*ZDbpZSfwuk*8?*usp* = *sharing*. Acesso em: 16 set. 2024.

FACE, H. **Transformers Documentation**. [S.l.], 2024. Disponível em: https://huggingface.co/transformers/. Acesso em: 4 set. 2024.

FOROUZAN, B. A. **Data Communications and Networking**. 5th. ed. [S.l.]: McGraw-Hill, 2012. Capítulo 3: Camadas de Rede e Conectividade. Disponível em: https://www.mhhe.com/engcs/compsci/forouzan/datacommunications/. Acesso em: 4 set. 2024.

FOUNDATION, P. S. **Python Socket Library**. [S.l.], 2024. Disponível em: https://docs.python.org/3/library/socket.html. Acesso em: 4 set. 2024.

FOUNDATION, P. S. **Python Threading Library**. [S.l.], 2024. Disponível em: https://docs.python.org/3/library/threading.html. Acesso em: 4 set. 2024.

GILLESPIE, T. **Custodians of the Internet:** Platforms, Content Moderation, and the Hidden Decisions That Shape Social Media. [S.l.]: Yale University Press, 2021. Disponível em: https://yalebooks.yale.edu/book/9780300252202/custodians-of-the-internet. Acesso em: 16 set. 2024.

GOGONI, R. **O que é algoritmo?** 2023. Disponível em: https://tecnoblog.net/responde/o-que-e-algoritmo/. Acesso em: 12 nov. 2023.

GOMES, F. G. B. **Análise de Discurso de Ódio em Chats Interativos**. 2024. Disponível em: <https://colab.research.google.com/drive/1OoM6uVPVjKEX6J04Uoej6KgdiI9rIsB9#scrollTo=HdYOYEFBkxtz>. Acesso em: 26 set. 2024.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge: MIT Press, 2016. Disponível em: https://www.deeplearningbook.org/. Acesso em: 15 set. 2024.

JETBRAINS. **PyCharm: IDE para Python**. 2024. Disponível em: https://www.jetbrains.com/pycharm/. Acesso em: 4 set. 2024.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing:** An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2. ed. [S.l.]: Prentice Hall, 2008. Disponível em: https://web.stanford.edu/ jurafsky/slp2/. Acesso em: 2 . 2024.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 3rd. ed. [S.l.]: Pearson, 2021. Disponível em: https://web.stanford.edu/ jurafsky/slp3/. Acesso em: 15 set. 2024.

KUROSE, J. F.; ROSS, K. W. **Computer Networking:** A Top-Down Approach. 7th. ed. [S.l.]: Pearson, 2017. Capítulo 2: Camada de Aplicação. Disponível em: https://www.amazon.com.br/Computer-Networking-Top-Down-James-Kurose/dp/9356061319. Acesso em: 4 set. 2024.

LIVERPOOL, U. of. **Why Artificial Intelligence Is So Important in Today's World**. 2024. Disponível em: https://online.liverpool.ac.uk/why-artificial-intelligence-is-so-important-in-todays-world/. Acesso em: 4 set. 2024.

LLANSO, E.; HOBOKEN, J. V. Ai, algorithms, and the struggle for control over online content moderation. **Institute for Information Law (IViR)**, 2020. Disponível em: https://www.ivir.nl/publicaties/download/AI-Llanso-Van-Hoboken-Feb-2020.pdf. Acesso em: 27 set. 2024.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. **Introduction to Information Retrieval**. Cambridge: Cambridge University Press, 2008. Disponível em: https://nlp.stanford.edu/IR-book/. Acesso em: 15 set. 2024.

MATIAS, J. N.; ALMEIDA, S. M.; COSTA, T. F. Protecting online users: Effective strategies for moderation and education. **Cyberpsychology Journal**, v. 20, n. 1, p. 45–61, 2022. Acesso em: 16 set. 2024.

NOBATA, C.; TETREAULT, J.; THET, T.; CHOI, Y. Abusive language detection in online user content. In: **Proceedings of the 25th International Conference on World Wide Web**. [S.l.]: International World Wide Web Conferences Steering Committee, 2016. p. 145–153. Disponível em: https://dl.acm.org/doi/10.1145/2872427.2883081. Acesso em: 18 set. 2024.

REPúBLICA, P. da. **Lei nº 12.965, de 23 de abril de 2014**. 2014. Disponível em: https://www.planalto.gov.br/ccivil$_0$3/$_a$*to*2011 − 2014/2014/*lei*/*l*12965.*htm*. Acesso em: 6 set. 2024.

REPúBLICA, P. da. **Lei n.º 13.709, de 14 de agosto de 2018**. 2018. Disponível em: https://www.planalto.gov.br/ccivil$_0$3/$_a$*to*2015 − 2018/2018/*lei*/*l*13709.*htm*. Acesso em: 4 set. 2024.

RIBEIRO, M. H.; SANTOS, C. R.; SILVA, A. L. Moderating hate speech in online platforms: Approaches and challenges. **Journal of Digital Communication**, v. 15, n. 2, p. 123–139, 2023. Disponível em: https://www.journalofdigitalcommunication.com/article/view/123. Acesso em: 16 set. 2024.

RODRIGUES, R. C.; TANTI, M.; AGERRI, R. **Evaluation of Portuguese Language Models**. 2023. Disponível em: https://github.com/ruanchaves/eplm. Acesso em: 6 set. 2024.

RUSSELL, S.; NORVIG, P. Artificial intelligence: A modern approach, 2/e. **Prentice Hall series in artificial intelligence**, p. 31–52, 2003. Disponível em: https://www.pearson.com/store/p/artificial-intelligence-a-modern-approach/P100000671272. Acesso em: 2 set. 2024.

SCHMIDT, A.; WIEGAND, M. A survey on hate speech detection using natural language processing. In: **Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media**. [S.l.: s.n.], 2017. p. 1–10. Disponível em: https://aclanthology.org/W17-3001. Acesso em: 18 set. 2024.

SERVICES phoenixNAP I. What is a network port? **PhoenixNAP Knowledge Base**, 2024. Disponível em: https://phoenixnap.com/kb/what-is-a-network-port. Acesso em: 17 set. 2024.

SINGER, P. W.; FRIEDMAN, A. **Cybersecurity and Cyberwar:** What Everyone Needs to Know. New York: Oxford University Press, 2014. Disponível em: https://global.oup.com/academic/product/cybersecurity-and-cyberwar-9780190213652. Acesso em: 16 set. 2024.

SOUZA, M.; OLIVEIRA, F.; GOMES, R. Aplicação de bertimbau para detecção de discurso de Ódio em redes sociais. **Revista Brasileira de Inteligência Artificial**, v. 23, n. 3, p. 45–62, 2022. Disponível em: https://www.researchgate.net/publication/363217132/$Aplicacao_{d}e_{B}ERTimbau_{p}ara_{D}eteccao_{d}e_{D}isco_{d}e_{O}dio_{e}m_{R}edes_{S}ociais$. Acesso em: 19 set. 2024.

TANENBAUM, A. S.; WETHERALL, D. J. **Computer Networks**. 5th. ed. [S.l.]: Prentice Hall, 2010. Capítulo 1: Introdução às Redes. Disponível em: https://www.amazon.com/Computer-Networks-Andrew-S-Tanenbaum/dp/0132126958. Acesso em: 4 set. 2024.

TAVARES, L. A.; MEIRA, M. C.; AMARAL, S. F. do. Inteligência artificial na educação: Survey. **Brazilian Journal of Development**, v. 6, n. 7, p. 48699–48714, 2020. Disponível em: https://www.brazilianjournalofdevelopment.com/index.php/BJD/article/view/439. Acesso em: 2 set. 2024.

TUFEKCI, Z. **Twitter and Tear Gas:** The Power and Fragility of Networked Protest. [S.l.]: Yale University Press, 2018. Disponível em: https://yalebooks.yale.edu/book/9780300234085/twitter-and-tear-gas. Acesso em: 16 set. 2024.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER Łukasz; POLOSUKHIN, I. Attention is all you need. In: **Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)**. [S.l.: s.n.], 2017. Disponível em: https://arxiv.org/abs/1706.03762. Acesso em: 16 set. 2024.

VIRTANEN, P. *et al.* Scipy 1.0: Fundamental algorithms for scientific computing in python. **Nature Methods**, Nature Publishing Group, v. 17, n. 3, p. 261–272, 2020. Disponível em: https://www.nature.com/articles/s41592-019-0686-2. Acesso em: 4 set. 2024.

APÊNDICE A – CÓDIGOS-FONTES UTILIZADOS DO SERVIDOR E CLIENTE

Código-fonte 1 – Código em Python do Servidor de Chat

```
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext, messagebox, ttk,
    simpledialog
from datetime import datetime
from transformers import AutoModelForSequenceClassification
    , AutoTokenizer, AutoConfig
import numpy as np
import torch
from scipy.special import softmax
import time

MAX_USUARIOS = 3  # N mero m ximo de usu rios que podem
    se conectar ao servidor
clientes_banidos = []  # Lista de IPs banidos
clientes = []

# Vari veis para m tricas
num_mensagens_classificadas = 0
num_ofensivas = 0
num_nao_ofensivas = 0
tempos_classificacao = []

# Carregando modelo de classifica  o de discurso ofensivo
model_name = "ruanchaves/bert-large-portuguese-cased-hatebr
    "
model = AutoModelForSequenceClassification.from_pretrained(
    model_name)
```

```
25 tokenizer = AutoTokenizer.from_pretrained(model_name)
26 config = AutoConfig.from_pretrained(model_name)
27
28 def classify_text(text):
29     global num_mensagens_classificadas, num_ofensivas,
       num_nao_ofensivas, tempos_classificacao
30
31     if model is None or tokenizer is None or config is None
       :
32         return "Erro: Modelo n o carregado."
33
34     start_time = time.time()
35     model_input = tokenizer(text, padding=True,
       return_tensors="pt")
36
37     with torch.no_grad():
38         output = model(**model_input)
39         scores = output.logits[0].detach().numpy()
40         scores = softmax(scores)
41
42         predicted_class = np.argmax(scores)
43         label = config.id2label[predicted_class]
44
45         end_time = time.time()
46         tempos_classificacao.append(end_time - start_time)
47
48         num_mensagens_classificadas += 1
49         if label == 0:
50             num_nao_ofensivas += 1
51             return "N o     discurso Ofensivo"
52         elif label == 1:
53             num_ofensivas += 1
```

```
54              return "   discurso Ofensivo"
55          else:
56              return "Classifica  o  inv lida"
57
58  def receber_mensagens(client_socket, username,
        client_address):
59      while True:
60          try:
61              data = client_socket.recv(1024)
62              if not data:
63                  break
64              mensagem = data.decode('utf-8')
65
66              # Classificar o tipo de discurso (ofensivo ou
                    n  o)
67              classificacao = classify_text(mensagem)
68
69              # Atualiza a janela de chat com a mensagem do
                    usu rio e sua classifica  o
70              chat_window.config(state=tk.NORMAL)
71              chat_window.insert(tk.END, f"{username}: {
                    mensagem}\n")
72              chat_window.insert(tk.END, f"Classifica  o: {
                    classificacao}\n\n")
73              chat_window.see(tk.END)
74              chat_window.config(state=tk.DISABLED)
75
76              # Registra a mensagem no arquivo hist rico com
                     a classifica  o
77              log_message(f"{username}: {mensagem} -
                    Classifica  o: {classificacao}")
78
```

```
79              # Envia a mensagem recebida para todos os
                    outros usu rios
80              enviar_para_todos(f"{username}: {mensagem}",
                    client_socket)
81          except Exception as e:
82              print(f"Erro na recep  o de mensagens de {
                    username}: {e}")
83              break
84
85      client_socket.close()
86      remover_cliente(client_socket, username)
87
88  # Resto do c digo do servidor...
89
90  def enviar_para_todos(mensagem, remetente_socket):
91      for cliente, user, addr in clientes:
92          if cliente != remetente_socket:
93              try:
94                  cliente.send(mensagem.encode('utf-8'))
95              except Exception as e:
96                  print(f"Erro ao enviar mensagem para {user
                        }: {e}")
97                  remover_cliente(cliente, user)
98
99
100 def remover_cliente(client_socket, username):
101     global clientes
102     clientes = [(cliente, user, addr) for cliente, user,
            addr in clientes if cliente != client_socket]
103     log_message(f"{username} desconectado")
104     print(f"{username} desconectado")
105
```

```
106
107 def log_message(message):
108     with open("historico_chat.txt", "a") as f:
109         f.write(f"{datetime.now().strftime('%Y-%m-%d %H:%M
            :%S')} - {message}\n")
110
111
112 def enviar_mensagem_boas_vindas(client_socket, username):
113     mensagem_boas_vindas = f"Bem-vindo(a), {username}!"
114     client_socket.send(mensagem_boas_vindas.encode('utf-8')
        )
115     log_message(mensagem_boas_vindas)
116
117
118 def notificar_novo_usuario(username):
119     messagebox.showinfo("Novo(a) Usu rio(a)", f"Usu rio(a
        ) '{username}' conectou-se ao chat.")
120     log_message(f"Usu rio(a) '{username}' conectou-se ao
        chat.")
121
122
123 def banir_usuario():
124     username_or_ip = simpledialog.askstring("Banir Usu rio
        (a)", "Digite o nome de usu rio ou IP do(a)
        usu rio(a) a ser banido(a):")
125     if username_or_ip:
126         motivo = simpledialog.askstring("Banir Usu rio(a)"
            , "Digite o motivo do banimento:")
127         if motivo:
128             # Verifica se o input    um nome de usu rio
129             for i, (cliente, user, addr) in enumerate(
                clientes):
```

```
130                 if user == username_or_ip:
131                     clientes.pop(i)
132                     clientes_banidos.append(addr[0])
133                     cliente.send(f"Voc  foi banido do chat
                           . Motivo: {motivo}".encode('utf-8'))
134                     cliente.close()
135                     ban_attempts_text.insert(tk.END, f"O
                           usu rio '{username_or_ip}' foi
                           banido do chat. Motivo: {motivo}\n")
136                     log_message(f"Usu rio '{username_or_ip
                           }' foi banido do chat. Motivo: {
                           motivo}")
137                     return
138             # Se n o for um nome de usu rio, assume que
                      um IP e realiza o banimento
139             banir_ip(username_or_ip, motivo)
140
141
142 def banir_ip(ip_address, motivo):
143     if ip_address not in clientes_banidos:
144         clientes_banidos.append(ip_address)
145         ban_attempts_text.insert(tk.END, f"O IP '{
               ip_address}' foi banido do chat. Motivo: {motivo
               }\n")
146         log_message(f"IP '{ip_address}' foi banido do chat.
                Motivo: {motivo}")
147
148
149 def desbanir_usuario():
150     username_or_ip = simpledialog.askstring("Desbanir
           Usu rio(a)", "Digite o nome de usu rio ou IP do(a)
            usu rio(a) a ser desbanido(a):")
```

```
151     if username_or_ip:
152         desbanir_ip(username_or_ip)
153 
154 
155 def desbanir_ip(ip_address):
156     if ip_address in clientes_banidos:
157         clientes_banidos.remove(ip_address)
158         ban_attempts_text.insert(tk.END, f"O IP '{
               ip_address}' foi desbanido do chat.\n")
159         log_message(f"IP '{ip_address}' foi desbanido do
               chat.")
160 
161 
162 def iniciar_servidor():
163     global socket_servidor, clientes
164     host = '192.168.20.181'
165     porta = 1234
166 
167     socket_servidor = socket.socket(socket.AF_INET, socket.
           SOCK_STREAM)
168     socket_servidor.setsockopt(socket.SOL_SOCKET, socket.
           SO_REUSEADDR, 1)
169     socket_servidor.bind((host, porta))
170     socket_servidor.listen(MAX_USUARIOS)  # Definindo o
           n mero  m ximo de conex es pendentes
171 
172     ip_label.config(text=f"IP do servidor: {host}:{porta}")
173 
174     print(f"Servidor de chat iniciado em {host}:{porta}")
175 
176     while True:
```

```
177         socket_cliente, client_address = socket_servidor.
               accept()
178
179         # Verifica se o IP est  banido
180         if client_address[0] in clientes_banidos:
181             tentativa_mensagem = f"IP banido {
                   client_address[0]} tentou se conectar."
182             socket_cliente.send("Voc  est  banido do chat
                   .".encode('utf-8'))
183             socket_cliente.close()
184             ban_attempts_text.insert(tk.END, f"{
                   tentativa_mensagem}\n")
185             log_message(tentativa_mensagem)
186             continue
187
188         # Verifica se o n mero m ximo de usu rios foi
               atingido
189         if len(clientes) >= MAX_USUARIOS:
190             socket_cliente.send("N mero m ximo de
                   usu rios atingido. Tente novamente mais
                   tarde.".encode('utf-8'))
191             socket_cliente.close()
192             continue
193
194         username = socket_cliente.recv(1024).decode('utf-8'
               ).lower()  # Convertendo para min sculas
195
196         if username in [user[1] for cliente, user, addr in
               clientes]:
197             socket_cliente.send("O nome de usu rio  j
                   est  em uso. Por favor, escolha outro nome.
                   ".encode('utf-8'))
```

```
198             socket_cliente.close()
199             continue
200
201         clientes.append((socket_cliente, username,
                client_address))
202
203         # Notifica o servidor sobre o novo usu rio
204         notificar_novo_usuario(username)
205
206         # Envia a mensagem de boas-vindas para o novo
                usu rio
207         enviar_mensagem_boas_vindas(socket_cliente,
                username)
208
209         client_thread = threading.Thread(target=
                receber_mensagens, args=(socket_cliente,
                username, client_address))
210         client_thread.start()
211
212
213 def encerrar_servidor():
214     if messagebox.askyesno("Encerrar Servidor", "Deseja
            encerrar o servidor?"):
215         socket_servidor.close()
216         janela_servidor.quit()
217         janela_servidor.destroy()
218
219         # Exibir m tricas ao encerrar o servidor
220         exibir_metricas()
221
222 def exibir_metricas():
223     tempo_total = sum(tempos_classificacao)
```

```
224     tempo_medio = tempo_total / num_mensagens_classificadas
            if num_mensagens_classificadas > 0 else 0
225     metricas = (
226         f"N mero de mensagens classificadas: {
                num_mensagens_classificadas}\n"
227         f"Mensagens n o ofensivas: {num_nao_ofensivas}\n"
228         f"Mensagens ofensivas: {num_ofensivas}\n"
229         f"Tempo total de classifica  o: {tempo_total:.2f}
                segundos\n"
230         f"Tempo m dio de classifica  o: {tempo_medio:.2f
                } segundos\n"
231     )
232     print(metricas)
233     messagebox.showinfo("M tricas do Modelo", metricas)
234
235
236 janela_servidor = tk.Tk()
237 janela_servidor.title("Servidor de Chat")
238
239 ip_label = tk.Label(janela_servidor, text="IP do servidor:
       Aguardando inicializa  o...")
240 ip_label.pack()
241
242 notebook = ttk.Notebook(janela_servidor)
243 notebook.pack(expand=True, fill=tk.BOTH)
244
245 # Aba de chat
246 frame_chat = tk.Frame(notebook)
247 notebook.add(frame_chat, text="Chat")
248
249 chat_window = scrolledtext.ScrolledText(frame_chat, width
       =50, height=20)
```

```
250 chat_window.pack(padx=10, pady=10)
251 chat_window.config(state=tk.DISABLED)  # Definindo como
       somente leitura inicialmente
252
253 # Aba de banimentos
254 frame_banimentos = tk.Frame(notebook)
255 notebook.add(frame_banimentos, text="Banimentos")
256
257 ban_attempts_text = scrolledtext.ScrolledText(
       frame_banimentos, width=50, height=10)
258 ban_attempts_text.pack(pady=5)
259
260 # Bot es para banimento e desbanimento
261 botao_banir_usuario = tk.Button(frame_banimentos, text="
       Banir Usu rio", command=banir_usuario)
262 botao_banir_usuario.pack(pady=5)
263
264 botao_desbanir_usuario = tk.Button(frame_banimentos, text="
       Desbanir Usu rio", command=desbanir_usuario)
265 botao_desbanir_usuario.pack(pady=5)
266
267 # Bot o para encerrar o servidor
268 botao_encerrar = tk.Button(janela_servidor, text="Encerrar
       Servidor", command=encerrar_servidor)
269 botao_encerrar.pack(pady=10)
270
271 thread_iniciar_servidor = threading.Thread(target=
       iniciar_servidor)
272 thread_iniciar_servidor.start()
273
274 janela_servidor.mainloop()
```

Código-fonte 2 – Código em Python do Cliente de Chat

```
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext, simpledialog, messagebox

def receber_mensagens():
    while True:
        try:
            data = cliente_socket.recv(1024)
            if not data:
                raise Exception("Servidor desconectado")
            mensagem = data.decode('utf-8')
            if mensagem.startswith("###encerrar###"):
                nome_desconectado = mensagem.split("###")
                    [1]
                messagebox.showinfo("Aviso", f"{
                    nome_desconectado} saiu da conversa.")
                cliente_socket.close()
                janela_cliente.destroy()
                break
            else:
                janela_chat.insert(tk.END, f"{mensagem}\n")
        except Exception as e:
            print(f"Erro na recep  o de mensagens: {e}")
            messagebox.showinfo("Aviso", "O servidor foi
                desconectado.")
            cliente_socket.close()
            janela_cliente.destroy()
            break

def enviar_mensagem(event=None):
```

```
29      if cliente_socket:
30          mensagem = entrada_mensagem.get().strip()
31          if mensagem:
32              cliente_socket.send(mensagem.encode('utf-8'))
33              janela_chat.insert(tk.END, f"{nome_usuario}: {
                    mensagem}\n")
34              entrada_mensagem.delete(0, tk.END)
35          else:
36              messagebox.showwarning("Aviso", "Por favor,
                    insira uma mensagem antes de enviar.")
37      else:
38          messagebox.showwarning("Aviso", "Voc   est
                desconectado do servidor.")
39
40  def desconectar(nome_usuario):
41      global cliente_socket, janela_cliente
42      if cliente_socket:
43          mensagem = f"{nome_usuario} SAIU."
44          cliente_socket.send(mensagem.encode('utf-8'))
45          cliente_socket.close()
46          janela_cliente.destroy()
47
48  def obter_nome_usuario():
49      global nome_usuario, cliente_socket, janela_cliente,
            janela_chat, entrada_mensagem
50      nome_usuario = simpledialog.askstring("Nome de Usu rio
            ", "Digite seu nome de usu rio:")
51      if nome_usuario:
52          host = simpledialog.askstring("IP do Servidor", "
                Digite o IP do servidor:")
53          porta = simpledialog.askinteger("Porta", "Digite a
                porta de conex o do servidor:")
```

```
54          if host and porta:
55              cliente_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_STREAM)
56              try:
57                  cliente_socket.connect((host, porta))
58                  print("Conex o estabelecida com o servidor
                        ")
59                  cliente_socket.send(nome_usuario.encode('
                        utf-8'))
60
61                  janela_cliente = tk.Tk()
62                  janela_cliente.title("Cliente de Chat")
63
64                  janela_chat = scrolledtext.ScrolledText(
                        janela_cliente, width=50, height=20)
65                  janela_chat.pack(padx=10, pady=10)
66
67                  entrada_mensagem = tk.Entry(janela_cliente,
                         width=40)
68                  entrada_mensagem.pack(pady=10)
69                  entrada_mensagem.bind("<Return>",
                        enviar_mensagem)
70
71                  botao_enviar = tk.Button(janela_cliente,
                        text="Enviar", command=enviar_mensagem)
72                  botao_enviar.pack()
73
74                  botao_desconectar = tk.Button(
                        janela_cliente, text="Desconectar",
                        command=lambda: desconectar(nome_usuario
                        ))
75                  botao_desconectar.pack()
```

```
76                  janela_cliente.protocol("WM_DELETE_WINDOW",
                        lambda: desconectar(nome_usuario))
77
78                  receber_thread = threading.Thread(target=
                        receber_mensagens)
79                  receber_thread.start()
80
81                  janela_cliente.mainloop()
82
83              except Exception as e:
84                  print(f"Erro ao conectar ao servidor: {e}")
85                  exit()
86          else:
87              print("IP do servidor ou porta n o fornecidos.
                    ")
88              exit()
89      else:
90          print("Nome de usu rio inv lido.")
91
92  obter_nome_usuario()
```