UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS NATURAIS E DA SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO

Vanderson Faria Milagre

Análise da Defesa De Grandes Modelos De Linguagem Com Uso de Marca D'água

Alegre - ES
2025

Vanderson Faria Milagre

Análise da Defesa De Grandes Modelos De Linguagem Com Uso de Marca D'água

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo - UFES
Centro de Ciências Exatas, Naturais e da Saúde
Departamento de Computação

Orientador: Prof. Jacson Rodrigues Correia da Silva

Alegre - ES
2025

Vanderson Faria Milagre

Análise da Defesa De Grandes Modelos De Linguagem Com Uso de Marca D'água

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 21 de março de 2025.

COMISSÃO EXAMINADORA

Jacson R. C. Silva

Prof. D. Sc. Jacson Rodrigues Correia da Silva
Universidade Federal do Espírito Santo
Orientador

Pirovani

Prof. D. Sc. Juliana Pinheiro Campos Pirovani
Universidade Federal do Espírito Santo

Valéria da Silva

Prof. D. Sc. Valéria Alves da Silva
Universidade Federal do Espírito Santo

Resumo

Tarefas de processamento de linguagem natural como: geração e sumarização de texto, responder a perguntas, tradução automática, reconhecimento de entidades nomeadas e etc, são um desafio significativo para programação tradicional devido à complexidade de descrever todas as regras e exceções da língua humana de uma forma que um programa consiga compreender. Assim, modelos de inteligência artificial tornam-se alternativas eficientes para esses casos, devido à sua capacidade de condensar o conhecimento necessário ao serem treinados em dados relacionados ao problema que deseja-se resolver. Apesar deles necessitarem de muitos recursos para esse treinamento, seu desempenho demonstrou-se bom o bastante para algumas empresas escolherem desenvolvê-los. Uma das utilidades mais recorrentes no mercado para esses modelos, é ofertá-los como um serviço acessível para usuários externos. Entretanto, essa operação coloca o modelo em risco de um ataque de extração, no qual um usuário (também chamado de adversário) tenta extrair o conhecimento dele para montar uma base de dados, a qual ele pode depois utilizar para treinar um modelo clone, que apresenta acurácia similar ao original. Esses ataques de extração podem ser realizados em modelos de diversas áreas além do processamento de linguagem natural, como, por exemplo, o processamento de imagens. No entanto, há um método de identificação de clones para essa área, o qual é conhecido como marca d'água. Ele consiste em treinar o modelo para reconhecer um determinado padrão em suas entradas, isso em conjunto ao aprendizado de sua tarefa primária. Assim, durante um ataque de extração, o modelo clone também deve aprender a reconhecer esse padrão, facilitando sua identificação, a qual servirá de base para a abertura de um processo legal contra o adversário. Portanto, sabendo da eficácia da marca d'água para modelos de processamento de imagens, entre outros, este trabalho propõem uma análise da eficácia desse método na identificação de clones de modelos de processamento de linguagem natural. Neste trabalho, isso é feito por meio do treinamento de alguns modelos de linguagem BERT em diferentes tipos de marcas d'água desenvolvidos neste trabalho. Esses modelos são então extraídos, e seus clones são analisados para verificar se a marca d'água permaneceu após o ataque. A partir dos resultados desse processo, pode-se dizer que há métodos de aplicação de marca d'água que são efetivos para a proteção de modelos de processamento de linguagem natural, incluindo um que foi capaz de corretamente identifica-la em 65% das entradas para a tarefa de PTM, e 99% das entradas para a tarefa de PSS.

Palavras-chaves: Inteligência Artificial, Modelos de Processamento de Linguagem Natural, Ataque de Extração, Marcas D'Água

Abstract

Natural language processing tasks such as text generation and summarization, question answering, machine translation, named entity recognition, etc., are a significant challenge for traditional programming due to the complexity of describing all the rules and exceptions of human language in a way that a program can understand. Thus, artificial intelligence models become efficient alternatives for these cases, due to their ability to condense the necessary knowledge when trained on data related to the problem to be solved. Although they require a lot of resources for this training, their performance has proven good enough for some companies to choose to develop them. One of the most common uses in the market for these models is to offer them as an accessible service to external users. However, this operation puts the model at risk of an extraction attack, in which a user (also called an adversary) tries to extract its knowledge to build a database, which he can then use to train a clone model, which has similar accuracy to the original. These extraction attacks can be performed on models from various areas other than natural language processing, such as image processing. However, there is a clone identification method for this area, which is known as watermarking. It consists of training the model to recognize a certain pattern in its inputs, in conjunction with learning its primary task. Thus, during an extraction attack, the clone model must also learn to recognize this pattern, facilitating its identification, which will serve as the basis for opening a legal proceeding against the adversary. Therefore, knowing the effectiveness of watermarking for image processing models, among others, this work proposes an analysis of the effectiveness of this method in identifying clones of natural language processing models. In this work, this is done by training some BERT language models on different types of watermarks developed in this work. These models are then extracted, and their clones are analyzed to verify whether the watermark remained after the attack. From the results of this process, it can be said that there are watermarking methods that are effective for protecting natural language processing models, including one that was able to correctly identify it in 65% of the inputs for the PTM task, and 99% of the inputs for the PSS task.

Keywords: Artificial Intelligence, Natuaral Language Processing Models, Model Extraction, Watermarking

Lista de ilustrações

Figura 1 – Representação Gráfica de uma Camada de uma Rede Neural Recorrente. Tradução da figura de Elman (1990) 18
Figura 2 – Representação Gráfica de uma Célula LSTM. Tradução da figura de Houdt, Mosquera e Nápoles (2020) 20
Figura 3 – Representação Gráfica do Sistema de Atenção de um Transformer. Tradução da figura de Vaswani et al. (2017) 21
Figura 4 – Representação da Estrutura de um Transformer. Tradução da figura de Vaswani et al. (2017) 22
Figura 5 – Composição de um Token do BERT em relação as *Embeddings* Utilizadas. Tradução da figura de Kenton e Toutanova (2019) 25
Figura 6 – Representação de um ataque de extração em um modelo BERT. Tradução da figura de Krishna et al. (2019) 29
Figura 7 – Representação da ativação de neurônios de um modelo (quanto mais claro mais ativo) para dados legítimos e dados marcados. Tradução da figura de Jia et al. (2021) 31

Lista de tabelas

Tabela 1 – Resultados do Treinamento Inicial 38
Tabela 2 – Resultados da Extração de Controle 39
Tabela 3 – Resultados da Aplicação da Marca D'água 41
Tabela 4 – Resultados da Extração de Modelos com Marca D'água 42

Lista de abreviaturas e siglas

BERT	*Bidirectional Encoder Representations from Transformers*
GPT	*Generative Pre-Trained Transformer*
IA	Inteligência Artificial
LLM	Grande Modelo de Linguagem (*Large Language Model*)
LSTM	Memória de Curto Longo Prazo (*Long Short-Term Memory*)
PI	Propriedade Intelectual
PLN	Processamento de Linguagem Natural
PSS	Predição de Sentença Seguinte
PTM	Predição de Token Mascarado
RNN	Rede Neural Recorrente (*Recurrent Neural Network*)
TMS	Tamanho Máximo de Sequência

Sumário

1 INTRODUÇÃO . . . 9
1.1 Importância do Problema . . . 11
1.2 Trabalhos Correlatos . . . 11
1.3 Organização do Trabalho . . . 12

2 OBJETIVOS . . . 13
2.1 Objetivos Específicos . . . 13

3 REFERENCIAL TEÓRICO . . . 14
3.1 Modelo de IA para o Processamento de Linguagem . . . 14
3.1.1 Redes Neurais . . . 15
3.1.1.1 Retropropagação e Outras Ferramentas de Aprendizado . . . 15
3.1.1.2 Redes Neurais Recorrentes . . . 17
3.1.2 Grandes Modelos de Linguagem . . . 19
3.1.2.1 Memória de Curto e Longo Prazo . . . 19
3.1.2.2 O Transformer . . . 20
3.1.3 Modelo BERT . . . 24
3.2 Destilação de Modelos . . . 26
3.2.1 Destilação de Modelos . . . 26
3.2.2 Ataques de Extração de Modelos . . . 28
3.3 Marca d'Água - Watermark . . . 29

4 METODOLOGIA . . . 32
4.1 Ferramentas Utilizadas . . . 32
4.2 Treinamento Inicial . . . 32
4.3 Processo de Extração . . . 34
4.4 Aplicação e Extração da Marca D'água . . . 35
4.5 Medição da Acurácia dos Modelos . . . 36

5 RESULTADOS . . . 37
5.1 Treinamento Inicial . . . 37
5.2 Processo de Extração . . . 38
5.3 Aplicação da Marca D'água . . . 39
5.4 Extração da Marca D'água . . . 41

6 CONCLUSÃO . . . 44

REFERÊNCIAS **45**

1 Introdução

A automatização de tarefas de Processamento de Linguagem Natural (PLN) como tradução e geração de texto se trata de um campo de estudos relativamente jovem cujos primeiros trabalhos eram mais dedicados à definição de técnicas (BAR-HILLEL, 1960; YNGVE, 1957). Isso começou a mudar a partir da década de sessenta com o desenvolvimento dos primeiros programas a fazerem uso dessas técnicas para resolverem alguns problemas comuns. Dentre eles destaca-se a ELIZA, um precursor dos primeiros *chatbots*. (WEIZENBAUM, 1966) Ela fazia uso de um conjunto de regras para determinar suas respostas. Porém, em meados da década de 80, esse método de PLN começou a ser substituído por processos baseados em modelos probabilísticos como o Modelo Oculto de Markov (JELINEK, 1985).

Em conjunto com os modelos de Markov, uma outra estratégia eficiente para o PLN foi observada no campo de Inteligência Artificial (IA), as Redes Neurais Artificiais. Apesar dela já ser estudada há alguns anos (MCCULLOCH; PITTS, 1943; ROSENBLATT et al., 1962), foi o trabalho de Rumelhart, Hinton e Williams (1986) que demonstrou o grande potencial da técnica para o PLN.

Os primeiros modelos de PLN baseados em redes neurais já exibiam uma performance muito maior que seus predecessores (BENGIO; DUCHARME; VINCENT, 2000), e todos os avanços subsequentes nessa tecnologia, como redes neurais recorrentes (ELMAN, 1990) e memória de curto e longo prazo (HOCHREITER, 1997) mantiveram esse padrão de melhoria em relação aos que vieram antes deles.

O próximo grande desenvolvimento, e sistema utilizado até hoje seria o *transformer*, o qual é capaz de regular a atenção que a rede neural dá para cada palavra de sua entrada, permitindo-a aprender quais são os termos mais importantes para a compreensão do significado de uma sentença (VASWANI et al., 2017). A partir dessa arquitetura foram desenvolvidos alguns dos modelos de PLN mais reconhecidos até hoje como o *Generative Pretrained Transformer* (GPT) (RADFORD et al., 2018) e o *Bidirectional Encoder Representations from Trasnformers* (BERT) (KENTON; TOUTANOVA, 2019).

Entretanto, ao mesmo tempo que a qualidade dos novos modelos aumentava, o número de parâmetros deles também crescia. Foi até mesmo observado que os modelos de PLN mais avançados apresentavam mais parâmetros do que seria realmente necessário para apresentar uma boa acurácia em suas tarefas (ZHANG et al., 2021). Esse fenômeno das redes neurais é chamado do super parametrização, e ocorre propositalmente, pois é comprovado que um maior número de parâmetros facilita o treinamento do modelo (DU; LEE, 2018). Contudo, como esses parâmetros a mais são principalmente úteis para o treinamento, o ideal seria que eles fossem removidos após esse processo ser finalizado, algo realizado pela destilação.

A destilação se trata de um método de transferência de conhecimento de um modelo

para outro, onde o modelo receptor desse conhecimento, chamado de estudante, apresenta um tamanho menor em relação ao modelo que lhe está passando o conhecimento, chamado de professor. Logo, o resultado final da destilação é um modelo com desempenho similar ao seu professor, mas sem necessitar de tantos recursos quanto ele (HINTON; VINYALS; DEAN, 2015). Um exemplo desse processo muito reconhecido na literatura é o DistilBERT, uma versão destilada do BERT cujo tamanho do modelo é reduzido em 40%, mas ainda apresenta 97% da performance do modelo original (SANH et al., 2019). No entanto, apesar dos processos de destilação mais avançados requererem acesso direto ao modelo original para serem utilizados (SANH et al., 2019; JIAO et al., 2020; TANG et al., 2019), há um grupo específico de métodos de destilação que não precisam desse acesso, eles são categorizados como métodos de extração.

Os métodos de extração, também chamados de ataques de extração, representam um risco para modelos de IA pois se tratam de operações de destilação capazes de serem realizadas por meio da interface pública do modelo. Segundo a pesquisa de Tramèr et al. (2016), um ataque de extração pode ser caracterizado como um processo em que uma entidade adversária envia uma série de entradas distintas para um modelo alvo e utiliza as saídas dessas para treinar um novo modelo chamado de substituto ou usurpador, o qual apresenta uma performance similar ao original. Para modelos de PLN, esse risco é ainda maior porque o adversário pode realizar a extração de, pelo menos, o conhecimento linguístico do modelo utilizando entradas não relacionadas à tarefa principal dele, ou que sequer apresentem algum sentido semântico (KRISHNA et al., 2019).

Logo, surge a necessidade de uma técnica para proteger o conhecimento, ou Propriedade Intelectual (PI), de um modelo contra um ataque de extração, ou pelo menos identificar quando um clone é feito e lançado para o público. Um método que tem se demonstrado eficiente para a proteção de PI em modelos de redes neurais profundas é a marca d'água. Apresentada na pesquisa de Uchida et al. (2017), ela consiste em treinar a rede neural para reconhecer quando um padrão específico estiver presente na sua entrada, isso em conjunto com aprender a ter boa performance em seu objetivo principal. Esse duplo aprendizado é possível devido, novamente, ao grande número de parâmetros do modelo, os quais são versáteis o bastante para representar o conhecimento de ambas as tarefas ao mesmo tempo (ADI et al., 2018). Apesar da técnica ser relativamente recente, diversas implementações dela já foram desenvolvidas, tanto para redes neurais profundas (ADI et al., 2018; ZHANG et al., 2018), quanto para modelos generativos (geradores de imagens) (PENG et al., 2023a; YANG; PENG; JIA, 2024). Porém, sua aplicação em modelos de PLN ainda é muito pouco estudada (PENG et al., 2023b; HE et al., 2022).

Portanto, o propósito deste trabalho foi analisar o desempenho da marca d'água em modelos de PLN, especificamente, seu impacto na acurácia do modelo e também sua utilização para identificação em modelos de PLN clonados, ou seja, a verificação de sua resistência no modelo clonado.

1.1 Importância do Problema

Modelos de IA para o PLN requerem centenas de milhares de exemplos, dias de treinamento e são compostos por milhões de parâmetros (RADFORD et al., 2018; KENTON; TOUTANOVA, 2019), tornado-os extremamente valiosos para seus desenvolvedores como peças de sua PI. Logo, é de interesse minimizar a chance do modelo ser extraído e maximizar a velocidade de identificação de seus usurpadores, já que esses representam potenciais competidores para o modelo original.

Além disso, usurpadores apresentam a vantagem significativa de terem adquirido a mesma acurácia que o modelo original usando uma base de dados de menor custo, permitindo que seus detentores cobrem menos pelos seus serviços, ou gastem seus recursos remanescentes na continuação do treinamento do modelo usurpador, para que esse alcance uma performance ainda melhor que o modelo original. Um possível exemplo do segundo caso é o Deepseek (DEEPSEEK-AI et al., 2025), um modelo chinês publicado no início de 2025 o qual apresenta performance melhor que as versões mais avançadas do ChatGPT, e está sendo acusado de ter sido treinado em conhecimento extraído do modelo da OpenAI (OPENAI. . . , 2025).

Como as marcas d'água já são responsáveis por realizar esse papel de identificação em outros tipos de modelos de IA (ADI et al., 2018; ZHANG et al., 2018; PENG et al., 2023a), é importante identificar não só a eficácia delas na proteção de modelos de PLN, como também possíveis pontos fracos que venham a apresentar, e a metodologia de aplicação necessária para elas serem de boa qualidade.

1.2 Trabalhos Correlatos

Apesar de poucos, ainda é possível destacar dois trabalhos interessantes sobre o uso de marcas d'água em modelos de PLN. O primeiro é a pesquisa de Peng et al. (2023b), na qual ele estabelece um conjunto de palavras chave, e o modelo é treinado para alterar sua representação interna da entrada dependendo da quantidade de vezes que uma das palavras chave for identificada nela. O segundo é o artigo de He et al. (2022), o qual também seleciona um conjunto de palavras chave do vocabulário do modelo, porém a marca d'água consiste em mapear essas palavras, quando elas forem identificadas na entrada, para uma opção de um respectivo conjunto de sinônimos dela, de acordo com uma série de regras pré-estabelecida.

Então, ambos os métodos requerem a seleção de um conjunto de palavras para a aplicação correta da marca d'água. Isso representa mais uma consideração a ter durante cada etapa do treinamento de um modelo, já que seria necessária a seleção de dados de treinamento que contenham exemplos suficientes dessas palavras para garantir que o modelo codifique o conhecimento da marca d'água. Além disso, as configurações de

aplicação e treino devem ser ajustadas com cuidado para garantir que a marca d'água não interfira com o desempenho do modelo em seu objetivo principal, pois nos dois casos ele estaria procurando por palavras únicas para determinar as alterações, as quais são mais prováveis de ocorrer em relação a padrões inteiros.

1.3 Organização do Trabalho

Este trabalho foi dividido nos seguintes capítulos: o Capítulo 2, que apresenta o objetivo principal do trabalho e divide-o em um conjunto de objetivos específicos; o Capítulo 3, que descreve os conceitos básicos necessários para a compreensão dos assuntos a serem discutidos; o Capítulo 4, que apresenta a metodologia de desenvolvimento do processo de aplicação da marca d'água e as ferramentas utilizadas; o Capítulo 5, que descreve os resultados observados durante os testes desse processo; e o Capítulo 6 que apresenta a conclusão alcançada a partir desses resultados.

2 Objetivos

O objetivo deste trabalho é avaliar a utilização de marcas d'água para a defesa de modelos de PLN, por meio do desenvolvimento de um processo de aplicação de marca d'água que é resistente à ataques de extração. Para isso, foi analisada a persistência da marca d'água em um modelo usurpador extraído de um modelo BERT (KENTON; TOUTANOVA, 2019), no qual o processo de aplicação deste trabalho foi utilizado.

2.1 Objetivos Específicos

Os seguintes passos foram definidos para alcançar o objetivo geral:

1. Selecionar um dataset público para o treinamento de modelos de PLN.
2. Treinar um modelo BERT nesse dataset. Esse será o modelo alvo.
3. Criar uma cópia desse dataset e adicionar a marca d'água a algumas das entradas dele.
4. Treinar um segundo modelo BERT no dataset marcado. Esse será o modelo marcado.
5. Realizar um ataque de extração em ambos os modelos BERT.
6. Avaliar e comparar as acurácias no objetivo principal e na marca d'água, de ambos os modelos extraídos.

3 Referencial Teórico

Neste capítulo foram estabelecidas algumas definições e conceitos utilizados no restante do trabalho. Para facilitar a compreensão ele foi dividido nas seguintes seções: a seção 3.1 apresenta a ideia de modelos de IA para o processamento de linguagem natural, seção 3.2 define o conceito de marcas d'água para modelos de IA, e a seção 3.3 define o processo de destilação de modelos.

3.1 Modelo de IA para o Processamento de Linguagem

Esta seção foi organizada como um breve histórico dos sistemas e técnicas de processamento de linguagem natural desenvolvidos nos últimos anos, com um foco especial dado para o BERT na subseção 3.1.3, o principal modelo deste trabalho.

O estudo de Processamento de Linguagem Natural (PLN) iniciou-se com o desenvolvimento das primeiras técnicas de tradução automática, a mais de cinquenta anos atrás (BAR-HILLEL, 1960), as quais se tratavam de conjuntos de regras sintáticas. No entanto, um dos primeiro programas a fazer uso dessas regras não foi um tradutor, mas o sistema ELIZA, um precursor dos *chatbots* atuais. ELIZA simulava uma conversa com o usuário por meio de uma interface simples, pela qual ela recebia mensagens e apresentava respostas com base no que foi enviado (WEIZENBAUM, 1966).

Após esses desenvolvimentos, durante as décadas de 1980 e 1990, os sistemas mais avançados de PLN não eram mais baseados em conjuntos de regras, eles usavam modelos probabilísticos, especificamente o Modelo Oculto de Markov (*Hidden Markov Model*) (JELINEK, 1985). A ideia por trás desse modelos consiste em possuir dois processos com seus próprios grupos de estados: o processo X, observável, e o processo Y, oculto. Sabendo que o estado atual de X depende do processamento de Y, o objetivo do modelo é conseguir compreender o estado de Y a partir de X. Porém, uma limitação lógica desses modelos é o fato de que eles somente podem ser aplicados em problemas que apresentam a propriedade de Markov, a qual dita que o estado atual de um processo deve ser independente de seus estados anteriores (BAUM; PETRIE, 1966). Apesar dessa limitação, o modelo demonstrou ser eficiente em diversas tarefas de PLN como marcação de partes de linguagem (BAHL et al., 1989), e reconhecimento de fala (RABINER, 1989).

Apesar dos avanços técnicos dos Modelos Ocultos de Markov, foi a área de aprendizado de máquina que apresentou as melhorias mais rápidas e significativas em sua performance.

3.1.1 Redes Neurais

O campo de aprendizado de máquina consiste no estudo e desenvolvimento de programas capazes de serem treinados para cumprir alguma tarefa. Isso é feito por meio da extração do conhecimento presente em uma base de dados que eles recebem, ao invés de uma lógica de programação rígida (SAMUEL, 1959). Os estudos pioneiros dessa área se tratam dos trabalhos de McCulloch e Pitts (1943), e Rosenblatt et al. (1962) que descreveram a modelagem computacional e matemática de um neurônio artificial, o qual se tornou elemento mais básico de todas as redes neurais.

O trabalho de McCulloch e Pitts (1943) demonstrou que um neurônio poderia ser representado por um portão lógico, e que, por extensão, uma série desses portões formaria uma rede capaz de realizar funções lógicas e aritméticas. Ele também proporcionou um modelo matemático de como essa rede seria estruturada, o qual se tornou a base para as primeiras redes neurais.

Por sua vez, Rosenblatt et al. (1962) usou essas representações de neurônios para definir o perceptron, a primeira representação computacional de um neurônio. Seu funcionamento consistia em receber um conjunto de entradas $X = (x_1, x_2, ..., x_n)$, e realizar a soma ponderada dessas entradas utilizando um conjunto de pesos $W = (w_1, w_2, ..., w_n)$, resultando em um valor Y. Ele ainda apresentou um algoritmo que ajustaria os pesos dos perceptrons conforme eles eram expostos a mais dados de treinamento, aprimorando sua capacidade de identificar os padrões em seus dados de entrada.

Um outro trabalho dessa época muito significativo para o campo de aprendizado de máquina, apesar de não se focar nessa área de estudo, foi o livro de Aho e Ullman (1972), *The Theory of Parsing Translation and Compiling*. O livro é focado no design de compiladores, mas ele também apresenta uma definição formal para o token, um conceito muito importante para PLN. Sabendo que um modelo de PLN geralmente recebe uma sentença como entrada, o primeiro passo de processamento dela é sua divisão em partes menores, as quais podem ser analisadas individualmente. Essa divisão de sentença é chamada de tokenização, e as partes geradas são chamadas de tokens. Na maioria dos casos, esses tokens são as palavras e pontuações que compõem a sentença, mas há tokenizadores, as ferramentas responsáveis pela tokenização, que aplicam uma divisão ainda mais granular (KENTON; TOUTANOVA, 2019). Para facilitar a compreensão deste trabalho, considere que os termos “token” e “palavra”, no contexto de entradas para um modelo de PLN, são sinônimos.

3.1.1.1 Retropropagação e Outras Ferramentas de Aprendizado

Entretanto, apesar desses avanços, redes neurais ainda não eram significativamente exploradas como ferramentas para resolver problemas de PLN. Isso somente começou a mudar com o desenvolvimento de modelos com múltiplas camadas de neurônios. Nesses

modelos, cada neurônio de uma camada está conectado a, e envia suas saídas para, todos os outros neurônios da camada imediatamente superior à dele. Assim, as entradas dos neurônios das camadas superiores são as somas ponderadas das saídas de suas camadas imediatamente inferiores, e o treinamento consiste no ajuste dos pesos de suas somas (IVAKHNENKO; LAPA; SERVICE, 1965).

Nisso, surge a necessidade de um algoritmo que consiga corretamente ajustar esses pesos através das múltiplas camada de uma rede neural, algo que foi realizado pelo algoritmo de retropropagação (*backpropagation*). Descrito no trabalho de Rumelhart, Hinton e Williams (1986), o algoritmo calcula a diferença entre a saída final da rede e o valor que era realmente esperado, diferença a qual representa o erro da rede. Com isso, o algoritmo determina qual é o gradiente do erro em relação a cada peso da rede, e ajusta-os na direção que diminuiria esse gradiente e, assim, o erro. Esse ajuste é realizado de trás para frente, ou seja, os gradientes de erro dos neurônios da última camada da rede são calculados e eles são ajustados primeiro, depois o processo se repete para os neurônios de cada uma das camadas intermediárias, mais conhecidas como camadas ocultas, até se chegar na camada inicial da rede.

Nesse mesmo trabalho, também foi observado que o processo de treinamento fica mais eficiente quando os pesos da rede não são ajustados a cada entrada que ela recebe, mas a cada conjunto de X entradas. Nesse caso, cada uma das X entradas é processada pelo modelo e seu respectivo erro é calculado. O modelo só ajusta seus pesos depois do grupo inteiro ter passado, usando uma combinação dos erros das entradas. Esse conjunto de entradas é chamado de lote (*batch*), e seu tamanho corresponde à quantidade de entradas que ele possui. Além disso, o processo de um lote passar pelo modelo e esse ajustar seus pesos de acordo com o erro do lote, é dito como uma "iteração" ou "passo" do treinamento (RUMELHART; HINTON; WILLIAMS, 1986).

Vale também notar, que a pesquisa de Rumelhart, Hinton e Williams (1986) descreve que o aprendizado é melhorado ao introduzir um parâmetro chamado de taxa de aprendizagem. Ele controla o tamanho do ajuste que algoritmo faz a cada iteração, isso permite que o desenvolvedor evite problemas como uma correção excessiva dos pesos levando-os a terem um maior erro do que antes de serem ajustados. Atualmente há outras ferramentas e técnicas relacionadas à taxa de aprendizagem, os quais permitem que ela seja adaptada durante o treinamento. Alguns exemplos são: agendadores de aprendizagem, que reduzem a taxa e então a intensidade das alterações nos pesos; e passos de aquecimento, nos quais a taxa não é ajustada (DARKEN; MOODY, 1991).

Além da retropropagação, uma outra ferramenta para o treinamento de redes neurais que surgiu nessa época foi o processo de decaimento de pesos, apresentado na pesquisa de Krogh e Hertz (1991). O propósito do decaimento de pesos é melhorar a generalização do modelo ao impedir que os pesos de alguns neurônios atinjam valores muitos altos, porque isso levaria eles, e por consequência a informação que codificam, a ter um impacto

desproporcional no resultado do modelo. Isso é feito ao ajustar o cálculo de erro dos neurônios específicos para que ele aumente o resultado final de acordo com a magnitude do peso, ou seja, neurônios com pesos maiores vão apresentar erros maiores, levando a ajustes mais intensos.

Um dos primeiros artigos a aplicar essas ferramentas para resolução de problemas de PLN veio de Bengio, Ducharme e Vincent (2000), o qual fez uso de uma rede neural de múltiplas camadas numa tarefa de predição de próxima palavra a partir de um contexto conhecido. Para isso, ele representa cada palavra de seu vocabulário como um vetor de n parâmetros, o qual é chamado de *embedding*. Nela, cada parâmetro se trata de uma representação numérica de uma das características da palavra. Isso permite que o modelo tenha uma compreensão mais profunda do significado de cada palavra e generalize mais facilmente. Durante o treinamento, o modelo aprende duas informações: a representação ideal de cada palavra do vocabulário, e como usar a combinação dessas representações para determinar a próxima palavra da sentença. A rede resultante demonstrou uma performance significativamente melhor que os métodos tradicionais da época.

3.1.1.2 Redes Neurais Recorrentes

Essas melhorias em desempenho continuaram a ser observadas em cada novo desenvolvimento técnico das redes neurais, começando pelas Redes Neurais Recorrentes (RNNs). Apresentadas pela primeira vez no artigo de Elman (1990), elas tentavam solucionar o problema da representação temporal que Elman tinha identificado nos modelos existentes até então. Segundo ele, a temporização de cada palavra em uma frase, ou seja, a ordem em que essas palavras seriam descritas, é necessária para compreender o significado dela. Assim, qualquer modelo que tente representar linguagem deveria representar essa temporização. Além disso, ele afirma que representações espaciais desse tempo dificultariam a aprendizagem do modelo, já que essas seriam incluídas como mais um elemento no vetor de representação da entrada, e o modelo teria de tentar distinguir qual parte do vetor representa aquilo que estava descrito na entrada, da parte que representa quando essa entrada foi descrita.

A solução apresentada pelo artigo foram as Redes Neurais Recorrentes (RNNs). A estrutura está representada na Figura 1, e como pode se ver, para cada camada oculta de uma dessas redes, ela também possuiria uma camada chamada de contexto, a qual guardaria o valor da camada oculta na iteração de treinamento anterior à atual. Especificamente, em uma iteração de treinamento qualquer t, a entrada de cada camada oculta da rede seria uma combinação da saída da camada anterior a ela, indicada na imagem como "camada de entrada", com os valores registrados em sua respectiva camada de contexto, os quais se tratam de seus próprios valores na iteração *t-1*. Após a camada oculta passar seu resultado para a camada seguinte, ela ativaria sua camada de contexto e registraria seu valor atual nela (ELMAN, 1990).

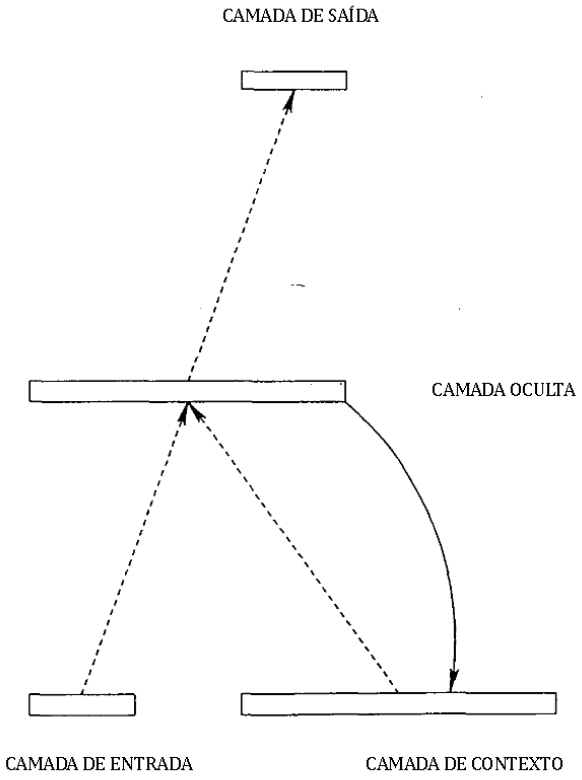

Figura 1 – Representação Gráfica de uma Camada de uma Rede Neural Recorrente. Tradução da figura de Elman (1990)

O propósito dessa estrutura é atuar como uma memória de curto prazo para o modelo, permitindo que seu processamento atual seja influenciado por aquilo que veio logo antes dele, e proporciona uma representação da temporização de cada palavra em uma frase. Por meio desse novo processo, a rede neural recorrente conseguiu corretamente aprender conceitos gramaticais como concordância entre sujeito e verbo e a hierarquia das palavras de uma frase (ELMAN, 1990).

Entretanto, um ponto fraco dessa estrutura são entradas compostas por sentenças longas. Nesses casos, durante o processo de retropropagação, conforme o erro da rede é ajustado para corrigir os pesos de cada um dos neurônios, ele sofre uma de duas alterações: ou ele continuamente aumenta (explode), ou ele continuamente diminui (desaparece) (GRAVES; MOHAMED; HINTON, 2013). Ambas geram problemas para o treinamento do modelo. No primeiro caso, um valor de erro muito alto impede o modelo de fazer pequenos ajustes em seus pesos levando-o a constantemente ter de se corrigir. Já no segundo caso, o tempo de treinamento aumenta significativamente, pois o modelo irá demorar cada vez mais a levar seus pesos aos valores corretos deles. Em certos casos, a segunda alteração pode até paralisar completamente o processo de aprendizagem, pois o erro se torna tão pequeno que o modelo não é mais capaz de ajustar seus pesos. A causa dessas alterações

constantes seria o próprio processo de memória de curto prazo, já que como a camada aplica todos os mesmos valores que ela tinha em sua iteração anterior em sua entrada, ela realizaria repetidamente certas modificações em seu gradiente, independentemente do efeito que essas teriam no erro total da rede (SCHMIDHUBER, 1992).

3.1.2 Grandes Modelos de Linguagem

Para resolver esse problema foi necessário o desenvolvimento de técnicas mais avançadas, arquiteturas mais complexas, e redes com ainda mais parâmetros. Essa última característica se tornou a principal identificação dos Grandes Modelos de Linguagem (LLM), os quais se tratam de modelos de PLN cujo número de parâmetros está na escala de centenas de milhões. Usando essa maior quantidade de parâmetros, os novos modelos se tornaram capazes de atingir um melhor desempenho, especialmente em entradas mais longas, nas quais o modelo deve analisar um contexto maior para cada token da sentença. Duas novas estruturas foram desenvolvidas para tirar o proveito máximo desses novos parâmetros: a Memória de Curto e Longo Prazo (LSTM) (HOCHREITER, 1997) e o transformer (VASWANI et al., 2017).

3.1.2.1 Memória de Curto e Longo Prazo

O primeiro grupo dos LLMs foram os modelos de LSTM. Apresentados no artigo de Hochreiter (1997), a característica marcante desses modelos é a presença da célula de memória de curto e longo prazo, ou célula LSTM. Ela se trata de uma estrutura composta por um estado interno e três portões, o portão de entrada, o portão de saída e o portão de esquecimento, e pode ser observada na Figura 2. Em cada iteração de execução, a célula recebe seu estado oculto da iteração anterior e a entrada atual, como em uma RNN usual. A diferença está em como o estado interno da célula afeta essas entradas e como ele é alterado por elas, algo que é controlado pelos portões da célula. Especificamente em cada iteração é realizado o seguinte processamento dentro da célula: o portão de esquecimento determina quanta informação do estado interno será mantida com base nas suas entradas, depois o portão de entrada controla o quanto das novas entradas será salvo dentro do estado interno, por fim o portão de saída controla o quanto das entradas e do estado interno será passado como o estado oculto da próxima iteração desta célula, e como a entrada da próxima célula LSTM da rede.

Por meio dessa estrutura o modelo seria capaz de simular uma memória de longo prazo, e controlar quando e como essa memória influenciaria a entrada da camada. Isso garante que ele não realize alterações desnecessárias em seu gradiente de erro, levando-o a explodir ou desaparecer (HOCHREITER, 1997). Assim, os modelos de LSTM se tornaram o ápice da eficiência em tarefas de PLN, especialmente aquelas que envolviam a

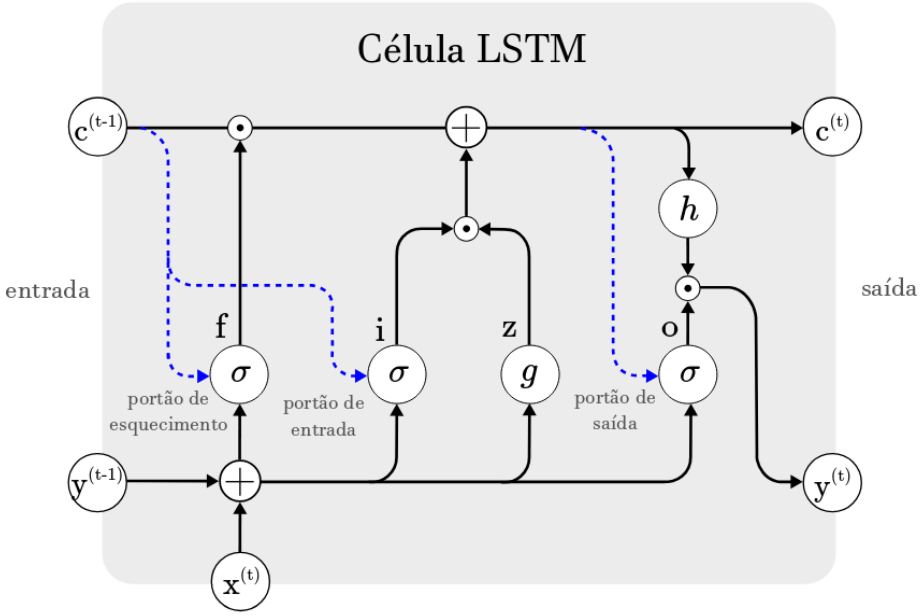

Figura 2 – Representação Gráfica de uma Célula LSTM. Tradução da figura de Houdt, Mosquera e Nápoles (2020)

análise de múltiplas sentenças longas, como tarefas de reconhecimento de fala (GRAVES; MOHAMED; HINTON, 2013).

No entanto, a característica principal das RNNs, o seu processamento linear das palavras para codificar a sua ordem na sentença, eventualmente se tornou um gargalo de performance, pois ele diretamente limitava o potencial de paralelização dos modelos, impedindo-os de extrair o máximo de benefício do hardware moderno. A solução para esse problema viria na pesquisa de Vaswani et al. (2017), a qual apresentou o mecanismo do transformer.

3.1.2.2 O Transformer

O transformer é a peça chave do segundo e mais diversificado grupo de LLMs. Seu funcionamento pode separado em duas partes principais: os mecânismos de atenção, e as pilhas de encoder e decoder.

Os sistemas de atenção servem para permitir que o próprio modelo determine quais palavras e posições da entrada apresentavam uma significância maior para a palavra que está sendo processada. No caso do transformer, é utilizado um sistema próprio de atenção chamado de Atenção Escalada do Produto Escalar (*Scaled Dot-Product Attention*). Esse sistema está representado no lado esquerdo da Figura 3, e consiste no seguinte: durante o processando da atenção do *embedding* de uma palavra, o modelo estabelece três dados a partir da *embedding*: a chave (K), a query (Q) e o valor (V). O cálculo da atenção da palavra é feito a partir do produto escalar da query da palavra atual com as chaves das palavras restantes da entrada. As matrizes resultantes são divididas pela raiz quadrada

de sua dimensão e passam por uma função softmax. Os resultados desse processo são usados como os pesos para a soma ponderada dos valores da palavra sendo processada (VASWANI et al., 2017).

Entretanto, para garantir que o modelo tenha uma compreensão maior de sua entrada, o cálculo de atenção é realizado múltiplas vezes em paralelo, num processo chamado de Atenção de Múlti-Cabeças (Multi-Head Attention) e representado no lado direito da Figura 3. Para que isso ocorra, o valor, query e chave da entrada são replicados, essas réplicas passam por camadas lineares independentes e depois são enviadas para suas respectivas camadas de atenção, as quais existem uma ao lado da outra. Essas camadas recebem o nome de cabeças de atenção. Após elas terminarem seu processamento, seus resultados são concatenados e o valor resultante passa por uma última camada linear antes de seguir para a próxima etapa da rede. A ideia seria que cada uma das cabeças faria uma representação distinta da entrada, permitindo que o modelo codifique diferentes significados ao mesmo tempo (VASWANI et al., 2017).

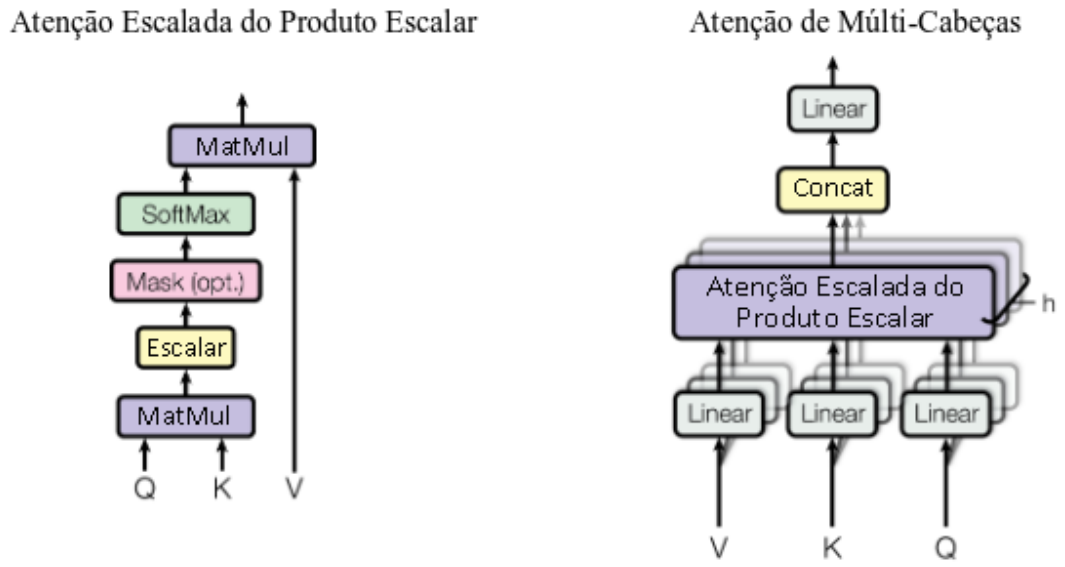

Figura 3 – Representação Gráfica do Sistema de Atenção de um Transformer. Tradução da figura de Vaswani et al. (2017)

Já as pilhas de encoder e decoder, são duas estruturas de organização para os blocos de atenção (os conjuntos de camadas de atenção). Uma representação dessa estrutura está na Figura 4. Um encoder é dividido em duas partes, primeiro, um bloco de atenção, e depois, uma camada completamente conectada de neurônios. Ambas as partes são seguidas de uma camada de normalização e o conjunto é repetido N vezes. O decoder tem uma estrutura similar, mas entre a primeiro bloco de atenção e a camada completamente conectada, há um segundo bloco de atenção, cuja entrada são as saídas do respectivo encoder. Assim,

a entrada para a camada de normalização após o segundo bloco de atenção se trata da saída de ambos os blocos do decoder.

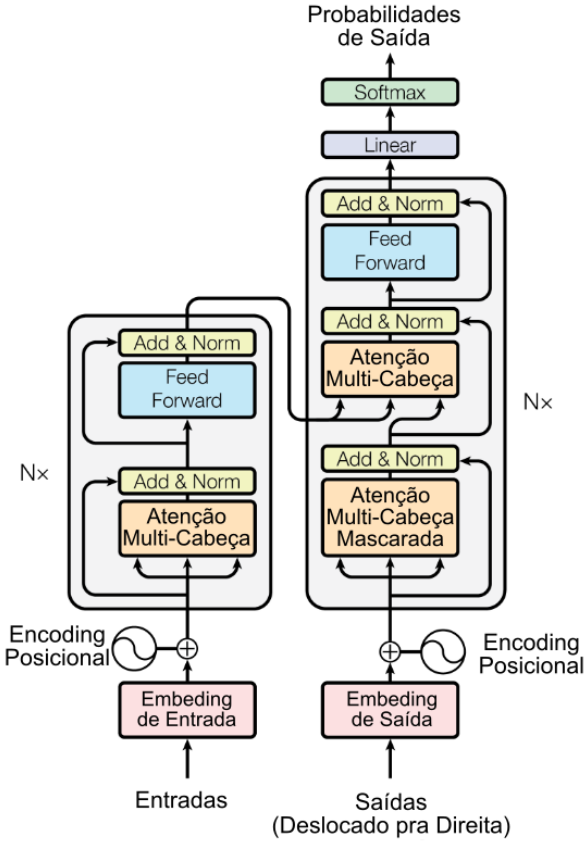

Figura 4 – Representação da Estrutura de um Transformer. Tradução da figura de Vaswani et al. (2017)

A maneira como cada pilha processa a entrada também difere. O encoder processa a sentença de entrada inteira, enquanto o decoder processa cada token um por um. A ideia por trás desse funcionamento é que o encoder agiria como uma memória da entrada inteira, a qual o decoder pode analisar enquanto processa cada um dos tokens. Em adição a isso, o decoder é auto-regressivo, ou seja, a entrada para um token também considera os tokens que foram gerados para a entrada anterior. Devido a isso, o decoder somente pode acessar as informações dos tokens anteriores ao qual ele está processando, já que somente esses já teriam sido processados (VASWANI et al., 2017).

Além da atenção e da estrutura em duas pilhas, o transformer ainda apresenta mais um sistema importante, as *embeddings* posicionais. Elas são adicionadas às *embeddings* das palavras antes de sua atenção ser calculada, e representariam o posicionamento da palavra processada em sua sentença de entrada, já que o transformer não usaria mais o processamento sequencial para fazer essa representação, como as redes recorrentes fazem (VASWANI et al., 2017).

Esses sistemas permitiram que o transformer atingisse uma performance similar aos modelos recorrentes mais avançados, porém com o diferencial de ser completamente

paralelizável. Assim, ele pode fazer uso total dos componentes de hardware modernos para alcançar esses resultados num tempo significativamente menor que esses modelos (VASWANI et al., 2017).

Um dos modelos que melhor exemplifica o potencial dos transformers foi o GPT – *Generative Pre-Trained Transformer*. Apresentado no artigo de Radford et al. (2018), o GPT se trata de um modelo transformer de decoder puro, ou seja, ele não possui uma pilha de encoder. Como o foco principal dele são tarefas de geração de texto, nas quais os tokens são gerados um por um, essa arquitetura provou-se mais eficiente.

Ele também foi desenvolvido com o intuito de apresentar um solução para um problema significativo em PLN, a falta de exemplos de texto categorizados, os quais são imprescindíveis durante o treinamento de um modelo para uma tarefa específica. Essa solução se trata de dividir o processo de treinamento em duas partes chamadas de pré-treinamento e aprimoramento (RADFORD et al., 2018).

Durante o pré-treinamento o modelo receberia uma vasta quantidade de entradas de texto não categorizadas, as quais são mais fáceis de adquirir, e seria treinado numa tarefa de predição de próxima palavra. Essa tarefa consiste no modelo aprender a prever a próxima palavra de uma sentença que ele recebeu. Para isso a saída do modelo se trata de valores chamados de *logits*, os quais podem ser interpretados como pontuações que ele dá para cada uma das palavras do vocabulário dele, e que são transformadas em probabilidades para indicar a confiança do modelo em qual seria a próxima palavra da sentença (RADFORD et al., 2018).

Após isso, o modelo poderia ser aprimorado para qualquer tarefa de PLN por meio da adição de um conjunto de transformações específicas à tarefa que ele está sendo aprimorado, e de um treinamento mais curto em cima de um conjunto menor de entradas categorizadas. A teoria por trás dessa divisão é que o modelo aprenderia conceitos básicos de linguagem, como gramática, semântica, estruturação de sentenças e lógica básica, durante seu pré-treinamento, e então ele poderia se concentrar em atingir uma boa performance em sua tarefa específica durante o aprimoramento (RADFORD et al., 2018).

A arquitetura do transformer foi utilizada devido a sua capacidade de modelar relações entre partes distantes do texto, algo especialmente útil em tarefas como tradução automática, geração de documentos e análise sintática. No caso da tarefa de predição de próxima palavra do pré-treinamento do GPT, o transformer determina seu resultado usando somente as palavras que viriam antes dela na sentença. Isso garante que o modelo gere suas predições usando somente informações contextualmente apropriadas para tarefas de geração de texto, algo necessário para que o modelo seja o mais adaptável possível a diferentes tarefas de PLN (RADFORD et al., 2018).

Por meio desses avanços o GPT conseguiu uma melhora de até 8,9% em 9 diferentes tarefas de PLN. E provou a qualidade da separação do processo de treinamento em duas etapas, pois os modelos de estado da arte nessas tarefas eram todos treinados

especificamente nelas desde seu estado inicial (RADFORD et al., 2018).

Outro modelo além do GPT que implementou o transformer para o PLN foi o BERT, porém ele apresenta algumas características diferentes em sua arquitetura e treinamento que permitem-no atingir uma boa performance, especialmente em tarefas de nível de sentença.

3.1.3 Modelo BERT

O foco principal deste trabalho, o modelo BERT – *Bidirectional Encoder Representations from Transformers* – foi apresentado na pesquisa de Kenton e Toutanova (2019) e se trata de outro modelo pré-treinado para o PLN baseado em transformers, similar ao GPT, porém com algumas distinções significativas.

Primeiramente, a arquitetura dos transformers dos modelos. GPT é um modelo de decoder puro tornando eficiente em tarefas de geração de texto (RADFORD et al., 2018). Entretanto, o BERT é um modelo de encoder puro (KENTON; TOUTANOVA, 2019). Como o BERT tenta apresentar um desempenho melhor especialmente em tarefas de nível de sentença, é necessária uma arquitetura que permita-lhe considerar a sentença inteira ao processar um token, para garantir que um significado mais completo para esse seja alcançado, e o decoder limita a análise a somente os tokens anteriores ao que está sendo processado.

Assim, o BERT realiza todas as suas análises considerando todos os tokens da sentença. Porém, para garantir que o modelo ainda seja capaz de apresentar boa acurácia em tarefas de geração, e não dependa da sentença inteira estar disponível em qualquer situação, o processo de pré-treinamento foi modificado. Especificamente, durante a execução dele alguns tokens seriam mascarados, e o objetivo da rede seria determinar para quais tokens eles eram originalmente mapeados. Essa se trata da primeira tarefa de treinamento do BERT, chamada de Predição de Token Mascarado (PTM) (KENTON; TOUTANOVA, 2019).

Em mais detalhes, o processo de mascaração consiste em selecionar, aleatoriamente, 15% dos tokens da sentença de entrada para serem mascarados de alguma forma. As possibilidades de mascaração seriam as seguintes: em 80% dos casos, o token é substituído por um token especial chamado de [MASK]; em 10% dos casos, o token é convertido em um token aleatório do vocabulário; e nos outros 10%, o token não sofre qualquer alteração. O propósito desse processo é garantir que o modelo sempre tente realizar a predição com qualquer token da sentença, pois, se somente o token especial [MASK] fosse utilizado, o modelo aprenderia a somente prever quando o encontrasse, algo que seria inútil pois a mascaração ocorre somente durante o treinamento e não na execução usual. Além disso, esse processo evita que a interpretação de um token fique completamente dependente de outro token, já que qualquer um deles pode ter sido mascarado e, portanto, divergir completamente do significado real da sentença (KENTON; TOUTANOVA, 2019).

A segunda distinção do BERT em relação do GPT, é o método de processamento de sentenças do modelo. Para que o modelo tenha uma melhor compreensão dos relacionamento entre sentenças de um texto, o BERT foi projetado de tal forma que ele possa receber tanto uma quanto duas sentenças como sua entrada. Isso é feito por dois métodos: a inserção de um token especial [SEP] entre a representação dos termos de cada sentença, e um *embedding* especial adicionado aos tokens, que representaria a sentença à qual pertence esse termo. Assim, o modelo representa cada termo ou palavra da entrada da rede como uma combinação de três *embeddings*, cada qual codifica uma informação diferente: a *embedding* de token, que representa a palavra ou termo, a *embedding* posicional, que representa a posição da palavra dentro da entrada e a *embedding* de sentença, que representa a qual sentença pertence a palavra. Uma representação gráfica dessa combinação pode ser observada na Figura 5.

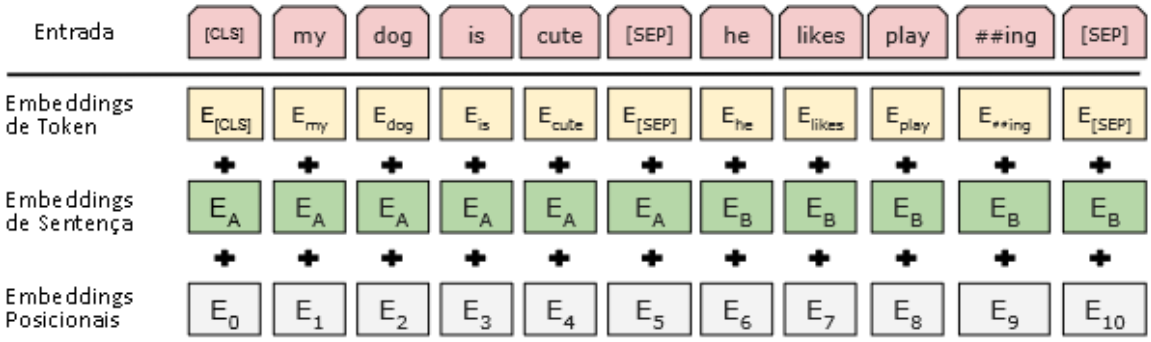

Figura 5 – Composição de um Token do BERT em relação as *Embeddings* Utilizadas. Tradução da figura de Kenton e Toutanova (2019)

Essa combinação de sentenças na entrada do modelo também se trata da segunda tarefa de treinamento do BERT, chamada de Predição de Sentença Seguinte (PSS). Para essa tarefa o modelo deve determinar se a segunda sentença de uma entrada é realmente sucessora da primeira. Nesse processo, 50% dos pares de sentença enviados para o BERT são realmente sentenças seguidas em seu texto original, e nos outros 50%, são um par de sentenças aleatória do corpus de treinamento.

Uma outra observação importante sobre o treinamento do BERT tem a ver com seu Tamanho Máximo de Sequência (TMS). Esse se trata do número máximo de tokens que uma entrada pode possuir, e ainda ser processável pelo modelo. No caso do BERT esse número é de 512 tokens, mas durante 90% do treinamento ele somente recebe entradas com 256 tokens ou menos. Essa redução é feita para acelerar o processo de treinamento, mas ela é desfeita nos últimos 10% dos tokens de treinamento para evitar que prejudique o conhecimento do modelo, especialmente em relação às suas *embeddings* posicionais (KENTON; TOUTANOVA, 2019).

Quanto à sua estrutura, o BERT segue o padrão estabelecido por Vaswani et al. (2017), em que as camadas do modelo se tratam de blocos de transformer, também chamados de blocos de atenção. Esses blocos são conectados uns aos outros por uma mini rede neural clássica, com apenas uma camada oculta. No caso do BERT a sua pesquisa apresenta duas arquiteturas principais: BERT Base e BERT Grande, as quais diferem na quantidade de blocos de atenção, no número de cabeças em cada bloco e no tamanho da camada oculta da rede neural usada para conectar os blocos de atenção. Especificamente, o BERT Base apresenta 12 blocos de atenção, com 12 cabeças por bloco e 768 neurônios na camada oculta; enquanto o BERT Grande possui 24 blocos, 16 cabeças por bloco e 1024 neurônios na camada oculta.

Esse números são uma demonstração do tamanho que os modelos de PLN mais avançados podem alcançar, tamanho o qual também traz uma necessidade de hardwares mais potentes para executar e treinar o modelo, mais energia para alimentar as máquinas que o mantém e um maior tempo de treinamento. Assim, costuma ser inviável executar os modelos de PLN modernos em máquinas acessíveis a consumidores comuns ou dispositivos móveis. Porém, há processos que permitem a criação de novas versões desses modelos, as quais são executáveis em dispositivos com um hardware mais fraco e/ou consomem menos recursos que suas versões originais. Esses são os processos de destilação.

3.2 Destilação de Modelos

3.2.1 Destilação de Modelos

O conceito de destilação de conhecimento, ou destilação de modelos, foi introduzido no estudo de Hinton, Vinyals e Dean (2015) e consiste em treinar um modelo de arquitetura grande, chamado de professor, em um determinado objetivo, e depois usar as distribuições de probabilidade das predições dele como objetivos de treinamento de novo modelo com arquitetura menor, chamado de estudante. Especificamente, o modelo estudante é treinado para replicar essas distribuições de probabilidade, porém o estudo também demonstrou que o treinamento é facilitado quando os objetivos se tratam de uma média ponderada entre as distribuições e as classificações corretas utilizadas no treinamento do professor. Dessa forma, o estudante tem acesso a muito mais informação durante o seu treino do que o professor teve durante o dele. Isso permite-o utilizar menos exemplos de treinamento para alcançar uma acurácia comparável, além de chegar mais rápido nesse objetivo. Por fim, o estudo demonstrou que os modelos destilados apresentavam um redução de 1% a 2% na sua acurácia em relação ao modelo original, um impacto insignificante em relação à sua facilidade de treinamento e menor custo de operação, devido ao menor número de parâmetros.

A explicação para essa capacidade das redes neurais de manter sua performance apesar

da redução no tamanho, é descrita no trabalho de Zhang et al. (2021), o qual evidencia que a grande maioria dos modelos de rede neural modernos são super-parametrizados, ou seja, eles apresentam um número de parâmetros consideravelmente maior do que aquilo que seria realmente necessário para representar a função de generalização em seu objetivo principal. As redes possuem esse número excessivo de parâmetros porque, apesar de apenas uma fração deles ser necessária para representar a função objetivo, é comprovado que um modelo com um número maior deles é capaz de encontrar essa função mais rapidamente e com maior facilidade (HESTNESS et al., 2017).

A partir da pesquisa de Hinton, Vinyals e Dean (2015), diversos métodos de destilação foram desenvolvidos para os modelos de PLN, especialmente para o BERT. As tentativas de destilação do conhecimento dele se deram ínicio em duas pesquisas: a de Tang et al. (2019), que destilava o conhecimento do BERT para uma rede neural de memória de curto e longo prazo com uma arquitetura mais simples que a do modelo original; e a pesquisa de Sun et al. (2019), a qual realiza a destilação do BERT utilizando as saídas das camadas ocultas do modelo, ao invés da distribuição de probabilidades final utilizada pelos métodos de destilação usuais. Porém, os dois métodos anteriores precisam ser aplicados em um modelo BERT que já foi aprimorado para alguma tarefa, ou seja, o modelo resultante da destilação somente se preocupa com o conhecimento do BERT na tarefa a qual ele foi aprimorado, o que pode levar à perda de uma parte do conhecimento geral do modelo. Portanto, um método melhor seria aquele que consegue garantir a destilação de tanto o conhecimento geral, quanto do conhecimento específico, e um processo que consegue fazer isso é aquele apresentado na pesquisa do TinyBERT.

Segundo o artigo de Jiao et al. (2020), o modelo TinyBERT reduz a quantidade de blocos de atenção do BERT base para quatro ou seis blocos, dependendo das necessidades do usuário, e também reduz o tamanho da camada oculta da rede que conecta os blocos para 312 neurônios. Quanto ao treinamento do TinyBERT, ele é dividido em duas partes: uma para destilar o conhecimento geral do modelo BERT, o qual só foi pré-treinado, e outra para destilar o conhecimento do modelo BERT que já foi aprimorado para a tarefa objetivo. Isso significa que a destilação para o TinyBERT precisa ocorrer logo após as etapas do treinamento do BERT. Outra característica notável sobre esse processo de destilação é a quantidade de informações que são destiladas do modelo original. Durante o processo, o TinyBERT recebe conhecimento: das matrízes de atenção dos transformers, das camadas ocultas que conectam os blocos, dos logits da última camada do modelo e das embedings das entradas.

Entretanto, o TinyBERT ainda apresenta o problema de ter de ser destilado imediatamente para uma tarefa específica, perdendo a versatilidade encontrada em seu modelo professor. Teoricamente é possível parar o processo logo após a destilação do conhecimento geral, sem realizar o aprimoramento, mas a própria pesquisa já informa que o modelo resultante tem uma performance significativamente pior que o BERT original, caso ele

adquira seu conhecimento específico por algum outro método que não seja a destilação descrita no próprio trabalho (JIAO et al., 2020). Contudo, há um outro método destilação capaz de gerar um modelo que possa ser aprimorado conforme o usuário deseja. Ele é chamado de DistilBERT e foi apresentado na pesquisa de Sanh et al. (2019).

A estrutura do DistilBERT é relativamente simples, ela começa igual à do BERT Base com duas alterações para a redução do tamanho do modelo: exclusão de metade das camadas de transformer e remoção das *embeddings* de segmento. Além disso, antes do treinamento de destilação, as camadas de transformer do DistilBERT são inicializadas com os pesos de uma a cada duas camadas do BERT original, enquanto os pesos das outras partes também são os mesmos do modelo original.

O treinamento do DistilBERT é realizado com uma combinação de três perdas diferentes: a perda em relação ao objetivo principal da rede e a perda em relação às distribuições de probabilidade do modelo original, seguindo o método descrito por Hinton, Vinyals e Dean (2015), e um terceiro cálculo chamado de perda dos cossenos dos *embeddings*. Esse se trata do cálculo de perda de cossenos, o qual é um método para determinar a diferença entre os ângulos de dois vetores (BARZ; DENZLER, 2020), que é aplicado aos vetores de da camada oculta da rede.

Observa-se que os processos de destilação mencionados até o momento requerem que o aplicador deles tenha acesso a informações privadas do modelo original como: matrizes de atenção, valores das camadas ocultas, logits e etc. No entanto, há outros métodos de destilação de conhecimento que podem ser realizados sem necessidade de acesso a informações privadas do modelo original. Esse métodos são chamados de ataques de extração e representam um risco para modelos de IA.

3.2.2 Ataques de Extração de Modelos

Segundo a pesquisa de Tramèr et al. (2016), um ataque extração de um modelo pode ser definido da seguinte forma: considere um modelo de IA que foi treinado em uma tarefa específica e, então, compreendeu a função $f(x)$ capaz de corretamente resolver essa tarefa, um ataque de extração se trata de um processo realizado por um usuário desse modelo, chamado de adversário, em que ele tenta copiar $f(x)$ por meio de suas próprias interações com o modelo. Especificamente, o usuário envia uma entrada para o modelo e registra a saída desse. Esse processo é realizado diversas vezes, ao final das quais o adversário tem um conjunto de entradas relacionadas às suas saídas esperadas. Ele pode então usar essas informações para treinar seu próprio modelo na mesma tarefa, o qual é geralmente referido na literatura como modelo usurpador. Esse processo é apresentado na Figura 6. A pesquisa também mostra que esse processo é facilitado caso a rede neural apresente sua distribuição de confiança como saída, a qual descreve todas as possíveis respostas da rede para a tarefa em questão, relacionadas com suas respectivas confianças de serem a resposta correta.

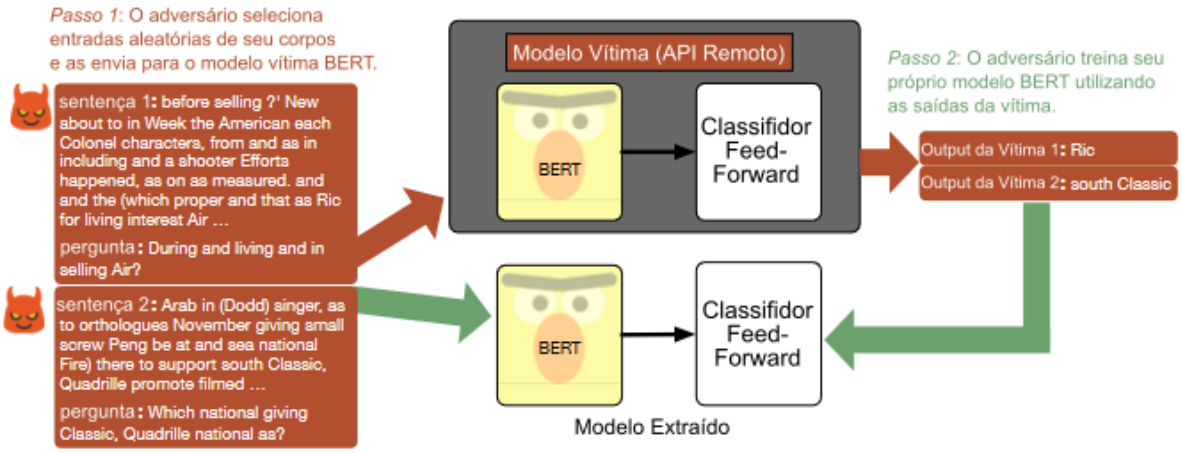

Figura 6 – Representação de um ataque de extração em um modelo BERT. Tradução da figura de Krishna et al. (2019)

Devido ao custo monetário, quantidade de informações e tempo necessários para treinar um modelo que apresenta boa performance em seu objetivo, o ataque de extração representa uma ameaça considerável à PI do detentor do modelo original, pois ele pode ser realizado por uma fração desse custo de treinamento. Além disso, caso o adversário tenha os meios para isso, ele pode seguir com o treinamento de seu usurpador após a extração para que ele atinja uma performance ainda melhor que o modelo original (OPENAI. . . , 2025).

O risco desse ataque tornou-se ainda maior após os resultados da pesquisa de Krishna et al. (2019), a qual demonstrou que, para modelos PLN, é possível extrair seu conhecimento usando dados de fora do conjunto de treinamento, ou até mesmo sequências aleatórias de palavras, desde que essas remetam de alguma forma ao objetivo principal da rede, apesar que esse processo reduz a qualidade do modelo extraído. Além disso, a pesquisa também mostrou que não é necessário para o modelo usurpador ter a mesma arquitetura que o original, podendo ele ser significativamente menor, apesar de ainda apresentar uma acurácia próxima à desse.

Portanto, modelos usurpadores representam uma competição ilegítima a modelos PLN honestos, além de uma violação da PI deles. Logo, é necessário algum método de identificação de usurpadores, e um dos mais reconhecidos atualmente é a marca d'água.

3.3 Marca d'Água - Watermark

A marca d'água foi descrita na pesquisa de Uchida et al. (2017) como uma ferramenta para combater ataques de extração em modelos de IA. Ela funciona como uma identificação única de um modelo, a qual também é copiada caso esse seja extraído por um adversário. Então, ela pode ser utilizada para determinar se um outro modelo suspeito é realmente um usurpador ou não. Especificamente, a marca d'água se refere a um padrão aprendido

pelo modelo de IA durante seu treinamento, em conjunto ao aprendizado de seu objetivo principal. Quando o padrão é identificado pelo modelo, sua saída deve ser alterada de uma forma já pré-determinado pelo aplicador da marca d'água, independentemente do restante da entrada. Porém, para que uma marca d'água seja realmente efetiva, o processo de aplicação dela não pode interferir com a acurácia do modelo final em seu objetivo principal.

No trabalho Uchida et al. (2017), a marca d'água é aplicada em uma rede neural convolucional profunda, e o processo de aplicação dela consiste em uma função regularizadora utilizada em uma das camadas do modelo, a qual altera os neurônios dessa durante o treinamento para reconhecerem a marca d'água. Assim, caso a identifiquem, eles devem alterar suas entradas para garantir que a saída da rede neural seja equivalente ao valor pré-determinado por seu aplicador. Isso deve ser feito com uma acurácia similar àquela que a rede apresenta para seu objetivo principal.

A partir dessa pesquisa outros métodos de aplicação de marca d'água foram desenvolvidos e eles diferem uns dos outro em dois aspectos principais: o processo de criação do padrão da marca d'água, e o meio pelo qual ele é aplicado ao modelo. Um método que exemplifica isso ao adotar uma metodologia mais simples é apresentado no trabalho de Adi et al. (2018). A pesquisa descreve como é possível aplicar a marca d'água na rede neural, durante o treinamento dessa, por meio da intercalação de exemplos que apresentam o padrão desejado com exemplos "legítimos", ou seja sem o padrão da marca d'água. Nesse caso, a rede é treinada para retornar as saídas pré-determinadas quando ela recebe entradas com o padrão. Assim, os neurônios responsáveis pelo reconhecimento da marca d'água estariam dispersos pela rede, ao invés de estarem localizados em uma única camada.

Esse aprendizado simultâneo de dois objetivos é crucial para a aplicação de marcas d'água, e ele é possível devido à super-parametrização dos modelos modernos. Isso porque esse grande número de parâmetros permite que a rede aprenda a identificar padrões simples nas suas entradas, em conjunto com seu objetivo principal, sem apresentar perda de acurácia significativa em nenhum dos dois (HESTNESS et al., 2017; ADI et al., 2018).

Entretanto, mesmo que o modelo aprenda corretamente ambos os objetivos, isso não garante que a marca d'água será copiada pelo usurpador durante um ataque de extração. Essa possibilidade foi observada na pesquisa de Jia et al. (2021), na qual ele teoriza que modelos treinados em múltiplas tarefas dividem seus neurônios em grupos, e esses codificam o conhecimento de objetivos diferentes. Isso é um problema considerável para a aplicação de marca d'água. Considere a Figura 7, a qual representa os neurônios de um modelo treinado numa tarefa qualquer em conjunto à tarefa de identificação do padrão de uma marca d'água, uma parte desses tem a serventia única de se ativar quando o padrão for reconhecido na entrada. Portanto, os neurônios somente demonstrariam uma ativação significativa quando recebessem uma entrada com a marca d'água. Como o adversário não teria conhecimento da marca d'água, ele dificilmente enviaria uma entrada que apresentasse

o padrão dela. Logo, como os neurônios de reconhecimento nunca seriam ativados, seu conhecimento seria perdido durante a extração, e o modelo usurpador resultante desse ataque não seria identificável pelo detentor original.

Figura 7 – Representação da ativação de neurônios de um modelo (quanto mais claro mais ativo) para dados legítimos e dados marcados. Tradução da figura de Jia et al. (2021)

A solução para esse problema consiste em, de alguma forma, fazer com que o modelo "entrelace" o conhecimento do objetivo principal com o conhecimento da marca d'água, ou seja, garantir que a codificação de ambos os objetivos seja dispersa por todos os neurônios do modelo, ao invés de ser concentrada em grupos específicos (JIA et al., 2021).

No caso da pesquisa de Jia et al. (2021), foi utilizado um cálculo adicional de erro para as entradas de marca d'água. Nelas, além do cálculo de erro que a rede aplica a qualquer entrada, também é realizado outro cálculo utilizando a função de *Soft Nearest Neighbor Loss* (SNNL), a qual calcula a diferença média entre duas entradas de diferentes grupos. Assim, ao minimizar essa perda, o modelo estaria entrelaçando esses dois grupos. Neste caso um dos grupos seria composto pelas entradas legitimas e o outro pelas entradas de marca d'água, e então a perda final seria uma combinação dessas duas perdas descritas. O objetivo desse processo é que, além do modelo aprender a identificar corretamente quaisquer entradas, legitimas e marca d'água, ele mantivesse suas representações internas dessas similares, pois assim elas seriam tratadas de maneira similar enquanto são processadas pela rede.

4 Metodologia

A análise da persistência de uma marca d'água realizada neste trabalho consistiu em três etapas: O treinamento inicial do modelo, seção 4.2, que corresponde ao treinamento de um modelo local para servir como o exemplo de controle da pesquisa; o processo de extração, seção 4.3, o qual se trata de uma descrição de como a extração do modelo foi realizada; e a aplicação da marca d'água, seção 4.4, que explica quais foram as configurações e processos utilizados para garantir que o modelo consiga aprender a identificar a marca d'água. Além disso, a seção 4.1, apresenta as especificações da máquina utilizada e uma visão geral das ferramentas usadas neste trabalho. Por fim, a seção 4.5 descreve como a medição da acurácia dos modelos foi realizada.

4.1 Ferramentas Utilizadas

Este trabalho foi desenvolvido usando Pytorch, uma framework de aprendizado de máquina escrita em Python. Ela apresenta diversas aplicações e ferramentas úteis para o desenvolvimento de modelos de IA para tarefas de PLN. Além disso, ela também é a framework primária da biblioteca HuggingFace (HUGGING. . . , 2024c), outra parte importante desta pesquisa. Em mais detalhes, a biblioteca HugginFace se trata de um repositório de diversas ferramentas para o desenvolvimento e treinamento de sistemas de IA, como: datasets, tokenizadores, implementações de modelos e etc. As ferramentas específicas que foram utilizadas neste trabalho são discutidas nas seções seguintes.

Todos os testes foram realizados em uma máquina com as seguintes especificações: processador Intel i7-13700F, memória RAM de 32GB, placa de vídeo NVIDIA GeForce RTX 4070 com 12GB de RAM, e armazenamento NVME.

4.2 Treinamento Inicial

O treinamento inicial do modelo BERT foi feito usando uma combinação de dois datasets baixados do repositório da HuggingFace (HUGGING. . . , 2024c), o dataset da *Wikipedia* (HUGGING. . . , 2024e), composto por artigos em inglês completos da enciclopédia digital, e o dataset do *Bookcorpus* (HUGGING. . . , 2024b), o qual se trata de passagens de diversas obras literárias, também escrito em inglês. Esses datasets foram selecionados por serem versões atualizadas dos mesmos datasets utilizados no treinamento original do BERT (KENTON; TOUTANOVA, 2019), logo já há um histórico do modelo conseguir adquirir bom conhecimento linguístico a partir deles. Além do histórico pré-existente de boa qualidade, este trabalho foi realizado utilizando dados em inglês porque ele necessitava

de bases de dados de textos longos publicamente disponíveis, as quais são mais fáceis de encontrar para a língua inglesa.

Um pré-processamento foi realizado no dataset da *Wikipedia* para dividir os artigos que compõem suas entradas em uma série de parágrafos, já que o tamanho de sequência máximo para o modelo, ou seja o número de tokens que ele consegue analisar ao mesmo tempo, não é grande o bastante para receber um artigo inteiro. Além disso, foram removidos os cabeçalhos, as listas e as tabelas dos artigos, de acordo com o que está descrito em Kenton e Toutanova (2019). Essas informações são removidas porque o objetivo desta etapa é que o modelo aprenda as relações entre sentenças de um texto, e esses tipos de dados acabariam interferindo nesse aprendizado.

Após esse pré-processamento, os datasets foram tokenizados. Esse processo ocorreu separadamente do treinamento por causa de ser muito demorado, o que resultaria em um tempo de treino consideravelmente maior. Como este trabalho é focado no treinamento de modelos, a tokenização dos datasets foi realizada usando o tokenizador pré-treinado da HugginFace para o BERT (HUGGING..., 2024a), o qual já é preparado para tratar de entradas em inglês.

Para carregar as entradas dos datasets de uma forma que o modelo possa receber, foi desenvolvida uma classe chamada de *TokenizedBERTDataset*. Ela é inicializada com os datasets informados anteriormente, mas realiza alguns processamentos antes de retornar as entradas que lhe são requisitadas. Especificamente, antes de uma entrada ser retornada ela é ajustada para as duas tarefas de treinamento do modelo BERT, a análise de relacionamento de sentenças, e a predição de token mascarado. Para que a primeira tarefa seja feita corretamente, o *TokenizedBERTDataset* realiza o processo de combinação de sentenças, que foi descrito no artigo original do BERT (KENTON; TOUTANOVA, 2019), ou seja a sentença da entrada selecionada é combinada a uma outra sentença dos Datasets, a qual 50% das vezes é a sentença logo após ela, e 50% das vezes é uma outra sentença qualquer. Também são adicionadas as embeddings de segmento para que o modelo possa distinguir os tokens de uma sentenças dos tokens da outra. Já para a execução da segunda tarefa, é realizada a mascaração dos tokens dessa combinação de sentenças, seguindo os parâmetros estabelecidos no artigo original do BERT. Especificamente, 15% dos tokens são mascarados e desses 80% são convertidos no token [MASK], 10% são substituidos por um token aleatório do vocabulário e os últimos 10% são mantidos os mesmos.

Vale notar que o treinamento original do BERT for realizado com um lote de 256 entradas por 1.000.000 passos (KENTON; TOUTANOVA, 2019). Os modelos treinados nesta pesquisa utilizaram tamanhos significativamente menores, e bem menos iterações de treinamento. Isso ocorreu devido a limitações de tempo e hardware disponíveis. A máquina utilizada para os treinamentos não possuía a memória necessária para comportar um lote com 256 entradas, fazendo com que um treinamento usando essa configuração fosse impossível. E, apesar da máquina ser tecnicamente capaz de treinar um modelo

por 1.000.000 de iterações, ele levava um tempo considerável para fazer isso. Como este trabalho necessitava que múltiplos modelos fossem treinados com configurações similares, era necessária uma alternativa mais eficiente. As exatas configurações de treinamento utilizadas podem ser observadas na seção 5.1.

As únicas configurações que permaneceram inalteradas foram: a mudança do TMS que é realizada durante o treinamento, a taxa de aprendizado inicial e agendador de aprendizagem (KENTON; TOUTANOVA, 2019). Além disso, também foi adicionado um processo de decaimento de pesos ao treinamento, o qual não fazia parte do processo original, mas ajuda na generalização.

4.3 Processo de Extração

Após o modelo ser treinado por completo, é realizado o processo de extração de seu conhecimento, o qual pode ser dividido em duas etapas: a geração do dataset de extração, e o treinamento do modelo usurpador nesse dataset.

Como já foi explicado no capítulo seção 3.2, esse dataset é criado ao enviar uma série de entradas para o modelo e relacioná-las com suas respectivas saídas. Especificamente para este projeto, as entradas vieram de um outro dataset baixado do repositório da HuggingFace, o *OpenWebText* (HUGGING..., 2024d), o qual é composto por diversos artigos disponíveis online. Esse dataset foi selecionado devido à estruturação de suas entradas, as quais se tratam de textos longos em que há relacionamento entre as sentenças. Isso permite que o dataset extraia o conhecimento de predição de próxima sentença do modelo.

O mesmo pré-processamento aplicado no dataset da *Wikipedia* foi repetido para este dataset, ou seja, suas entradas, as quais se tratavam de artigos inteiros, foram dividias em parágrafos para que as informações do dataset sejam processadas corretamente pelo modelo. Além disso, da mesma forma que os datasets anteriores, as entradas do *OpenWebText* dataset também foram tokenizadas antes de serem processadas.

Após o pré-processamento e a tokenização, as entradas do *OpenWebText* dataset são alimentadas ao modelo uma por uma, e são registradas em um novo dataset, em conjunto com suas respectivas saídas. Esse novo dataset é chamado de dataset de extração. Entretanto, o dataset de extração gerado é consideravelmente menor que o *OpenWebText* (o qual possui cerca de 160 milhões de entradas após o pré-processamento), já que foi possível atingir uma boa acurácia no modelo extraído utilizando bem menos elementos que o tamanho total dele. Os modelos extraídos relacionados ao número de iterações de treinamento que tiveram no dataset de extração, e suas respectivas acurácias finais podem ser observados na Tabela 2 da seção 5.2.

Por fim, com o dataset de extração em mãos, é possível treinar o modelo usurpador. Como um adversário não teria conhecimento exato das configurações de treinamento

do modelo original, elas foram simplificadas para melhor representar esse cenário. As únicas mudanças realizadas para atender a essa condição foram: o TMS não ser mais alterado durante o treinamento, então o modelo utiliza seu tamanho máximo durante todo o processo; e não haverem passos de aquecimento para a taxa de aprendizagem.

4.4 Aplicação e Extração da Marca D'água

Para a aplicação da marca d'água foi desenvolvida uma nova classe, *WatermarkedTokenizedBERTDataset*. Ela se trata de uma versão modificada da classe *TokenizedBERTDataset*, realizando as mesmas tarefas que ela, e uma tarefa adicional, a adição dinâmica da marca d'água nas entradas do dataset, a qual altera o resultado das duas tarefas de treinamento do BERT. Isso para garantir que a presença da marca d'água seja dispersa por toda a estrutura do modelo, e então aumentar a chance do conhecimento de identificação dela ser extraído por um adversário durante um ataque.

Em mais detalhes, durante o treinamento do modelo, conforme as entradas do dataset são carregadas, a classe *WatermarkedTokenizedBERTDataset* insere o padrão da marca d'água aleatoriamente em uma determinada porcentagem dessas entradas. Esse padrão afeta a tarefa de PTM ao mapear aqueles que o compõem para um dos tokens do vocabulário do BERT que não é utilizado no processamento de texto usual, o qual pode ser referido neste trabalho como [MARK]. E ele afeta a tarefa de PSS ao garantir que qualquer entrada marcada indique que suas sentenças são seguintes, independentemente desse ser realmente o caso ou não. Esse processo é apelidado dentro deste trabalho como "aplicação simples".

Entretanto, após testes com a aplicação simples, foi observado que o modelo extraído apresentava resultados ruins quanto à identificação marca d'água na tarefa PSS e péssimos na tarefa de PTM. Especificamente, o modelo nunca mapeava os tokens do padrão para o token [MARK], ou seja, o conhecimento associado a eles era completamente perdido durante o processo de extração do modelo marcado. A possível explicação para essa perda total foi uma falha no entrelaçamento do conhecimento do objetivo principal do modelo, e o conhecimento de identificação da marca d'água na tarefa de PTM. Devido a essa divisão, o ataque de extração do adversário somente extraiu o conhecimento do objetivo do modelo, já que esse era o único que ele estava tentando adquirir (JIA et al., 2021). Uma explicação mais extensa dessa possibilidade está na seção 3.3.

A solução foi garantir o entrelaçamento dos conhecimentos do modelo por meio da adição de uma etapa adicional no processo de aplicação da marca d'água. Após a inserção do padrão em uma entrada, os outros tokens da entrada até uma distância X desse padrão são mapeados para o token [MARK]. Essa distância era controlada por um novo parâmetro de configuração da rede, chamado de "raio de influência". Assim, durante o treinamento, quando o modelo analisasse um token, mesmo que esse não pertencesse ao padrão da marca d'água, ele poderia ter seu significado alterado pelo padrão. Além disso, foi adicionado

mais um parâmetro de configuração que controlava a quantidade de vezes que o padrão da marca d'água deveria ser inserido em uma entrada marcada, o qual foi chamado de "marcas por entrada". Isso foi feito para aumentar o número de tokens marcados aos quais o modelo seria exposto, sem aumentar o número de entradas marcadas, o que poderia prejudicar o aprendizado de seu objetivo principal, especialmente na tarfa de PSS. Essas novas configurações permitiram um maior entrelaçamento entre os conhecimentos do modelo para seu objetivo principal e para sua marca d'água. Esse novo processo é nomeado dentro deste trabalho de "aplicação entrelaçada".

4.5 Medição da Acurácia dos Modelos

Para calcular a acurácia dos modelos treinados durante esta pesquisa, foi utilizada um métrica da biblioteca Evaluate do Python (EVALUATE..., 2025). As métricas se tratam de ferramentas para medir a performance de um modelo em uma determinada tarefa. No caso dos modelos deste trabalho, somente foi utilizada a métrica de acurácia (HUGGING..., 2025), a qual simplesmente mede a porcentagem de equivalência entre um grupo de predições e um grupo de resultados esperados, também chamados de referências.

Em mais detalhes, no caso de um modelo não marcado, na tarefa de PTM, a métrica de acurácia calculou a equivalência da predição do modelo para os tokens em relação ao mapeamento real deles. Em outras palavras, se um token não foi mascarado, o valor final da métrica será melhor se a predição do modelo para ele indicou o próprio token. E no caso de um token mascarado, independentemente de qual tipo de mascaração foi utilizado, a métrica será melhor se a predição indicar o mapeamento real do token, ao invés do valor que ele aparenta. Quanto à tarefa de PSS, a mesma lógica é aplicada, ou seja, o valor melhora quando a predição do modelo equivale ao relacionamento real das sentenças.

No caso dos modelos mascarados, o cálculo é idêntico, levando em conta as alterações que as marcas fazem nas referências das entradas. No entanto, vale notar que três métricas de acurácia são utilizadas para esses modelos na tarefa de PTM, cada qual medindo um grupo diferente de tokens. Especificamente, uma métrica mede apenas tokens legítimos, outra mede apenas tokens marcados e a última mede todos os tokens das entradas.

5 Resultados

Este capítulo apresenta as configurações utilizadas e resultados encontrados durante a experimentação da pesquisa [1], para avaliar as acurácias dos modelos extraídos em relação ao seu objetivo principal e à marca d'água. Essa experimentação foi dividida nas seguintes etapas do projeto: o treinamento inicial do modelo controle, seção 5.1, a realização do ataque de extração ao modelo, seção 5.2, a aplicação da marca d'água, seção 5.3, e a realização do ataque de extração a um modelo que apresenta a marca d'água, seção 5.4.

O seguinte processo foi usado para calcular cada um dos resultados apresentados nas tabelas desta seção: a acurácia da tarefa especificada foi calculada para 100 entradas no modelo descrito, usando a métrica descrita na seção 4.5; esse foi repetido 10 vezes, sempre com um conjunto de 100 entradas diferentes; o resultado final se trata da média aritmética dos resultados desses 10 cálculos.

Vale notar, nos resultados da seção 5.3 e da seção 5.4, as acurácias das duas tarefas foram subdivididas em três grupos, de acordo com o tipo de token que foi analisado. Esses grupos são: tokens normais: que considera só os tokens que não foram influenciados pela marca d'água; tokens marcados: que considera os tokens da marca d'água, ou que sofreram influência dela; e completo: que considera todos os tokens da entrada.

Por fim, nas métricas da tarefa de PSS, uma acurácia próxima a 50%, indica que o modelo não apresenta qualquer conhecimento necessário para a resolução dela. Quando um modelo não possui conhecimento necessário para atingir um objetivo, ele simplesmente apresenta um resultado aleatório. Como a tarefa de PSS, possui somente dois possíveis resultados, se as sentenças são ou não seguintes, mesmo um modelo sem qualquer treinamento seria capaz de atingir uma acurácia de 50%.

5.1 Treinamento Inicial

Como já foi mencionado na seção 4.2, todos os modelos treinados neste trabalho apresentam configurações inferiores às do modelo BERT original devido a limitações de hardware e tempo. Porém, isso não deve prejudicar os testes de forma significativa já que todos modelos desta seção, os quais serviram de base para o restante dos modelos do projeto, ainda apresentaram acurácia acima de 79% em ambas as tarefas.

Em mais detalhes, foram testados três modelos para identificar qual número de iterações de treinamento resultaria em uma rede que apresentasse boa acurácia nos objetivos principais, mas não necessitasse de um tempo excessivo de treinamento, pois essa quantidade de iterações seria a base para os processos de aplicação de marca d'água e extração. Além

[1] O código disponível para a replicação do experimento pode ser encontrado em: <https://github.com/VFMiracle/TCC-Marca-Dagua-Modelos-PLN>

disso, o restante das configurações de treinamento dos modelos permaneceram as mesmas, em que o tamanho de lote foi de 16 entradas, a quantidade de iterações de aquecimento foi 10% do total de iterações, e o TMS inicial de treinamento foi de 256 tokens, mas é alterado após 90% dos passos para o tamanho de 512 tokens.

As quantidades de iterações testadas foram as seguintes:

- Modelo Controle 1 (M1): 25.000 iterações de treinamento, ou seja, o aquecimento ocorre nas primeiras 250 iterações e o TMS só é alterado após 22.500 iterações.
- Modelo Controle 2 (M2): 100.000 iterações de treinamento, ou seja, o aquecimento ocorre nas primeiras 1.000 iterações e o TMS só é alterado após 90.000 iterações.
- Modelo Controle 3 (M3): 1.000.000 iterações de treinamento, ou seja, o aquecimento ocorre nas primeiras 10.000 iterações e o TMS só é alterado após 900.000 iterações.

Tabela 1 – Resultados do Treinamento Inicial

	Acurácia PTM	Acurácia PSS	Tempo de Treinamento
M1	80,02%	79,63%	Aprox. 4h
M2	86,20%	86,22%	Aprox. 16h
M3	90,21%	89,90%	Aprox. 80h

Os resultados dos treinamentos podem ser observados na Tabela 1. A partir deles, foi selecionado o modelo M2 para ser a base dos próximos testes. Apesar do modelo M3 ter apresentados resultados melhores que ele em ambas as tarefas, ele também necessitou de consideravelmente mais tempo de treinamento. Como os modelos subsequentes usaram configurações similares à base que foi estabelecida nesta etapa, usar o modelo M3 para esse papel consequentemente aumentaria, de modo significativo, o tempo de realização de qualquer experimento.

5.2 Processo de Extração

Foi realizado um ataque de extração no modelo M2, o qual gerou um dataset com o conhecimento do modelo. Esse dataset foi utilizado para treinar um novo conjunto de modelos usurpadores por diferentes números de iterações, seguindo a metodologia descrita na seção 4.3. Especificamente, os números de iterações utilizados para cada modelo foram:

- Usurpador 1 (U1): 7.500 iterações de treinamento.
- Usurpador 2 (U2): 15.000 iterações de treinamento.
- Usurpador 3 (U3): 30.000 iterações de treinamento.

- Usurpador 4 (U4): 75.000 iterações de treinamento.
- Usurpador 5 (U5): 150.000 iterações de treinamento.
- Usurpador 6 (U6): 300.000 iterações de treinamento.
- Usurpador 7 (U7): 360.000 iterações de treinamento.

Tabela 2 – Resultados da Extração de Controle

	Acurácia PTM	Acurácia PSS	Tempo de Treinamento
U1	4,35%	50,34%	Aprox. 30min
U2	4,37%	49,62%	Aprox. 1h
U3	4,38%	49,51%	Aprox. 2h
U4	83,97%	81,39%	Aprox. 6h
U5	85.54%	81.36%	Aprox. 12h
U6	86.32%	82.10%	Aprox. 24h
U7	86.31%	82.40%	Aprox. 30h

Os resultados dos treinamentos podem ser observados na tabela Tabela 2. Como pode se ver na linha do modelo U7, apesar do aumento do número de iterações realizadas em relação ao modelo U6, a acurácia do U7 não apresentou nenhuma melhora significativa. Isso indica que o modelo já aprendeu todo o conhecimento que foi extraído para o Dataset de extração, então aumentar o número de iterações de treinamento nesse Dataset não trouxe mais nenhuma melhora. Na verdade, um aumento do número de iterações correria o risco de piorar a acurácia do modelo nos testes, já que ele poderia aprender a se adaptar somente aos dados de treinamento, perdendo sua capacidade de generalizar para novas informações.

Assim sendo, o modelo U6, treinado em 300.000 iterações, foi definido como a base para os próximos processos de extração desta pesquisa. Porque, ele apresentou os melhores resultados entre os modelos testados, foi extraído em um tempo aceitável e não deu sinais de estar sendo excessivamente treinado, pois seus resultados demonstram uma melhora notável em relação ao modelo U5.

5.3 Aplicação da Marca D'água

Quatro modelos foram treinados durante esta etapa de experimentação, dois usando a aplicação simples, e ou outros dois usando a aplicação entrelaçada. As configurações de treinamento e marca d'água de cada modelo estão listadas a seguir, sabendo que em todos os casos quando o TMS do modelo é alterado, ele sobe para 512 tokens:

- Modelo Com Marca D'água Simples 1 (WS1):

  - **Treinamento:** 100.000 iterações de treinamento, em que o aquecimento ocorre nas primeiras 1.000 iterações e o TMS é alterado após 90.000 iterações.
  - **Marca D'água:** a marca d'água foi aplicada em 10% das entradas, com o padrão: "### This is the Watermark ###".

- Modelo Com Marca D'água Simples 2 (WS2):
  - **Treinamento:** 500.000 iterações de treinamento, em que o aquecimento ocorre nas primeiras 5.000 iterações e o TMS é alterado após 450.000 iterações.
  - **Marca D'água:** a marca d'água foi aplicada em 20% das entradas, com o padrão: "### This is the Watermark ###".

- Modelo Com Marca D'água Entrelaçada 1 (WE1):
  - **Treinamento:** 150.000 iterações de treinamento, em que o aquecimento ocorre nas primeiras 1.500 iterações e o TMS é alterado após 13.500 iterações.
  - **Marca D'água:** a marca d'água foi aplicada em 15% das entradas, onde cada entrada recebeu 3 padrões, e cada padrão teve um raio de influência de 2 tokens. Foi utilizado o padrão: "### This is the Watermark ###".

- Modelo Com Marca D'água Entrelaçada 2 (WE2):
  - **Treinamento:** 150.000 iterações de treinamento, em que o aquecimento ocorre nas primeiras 1.500 iterações e o TMS é alterado após 13.500 iterações.
  - **Marca D'água:** a marca d'água foi aplicada em 15% das entradas, onde cada entrada recebeu 3 padrões, e cada padrão teve um raio de influência de 2 tokens. Foi utilizado o padrão: "###".

O padrão "### This is the Watermark ###" foi selecionado durante o treinamento dos modelos de aplicação simples porque era esperado que um padrão mais longo levasse a uma melhor persistência da marca d'água durante uma extração, especialmente com relação à tarefa de PTM, já que desse jeito o modelo seria exposto a um maior número de tokens marcados durante o treinamento. A combinação de tokens de texto comuns com carácteres incomuns (#), foi uma tentativa de garantir a identificabilidade da marca d'água, para que o modelo aprendesse ela corretamente, enquanto entrelaçava o conhecimento dela com o conhecimento do objetivo principal do modelo, porque o modelo ainda tentaria identificar os tokens de texto comuns.

Como a aplicação entrelaçada já tenta realizar esse entrelaçamento dentro de sua execução normal, independentemente do padrão que for utilizado, o teste do modelo WE2 foi realizado com um padrão composto apenas pelo carácter (#), para tornar a marca d'água mais identificável. Além disso, o tamanho do padrão foi diminuído, porque a aplicação entrelaçada insere-o múltiplas vezes nas entradas selecionadas, logo um padrão

menor aproxima mais o número de tokens marcados em cada um dos modelos testados. O padrão "### This is the Watermark ###" foi mantido para o teste do modelo WE1 como uma forma de comparação dos dois métodos de aplicação de marca d'água deste trabalho.

Tabela 3 – Resultados da Aplicação da Marca D'água

	Acurácia PTM			Acurácia PSS			
	Tokens Normais	Tokens Marcados	Completo	Tokens Normais	Tokens Marcados	Completo	Tempo de Treinamento
WS1	86,11%	100%	86,58%	86,92%	100%	91,81%	Aprox. 16h
WS2	86,31%	100%	86,72%	86,37%	100%	91,80%	Aprox. 40h
WE1	86,70%	100%	87,48%	87,34%	100%	89,21%	Aprox. 18h
WE2	87,08%	100%	87,44%	86,92%	100%	88,87%	Aprox. 19h

Os resultados podem ser observados na Tabela 3. Há dois pontos notáveis vindos desses resultados. Primeiro, todos os modelos apresentaram acurácia perfeita quanto à identificação da marca d'água em ambas as tarefas, isso demonstra que o modelo teve facilidade de reconhecer o padrão utilizado em meio ao restante dos dados de entrada, provavelmente devido ao uso do carácter "#", o qual é incomum no corpus de treino. Segundo, apesar do aumento considerável do número de iterações de treinamento do modelo WS2 em relação aos outros modelos, não houve qualquer reflexo disso em sua acurácia. Uma explicação provável para esse caso é que o modelo adquiriu todo conhecimento disponível nos dados apresentados, assim como o modelo U7 da seção 5.2.

Uma outra observação a se fazer sobre esses resultados é que, apesar de métodos de aplicação diferentes terem sido utilizados, todos os modelos apresentaram pouquíssimas variações entre as acurácias de suas tarefas (cerca de 1%).

5.4 Extração da Marca D'água

Foi realizado um ataque de extração em cada um dos modelos da seção 5.3. Todos os ataques foram realizados utilizando as configurações descritas na seção 5.2, pelo mesmo número de iterações que o modelo U6, ou seja, 300.000. Os seguintes modelos foram adquiridos e os resultados dos testes de acurácia deles estão exibidos na Tabela 4.

A coluna de tempo de extração foi omitida porque todos os modelos levaram aproximadamente o mesmo tempo de cerca de 24h para serem extraídos.

- Usurpador Com Marca D'água Simples 1 (UWS1): Extraído do modelo WS1
- Usurpador Com Marca D'água Simples 2 (UWS2): Extraído do modelo WS2
- Usurpador Com Marca D'água Entrelaçada 1 (UWE1): Extraído do modelo WE1

- Usurpador Com Marca D'água Entrelaçada 2 (UWE1): Extraído do modelo WE2

Tabela 4 – Resultados da Extração de Modelos com Marca D'água

	Acurácia PTM			Acurácia PSS		
	Tokens Normais	Tokens Marcados	Completo	Tokens Normais	Tokens Marcados	Completo
UWS1	86,10%	0%	83,14%	83,20%	62,17%	74,74%
UWS2	86,39%	0%	83,67%	81,80%	64,03%	71,40%
UWE1	86,34%	18,77%	82,36%	83,42%	43,51%	77,34%
UWE2	86,42%	67,55%	85,90%	83,16%	99,27%	85,61%

Como pode ser observado na Tabela 4, dos quatro modelos usurpadores, somente aqueles que passaram pela aplicação entrelaçada extraíram algum conhecimento da marca d'água para a tarefa de PTM. Isso demonstra que a aplicação simples fracassou no entrelaçamento do conhecimento da marca d'água para essa tarefa, com o restante do conhecimento do modelo.

Entretanto há uma diferença notável entre os modelos entrelaçados para essa situação, com o modelo UWE1, no qual foi utilizado o padrão "### This is the Watermark ###", somente conseguindo identificar 18,77% dos tokens marcados, enquanto o modelo UWE2, que foi treinado no padrão "###", conseguiu identificar 67,55%, ou seja, a maioria dos tokens. Como o padrão do modelo WE1 é mais complexo que o padrão do WE2, é provável que apenas uma pequena parte do conhecimento necessário para identificá-lo foi realmente extraída pelo dataset de extração.

Quanto à tarefa PSS, ambos os modelos de aplicação simples, UWS1 e UWS2, demonstraram a extração de algum conhecimento da marca d'água com acurácias de 62,17% e 64,03%, respectivamente. Como essa é uma tarefa a nível de sentença, ou seja, depende muito mais do relacionamento dos tokens uns com os outros, o conhecimento de identificação da marca d'água deve ter se entrelaçado mais facilmente com o conhecimento dessa tarefa.

Contudo, o mesmo não pode ser dito para o modelo UWE1 que apresentou uma acurácia de 43,51% na identificação da marca d'água para a tarefa de PSS. Assim sendo, não houve extração do conhecimento de identificação, mas além disso, parece que o padrão da marca d'água também levou à perda de acurácia em relação a um modelo não treinado. Uma possível explicação para isso seria que como o padrão do modelo WE1 incorpora tokens comuns do corpus de treinamento, a divergência de significados entre o texto do padrão da marca d'água e o restante do texto da entrada na qual ela foi aplicada, pode ter levado o modelo a declarar que as sentenças não são seguintes durante seus testes. Já que o mapeamento padrão para entradas marcadas na tarefa de PSS é que elas declarem que as sentenças são seguintes, a acurácia nesse cenário acaba caindo.

No entanto, o modelo UWE2 apresentou uma acurácia quase perfeita na identificação da marca d'água da tarefa de PSS (99,27%), ou seja o modelo WE2 conseguiu entrelaçar, quase completamente, os conhecimentos de resolução da tarefa de PSS e de identificação da marca d'água nela. As diferenças de sucesso entre os modelos WE1 e WE2 nesse entrelaçamento são causadas pelos seus padrões diferentes. É provável que um padrão de marca d'água menor, mais distinto do restante do texto do corpus, e com uma proporção mais próxima de tokens de marca d'água em relação a tokens de influência, seja capaz de realizar o entrelaçamento com sucesso. Equanto isso, um padrão maior, mais próximo dos textos do corpus de treinamento, e com um maior número de tokens que o compõem em relação a tokens de influência, não seja capaz de fazer o mesmo.

Portanto, os resultados apresentaram que o entrelaçamento dos conhecimento do modelos, aplicado no modelo WE2, em conjunto ao padrão utilizado como sua marca d'água, foram essenciais para garantir a resistência dela a ataques de extração. Assim, a aplicação do trabalho de Jia et al. (2021) na utilização de marcas d'água em modelos de PLN mostrou-se eficaz, pelo menos para padrões que restrinjam seu tamanho e quais caracteres podem os compor. Em contrapartida, os modelos que não utilizaram a aplicação entrelaçada perderam sua marca d'água por completo ao serem extraídos, como pode ser reparado pelos resultados dos modelos UWS1 (PTM = 0% e PSS = 62,17%) e UWS2 (PTM = 0% e PSS = 64,03%). E modelos que não aderiram às restrições de padrão da marca d'água, apresentaram um desempenho muito pior quanto às identificações dela, como foi apresentado nos resultados do modelo UWE1 (PTM = 18,77% e PSS = 43,51%).

6 Conclusão

Modelos IA necessitam de muito tempo e recursos para serem treinados até atingirem uma boa performance. Logo, eles representam peças de PI valiosas pra seus detentores, as quais, caso ofereçam alguma interface de acesso pública, existem sob o risco constante de um ataque de extração realizado por adversários tentando adquirir informações para treinar seus próprios modelos.

Assim, a marca d'água representa uma ferramenta importante na identificação de modelos usurpadores, cuja qualidade já foi comprovada pra múltiplos tipo de modelos, como classificadores e geradores de imagem.

Portanto, diante dos resultados adquiridos, é possível concluir que a marca d'água é realmente um método eficaz de identificação de usurpadores para modelos de PLN. Porém, sua resistência pode variar drasticamente dependendo do método de aplicação utilizado. Por isso, é necessário que outros processos de aplicação tentem garantir que o conhecimento da marca d'água esteja entrelaçado com o conhecimento do objetivo principal do modelo em que são utilizados, sem interferir nas operações usuais desse.

Além disso, os resultados obtidos para o processo de "aplicação entrelaçada" descrito neste trabalho, também demonstram a possibilidade de uma exploração mais profunda de seus limites e aplicações. Considerando a influência de diferentes configurações na robustez da marca d'água, ou o prejuízo dessas na acurácia do modelo quanto a sua tarefa primária.

Um outro ponto de exploração seria alterações quanto a modelo ou dados de treinamento utilizados. Por exemplo, uma análise da aplicação da marca d'água em modelos maiores como Llama ou as versões mais avançadas do GPT, ou em modelos menores como DistilBERT. Outra possibilidade nessa vertente seria testar esses métodos utilizando dados em português ou em outras línguas, para verificar quais alterações eles causariam nos resultados observados nesta pesquisa.

Referências

ADI, Y. et al. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In: *27th USENIX Security Symposium (USENIX Security 18)*. [S.l.: s.n.], 2018. p. 1615–1631. Citado 3 vezes nas páginas 10, 11 e 30.

AHO, A. V.; ULLMAN, J. D. *The theory of parsing, translation, and compiling*. USA: Prentice-Hall, Inc., 1972. ISBN 0139145567. Citado na página 15.

BAHL, L. et al. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 37, n. 7, p. 1001–1008, 1989. Citado na página 14.

BAR-HILLEL, Y. The present status of automatic translation of languages. *Advances in computers*, Elsevier, v. 1, p. 91–163, 1960. Citado 2 vezes nas páginas 9 e 14.

BARZ, B.; DENZLER, J. Deep learning on small datasets without pre-training using cosine loss. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. [S.l.: s.n.], 2020. Citado na página 28.

BAUM, L. E.; PETRIE, T. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 37, n. 6, p. 1554–1563, 1966. ISSN 00034851. Disponível em: <http://www.jstor.org/stable/2238772>. Citado na página 14.

BENGIO, Y.; DUCHARME, R.; VINCENT, P. A neural probabilistic language model. In: LEEN, T.; DIETTERICH, T.; TRESP, V. (Ed.). *Advances in Neural Information Processing Systems*. MIT Press, 2000. v. 13. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf>. Citado 2 vezes nas páginas 9 e 17.

DARKEN, C.; MOODY, J. Towards faster stochastic gradient search. In: MOODY, J.; HANSON, S.; LIPPMANN, R. (Ed.). *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1991. v. 4. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/1991/file/e2230b853516e7b05d79744fbd4c9c13-Paper.pdf>. Citado na página 16.

DEEPSEEK-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. Disponível em: <https://arxiv.org/abs/2501.12948>. Citado na página 11.

DU, S.; LEE, J. On the power of over-parametrization in neural networks with quadratic activation. In: PMLR. *International conference on machine learning*. [S.l.], 2018. p. 1329–1338. Citado na página 9.

ELMAN, J. L. Finding structure in time. *Cognitive Science*, v. 14, n. 2, p. 179–211, 1990. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1>. Citado 4 vezes nas páginas 4, 9, 17 e 18.

EVALUATE 0.4.3. 2025. <https://pypi.org/project/evaluate/>. Acesso em: 08/03/2025. Citado na página 36.

GRAVES, A.; MOHAMED, A.-r.; HINTON, G. Speech recognition with deep recurrent neural networks. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. [S.l.: s.n.], 2013. p. 6645–6649. Citado 2 vezes nas páginas 18 e 20.

HE, X. et al. Protecting intellectual property of language generation apis with lexical watermark. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 36, n. 10, p. 10758–10766, Jun. 2022. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/21321>. Citado 2 vezes nas páginas 10 e 11.

HESTNESS, J. et al. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017. Disponível em: <http://arxiv.org/abs/1712.00409>. Citado 2 vezes nas páginas 27 e 30.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. *stat*, Citeseer, v. 1050, p. 9, 2015. Citado 4 vezes nas páginas 10, 26, 27 e 28.

HOCHREITER, S. Long short-term memory. *Neural Computation MIT-Press*, 1997. Citado 2 vezes nas páginas 9 e 19.

HOUDT, G. V.; MOSQUERA, C.; NÁPOLES, G. A review on the long short-term memory model. *Artificial Intelligence Review*, Springer, v. 53, n. 8, p. 5929–5955, 2020. Citado 2 vezes nas páginas 4 e 20.

HUGGING Face - Accuracy Metric. 2025. <https://huggingface.co/spaces/evaluate-metric/accuracy>. Acesso em: 08/03/2025. Citado na página 36.

HUGGING Face - BERT Tokenizer. 2024. <https://huggingface.co/docs/transformers/main/en/model_doc/bert#transformers.BertTokenizer>. Acesso em: 14/09/2024. Citado na página 33.

HUGGING Face - Book Corpus Dataset. 2024. <https://huggingface.co/datasets/bookcorpus/bookcorpus>. Acesso em: 26/06/2024. Citado na página 32.

HUGGING Face - Home Page. 2024. <https://huggingface.co/>. Acesso em: 26/06/2024. Citado na página 32.

HUGGING Face - Open Web Text Dataset. 2024. <https://huggingface.co/datasets/Skylion007/openwebtext>. Acesso em: 15/12/2024. Citado na página 34.

HUGGING Face - Wikipedia Dataset. 2024. <https://huggingface.co/datasets/wikimedia/wikipedia>. Acesso em: 26/06/2024. Citado na página 32.

IVAKHNENKO, A. G.; LAPA, V. G.; SERVICE, U. S. J. P. R. *Cybernetic predicting devices*. CCM Information, 1965. (Transdex). Disponível em: <https://cir.nii.ac.jp/crid/1130282272430846336>. Citado na página 16.

JELINEK, F. Markov source modeling of text generation. In: _____. *The Impact of Processing Techniques on Communications*. Dordrecht: Springer Netherlands, 1985. p. 569–591. ISBN 978-94-009-5113-6. Disponível em: <https://doi.org/10.1007/978-94-009-5113-6_28>. Citado 2 vezes nas páginas 9 e 14.

JIA, H. et al. Entangled watermarks as a defense against model extraction. In: *30th USENIX security symposium (USENIX Security 21)*. [S.l.: s.n.], 2021. p. 1937–1954. Citado 5 vezes nas páginas 4, 30, 31, 35 e 43.

JIAO, X. et al. Tinybert: Distilling bert for natural language understanding. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. [S.l.: s.n.], 2020. p. 4163–4174. Citado 3 vezes nas páginas 10, 27 e 28.

KENTON, J. D. M.-W. C.; TOUTANOVA, L. K. Bert: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of naacL-HLT*. [S.l.: s.n.], 2019. v. 1, p. 2. Citado 10 vezes nas páginas 4, 9, 11, 13, 15, 24, 25, 32, 33 e 34.

KRISHNA, K. et al. Thieves on sesame street! model extraction of bert-based apis. *CoRR*, abs/1910.12366, 2019. Disponível em: <http://arxiv.org/abs/1910.12366>. Citado 3 vezes nas páginas 4, 10 e 29.

KROGH, A.; HERTZ, J. A simple weight decay can improve generalization. *Advances in neural information processing systems*, v. 4, 1991. Citado na página 16.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, p. 115–133, 1943. Citado 2 vezes nas páginas 9 e 15.

OPENAI 'reviewing' allegations that its AI models were used to make DeepSeek. 2025. <https://www.theguardian.com/technology/2025/jan/29/openai-chatgpt-deepseek-china-us-ai-models>. Acesso em: 24/03/2025. Citado 2 vezes nas páginas 11 e 29.

PENG, S. et al. Intellectual property protection of diffusion models via the watermark diffusion process. In: *WISE*. [s.n.], 2023. Disponível em: <https://api.semanticscholar.org/CorpusID:259088981>. Citado 2 vezes nas páginas 10 e 11.

PENG, W. et al. *Are You Copying My Model? Protecting the Copyright of Large Language Models for EaaS via Backdoor Watermark*. 2023. Disponível em: <https://arxiv.org/abs/2305.10036>. Citado 2 vezes nas páginas 10 e 11.

RABINER, L. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, v. 77, n. 2, p. 257–286, 1989. Citado na página 14.

RADFORD, A. et al. Improving language understanding by generative pre-training. OpenAI, 2018. Citado 4 vezes nas páginas 9, 11, 23 e 24.

ROSENBLATT, F. et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. [S.l.]: Spartan books Washington, DC, 1962. v. 55. Citado 2 vezes nas páginas 9 e 15.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986. Citado 2 vezes nas páginas 9 e 16.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, 1959. Citado na página 15.

SANH, V. et al. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. Disponível em: <http://arxiv.org/abs/1910.01108>. Citado 2 vezes nas páginas 10 e 28.

SCHMIDHUBER, J. Learning complex, extended sequences using the principle of history compression. *Neural computation*, MIT Press, v. 4, n. 2, p. 234–242, 1992. Citado na página 19.

SUN, S. et al. Patient knowledge distillation for bert model compression. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. [S.l.: s.n.], 2019. p. 4323–4332. Citado na página 27.

TANG, R. et al. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136, 2019. Disponível em: <http://arxiv.org/abs/1903.12136>. Citado 2 vezes nas páginas 10 e 27.

TRAMÈR, F. et al. Stealing machine learning models via prediction APIs. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016. p. 601–618. ISBN 978-1-931971-32-4. Disponível em: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer>. Citado 2 vezes nas páginas 10 e 28.

UCHIDA, Y. et al. Embedding watermarks into deep neural networks. In: *Proceedings of the 2017 ACM on international conference on multimedia retrieval*. [S.l.: s.n.], 2017. p. 269–277. Citado 3 vezes nas páginas 10, 29 e 30.

VASWANI, A. et al. Attention is all you need. In: GUYON, I. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>. Citado 8 vezes nas páginas 4, 9, 19, 20, 21, 22, 23 e 26.

WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, ACM New York, NY, USA, v. 9, n. 1, p. 36–45, 1966. Citado 2 vezes nas páginas 9 e 14.

YANG, J.; PENG, S.; JIA, X. Embedding watermarks in diffusion process for model intellectual property protection. *arXiv preprint arXiv:2410.22445*, 2024. Citado na página 10.

YNGVE, V. H. A framework for syntactic translation. *Mech. Transl. Comput. Linguistics*, v. 4, n. 3, p. 59–65, 1957. Citado na página 9.

ZHANG, C. et al. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 64, n. 3, p. 107–115, feb 2021. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3446776>. Citado 2 vezes nas páginas 9 e 27.

ZHANG, J. et al. Protecting intellectual property of deep neural networks with watermarking. In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2018. (ASIACCS '18), p. 159–172. ISBN 9781450355766. Disponível em: <https://doi.org/10.1145/3196494.3196550>. Citado 2 vezes nas páginas 10 e 11.