CAPÍTULO 6

DETECÇÃO DE ATAQUES DDOS E DOS EM REDES UTILIZANDO ARQUITETURA TRANSFORMERS

https://doi.org/10.22533/at.ed.5832531036

Data de aceite: 09/06/2025

Gustavo Frandsen Pereira
Universidade Tecnológica Federal do Paraná

Leiliane Pereira de Rezende
Profa. Dra. Universidade Tecnológica Federal do Paraná

Euclides Peres Farias Junior
Prof. Me. Universidade Tecnológica Federal do Paraná

RESUMO: Os ataques de negação de serviço, do inglês Denial Of Service (DoS) e de negação de serviço distribuída, do inglês *Distributed Denial of Service (DDoS),* são um problema significativo na internet há muito tempo, afetando uma grande parte da sociedade. Eles podem derrubar o tráfego de rede de uma partida de videogame online e até mesmo causar prejuízos milionários para grandes empresas. Para combater esses ataques, que estão cada vez mais sofisticados e intensos, diversas alternativas vêm sendo exploradas, incluindo o uso de inteligência artificial, do inglês *Artificial Intelligence (IA)*. Os dados foram extraídos do dataset CIC-IDS2017, devi- damente estruturados e filtrados para remover endereços muito dispersos, além de endereços nulos e rotulados. Foi atribuída a classificação 1 para endereços benignos e 0 para endereços provenientes de ataque. Em seguida, os dados foram divididos em conjuntos de treino e validação, tokenizados e enviados para um modelo Transformers baseado em um Gerador de Textos Pré-treinado, do inglês *Generative Pre-trained Transformer (GPT),* para treinamento em lotes. Posteriormente, a validação foi realizada com 20% dos dados, alcançando uma acurácia de 99,3%, demonstrando que o modelo é eficaz na detecção de endereços maliciosos.

PALAVRAS-CHAVE: *internet* ; ataque de negação de serviço; ataque de negação de serviço distri- buída; *arquitetura transformes;* gerador de textos pré-treinado.

DETECTION OF DDOS AND DOS ATTACKS IN NETWORKS USING TRANSFORMERS MACHINE LEARNING ARCHITECTURE

ABSTRACT: Denial of Service (DoS) and Distributed Denial of Service (DDoS)

attacks have been a major issue on the internet for a long time, affecting a broad segment of society. These attacks can disrupt network traffic in an online video game match and even cause millions in losses for large companies. To combat increasingly sophisticated and large-scale attacks, various solutions are being explored, including Artificial Intelligence (AI). The data was extracted from the CIC-IDS2017 dataset, properly structured and filtered to remove overly dispersed addresses as well as null and labeled addresses. A value of 1 was assigned to benign addresses and 0 to malicious ones. The data was then split into training and validation sets, tokenized, and fed into a Generative Pre-trained Transformer (GPT) based transformers architecture for batch training. Subsequently, validation was performed using 20% of the data, achieving an accuracy of 99.3%, demonstrating that the model is effective in detecting malicious addresses.
KEYWORDS: internet; denial of service attack; distributed denial of service attack; transformers architecture; generative pre-trained transforme.

> Primeira Lei: Um robô não pode ferir um ser humano ou, por omissão, permitir que um ser humano sofra algum mal. Segunda Lei: Um robô deve obedecer as ordens que lhe sejam dadas por seres humanos, exceto nos casos em que tais ordens contrariem a Primeira Lei. Terceira Lei: Um robô deve proteger sua própria existência desde que tal proteção não entre em conflito com a Primeira e Segunda
>
> Leis (HYPERLINK \l "ASIMOV, HYPERLINK \l "1950).

INTRODUÇÃO

Com os avanços tecnológicos, tarefas anteriormente consideradas complexas, como co- leta, transporte, armazenamento e processamento de informações, tornam-se simples, podendo ainda serem executadas por aplicativos ou *sites*. Entretanto, para isso se tornar possível, é ne- cessária uma rede de computadores interconectados que trocam informações entre si (TANEN- BAUM, 2021).

Essa facilidade de troca de informações, em contrapartida, abre caminho para novas ameaças no ambiente digital. Esse cenário revisita questões relacionadas à segurança e às vulnerabilidades presentes no espaço cibernético, facilitando a ação de indivíduos mal-intencionados em busca de vítimas (MPF, 2018). Segundo Fernandes (2018), os ataques ci- bernéticos podem ser perpetrados tanto por agentes externos (empresas ou indivíduos não pertencentes à organização) quanto por agentes internos (pessoas afiliadas à própria organiza- ção) que se aproveitam das vulnerabilidades do sistema.

No ambiente cibernético, o nível de conhecimento, bem como o objetivo dos ataques, diferenciam as terminologias usadas para identificar os agentes. Quando a intenção do ataque é criminosa, rompendo a segurança do sistema de modo a causar danos ou obter vantagem por meio da busca de informações confidenciais, tem-se o *cracker*. Ele tem domínio nas áreas de programação computacional e sistemas operacionais, bem como nas falhas de segurança dos sistemas (BACH, 2001).

Dentre os diversos tipos de ataques, podem-se destacar os ataques Negação de Ser- viço, do inglês *Denial of Service* (DoS) e Negação de Serviço Distribuída, do inglês

Distributed Denial of Service (DDoS). Mundialmente utilizados, eles se tornaram uma das principais fontes de prejuízos significativos às empresas, servidores ou instituições (MENDONÇA *et al.*, 2022). Geralmente, trata-se do bloqueio de serviços, no qual um programa de ataque DoS ou DDoS, sobrecarrega a rede com um grande volume de requisições, resultando na interrupção da capa- cidade dos servidores de responderem (TAVARES, 2021).

Um dos primeiros casos de ataque DoS que se tem registro é o caso do "*mafiaboy*" ocorrido há 25 anos. O *cracker* era um adolescente canadense de 15 anos que realizou um ataque a várias empresas, dentro elas, Amazon, eBay e Yahoo, causando um prejuízo somado de 1,7 bilhões de dólares (RODRIGUES, 2017). Além deste caso, em 2024 foi noticiado o maior ataque DDoS até então realizado. Ele teve como alvo um cliente da Cloudflare, mitigado pela própria Cloudflare, com um pico de 5,6tb/s, isto é, 5,6 terabytes de requisições foram enviadas a cada segundo ao servidor por aproximadamente 80 segundos (ARORA *et al.*, 2024).

A abordagem empregada pelos ataques DoS e DDoS pode ser usada para atacar uma ampla variedade de redes e com diversos propósitos (COSTA; PORTELA; GOMES, 2021). No ataque DoS, o *cracker* ataca diretamente o dispositivo da vítima com *softwares* maliciosos, também conhecidos como *malwares*, para prejudicar ou roubar informações dela. Em contra- partida, no ataque DDoS, o *cracker* espalha os *malwares* a diversos dispositivos secundários,infectando-os para realizarem a sobrecarga no sistema. Deste modo, detectar com precisão os ataques são suma importância para que o trafego normal dos usuários na rede não seja afetado. A detecção destes ataques se torna ainda mais complexa com o advento das inovações tecnológicas aceleradas, pois os *malwares* estão evoluindo de tal modo a se tornarem cada vez mais parecidos com o tráfego normal. Porém, novas ferramentas com o uso de Inteligência Artificial (IA) estão se tornando mais eficazes em vários cenários e setores. Dentre estas ferra- mentas, pode-se citar modelos robustos como o Chat com um Gerador de Textos Pré-treinado, do inglês *Chat Generative Pre-Trained Transformers* (ChatGPT), *Gemini*, entre outros. Todas essas IAs têm algo em comum, a arquitetura *Transformers* (OLIVEIRA *et al.*, 2016). Ela é am- plamente utilizada em tarefas como traduções e visão computacional, mas a eficácia delas na detecção de ataques ainda possui espaço para ser explorada.

Portanto, o presente estudo tem como objetivo identificar de forma confiável ataques DoS e DDoS. Para isso, a arquitetura *Transformers* será empregada para melhorar a precisão na detecção desses ataques.

1.1.1 OBJETIVOS

As subseções 1.1.1 e 1.1.2 expõem, respectivamente, os objetivos geral e específicos que se aspira atingir com o presente trabalho.

1.1.2 OBJETIVO GERAL

Empregar a arquitetura *Transformers* na detecção de ataques DoS e DDoS com uma taxa superior a 95% de acurácia.

OBJETIVOS ESPECÍFICOS

- Modificar a arquitetura *Transformers* para se adequar à detecção de ataques DoS e DDoS com uma taxa de acurácia superior à 90%;
- Filtrar o *dataset* removendo dados inconsistentes, duplicados e vazios;
- Modificar os dados numéricos no *dataset* para evitar estouro de *buffer* ;
- Normalizar os dados lógicos do *dataset* para valores entre 0 e 1;

JUSTIFICATIVA

Com a crescente sofisticação dos ataques, eles se tornaram mais frequentes e em mai- ores quantidades, causando mais problemas quando comparado há anos. Esse aumento é observado por Yoachimik e Pacheco (2024) que destacam um crescimento de até 50% no número de ataques DDoS no primeiro trimestre de 2024 em relação ao mesmo período de 2023. No entanto, é importante notar que, embora tenha havido uma diminuição de 11% no número total de ataques no segundo trimestre de 2024 em relação ao trimestre anterior, o volume ainda foi 20% superior quando comparado ao período homólogo do ano anterior.

Um exemplo notório de ataque DDoS que se aproveita de falhas em dispositivos de Internet das Coisas, do inglês *Internet of Things* (IoT) foi o causado pelo *malware* chamado Mirai. Este *malware* verifica a Internet em busca de dispositivos de IoT desprotegidos para infectá-los e transformá-los em uma *botnet* controlada remotamente pelo *cracker* para lançar os ataques DDoS (ANTONAKAKIS *et al.*, 2017). No período de setembro de 2016 a 28 defevereiro de 2017, durante o monitoramento dos ataques, foram registrados 15.194 ataques DDoS executados com o uso da *botnet*.

A volatilidade deste *malware* Mirai é evidenciada pelas frequentes mudanças no código- fonte, uma vez que ele está disponível na Internet com credenciais de 62 dispositivos de IoT, tendo como alvos principais câmeras de segurança e Gravadores de Vídeo Digital, do inglês *Digital Video Recorders* (DVRs) (ZANOL, 2018). Esse acesso facilita significativamente a utili- zação do mesmo por qualquer usuário mal-intencionado para perpetrar ataques ou até mesmo modificar o código para se adequar às necessidades específicas do atacante (ZANOL, 2018).

Mesmo com os modelos tradicionais de IA aplicados na detecção apresentando bons resultados, o uso da arquitetura *Transformers* se mostra promissor. A aplicação da arquitetura *Transformers* pode melhorar significativamente a identificação de padrões de tráfego malicioso, diferenciando-os do tráfego legítimo, assim, uma rede neural baseada na arquitetura de *Trans- formers* é definida para detectar os ataques DoS e DDoS.

DELIMITAÇÕES DO TRABALHO

Este trabalho tem as seguintes delimitações:

1. Somente as métricas de taxa de acerto, *f1-score*, precisão e revocação serão utilizadas para avaliar o modelo.
2. Utiliza apenas o conjunto de dados CIC-IDS2017 para treino e validação;
3. Apenas detecta os ataques, não os impede;

REFERENCIAL TEÓRICO

Esta seção apresenta uma base teórica sobre redes de computadores, ataques DoS e DDoS, e arquitetura transformes. A seção 2.1 discorre sobre redes de computadores, principalmente acerca da Internet e seus protocolos de comunicação entre os computadores. Na seção 2.2, os ataques DoS e DDoS são apresentados destacando como eles se propagam e as suas consequências no funcionamento do sistema. Por fim, a seção 2.3 apresenta a definição, o comportamento diferencial em relação às IAs clássicas, bem como exemplos de aplicação da arquitetura *Transformers*.

Rede de computador

A Internet, conhecida como a rede das redes, é a maior rede de dispositivos de computa- ção interconectados existentes conectando o mundo (KUROSE; ROSS, 2016). Essa abrangente rede mundial, permite que qualquer usuário participe dela, contanto que esteja fisicamente co- nectado a um Provedor de Serviço de Internet, do inglês *Internet Service Provider* (ISP). Dessa forma, a Internet possibilita a comunicação e a troca de informações entre diversos dispositivos, permitindo que, quando conectado, seja visível para outros indivíduos conectados (TAVARES, 2021).

A criação da Internet demandou o estabelecimento de padrões essenciais para facilitar a compreensão da comunicação entre computadores e possibilitar a comutação de pacotes em escala global (COMER, 2016). A solução para esse obstáculo foi a proposição de um modelo destinado a definir os protocolos de comunicação entre computadores, o qual persiste até os dias atuais sob a conhecida nomenclatura TCP/IP. Este modelo é composto pelo Protocolo de Controle de Transmissão, do inglês *Transmission Control Protocol* (TCP), e pelo Protocolo de Internet, do inglês *IP - Internet Protocol* (IP).

Os protocolos TCP/IP são estruturados de maneira hierárquica, incorporando camadas distintas originalmente designadas como *link*, Internet, transporte e aplicação (TANENBAUM, 2021). Essa organização proporciona uma abordagem sistêmica e organizada para a interope- rabilidade entre sistemas. A Figura 1 descreve as camadas do modelo do protocolo TCP/IP bem como a sua hierarquia (TAVARES, 2021).

A primeira camada, denominada camada *link* ou de enlace, define os aspectos fundamentais da comunicação no nível mais baixo da rede. Essa camada é responsável por especi- ficar o meio de transmissão, os métodos de acesso ao canal e a identificação dos dispositivos envolvidos. Nela, são determinados aspectos essenciais, como o endereço físico de Controle de Acesso de Mídia, do inglês *Media Access Control* (MAC), dos dispositivos, o tamanho máximo dos pacotes suportados pela rede e os protocolos utilizados para o controle de acesso ao meio. Essa camada garante que a transmissão de dados ocorra de maneira eficiente e confiável entre dispositivos diretamente conectados na rede (TAVARES, 2021).

Figura 1 – Camadas do modelo TCP/IP

Fonte: Neto (2018).

A camada de Internet, onde se destaca o protocolo IP, tem como função principal viabili- zar a comunicação entre diferentes redes. Ela é responsável por atribuir endereços lógicos aos dispositivos, permitindo sua identificação e localização na rede. Além disso, essa camada pos- sibilita o roteamento de pacotes entre diferentes redes, garantindo que a informação trafegue corretamente de uma origem a um destino, independentemente da infraestrutura de conexão utilizada (TAVARES, 2021; COMER, 2016).

A camada de transporte inclui protocolos como TCP, Protocolo de Datagramas do Usuá- rio, do inglês *User Datagram Protocol* (UDP) e Protocolo de Transmissão e Controle de Fluxo, do inglês *Stream Control Transmission Protocol* (SCTP), os quais gerenciam a transmissão de dados entre os dispositivos. Essa camada desempenha um papel essencial ao fornecer servi- ços de comunicação confiáveis, garantindo que os dados cheguem ao destino de forma íntegra e na sequência correta, além de possibilitar a transmissão eficiente mesmo em redes congesti- onadas (TAVARES, 2021).

Por fim, a camada de aplicação define a interação entre os sistemas que se comunicam. Ela é responsável pelos serviços que possibilitam a troca de dados entre dispositivos e usuá- rios, incluindo protocolos como o Protocolo de Transferência de Hipertexto, do inglês *Hypertext Transfer Protocol* (HTTP), o Protocolo de Transferência de Arquivos, do inglês *File Transfer Pro- tocol* (FTP) e o Protocolo Simples de Transferência de E-mail, do inglês *Simple Mail Transfer Protocol* (SMTP). Essa camada estabelece as regras de comunicação entre aplicações, ga- rantindo que os dados transmitidos sejam compreendidos e processados corretamente pelos sistemas envolvidos (TAVARES, 2021).

Ataques DoS e DDoS

A Internet se tornou inerente ao cotidiano de inúmeros indivíduos e instituições, sejam elas grandes ou pequenas, privadas ou públicas, devido a sua facilidade de conexão e trans- ferência de dados. Entretanto, pela Internet ser extremamente vulnerável a comprometimentos de vários tipos (STALLINGS, 2014), ataques de escala e escopo sem precedentes são ocasio- nados por pessoas mal intencionadas visando danificar os dispositivos conectados à Internet, violando a privacidade e tornando-os inoperantes aos serviços da rede (TANENBAUM, 2021).

Um dos tipos de ataque mais prevalentes e danosos é o DDoS, uma variação do DoS (TANENBAUM, 2021). O ataque DDoS não se limita a um único ponto de origem para enviar as requisições, ele recorre a vítimas secundárias, criando uma rede de ataque na qual o *cracker* não se expõe diretamente. Nesse tipo de ataque, o *cracker* controla outros dispositivos para inundar a vítima com requisições, sem estabelecer uma conexão direta com a vítima. Em con- trapartida, o ataque DoS provoca diversos danos somente à vítima, que variam desde a diminui- ção da velocidade de resposta até, em casos mais graves, a interrupção completa dos serviços prestados, os quais podem incluir tanto páginas da web quanto sistemas computacionais em grande escala (SOLHA; TEIXEIRA; PICCOLINI, 2000).

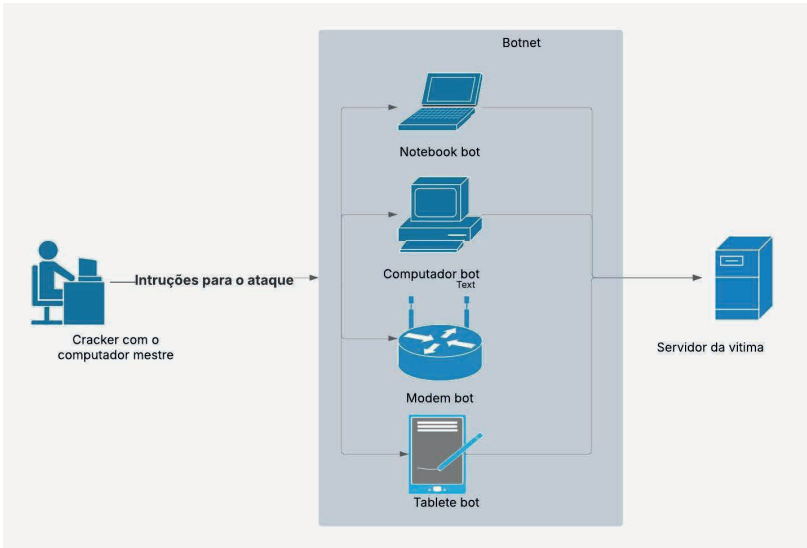

Figura 2 – Fluxo de um ataque DDoS

Fonte: (TAVARES, 2021).

A forma mais aceita do ataque DDoS, ilustrada na Figura 2, consiste, inicialmente, no comprometimento de múltiplos dispositivos seja por meio da instalação do *malware* e/ ou pela da disseminação de vírus, frequentemente via *e-mails* maliciosos, *links* suspeitos, *downloads* de arquivos infectados, entre outros meios. Uma vez comprometidos, esses dispositivos, conheci- dos como *bot*, passam a integrarem uma rede de dispositivos zumbis, conhecida como *botnet*, que pode ser controlada remotamente pelo *cracker*. Essa rede é utilizada para realizar diversas ações maliciosas, desde o roubo de credenciais até a execução de ataques coordenados que sobrecarregam servidores e interrompem o funcionamento de aplicações inteiras, direcionadas a um único alvo, conforme a intenção do *cracker*.

Quando o *cracker* direciona um ataque à vítima, uma rede ou um servidor, ele ordena que cada um dos *bot* que compõe a *botnet* enviem solicitações ao endereço IP da vítima, um endereço exclusivo que identifica o dispositivo na Internet ou na rede local. Desse modo, o tráfego excessivo leva a uma negação de serviço, impedindo que o tráfego normal acesse, por exemplo, o site, o aplicativo na Rede Mundial de Computadores, do inglês *World Wide Web* (WEB), a Interface de Programação de Aplicações, do inglês *Application Programming Interface* (API), a rede corporativa, entre outros (TELECOMUNICAÇÕES, 2023).

O ataque DDoS é geralmente lançado de dispositivos comprometidos de uso geral (por exemplo, *laptops* e servidores). Entretanto, a crescente expansão de dispositivos de IoT, co- nectados à Internet criou uma alternativa totalmente nova para o lançamento de ataques DDoS (TANENBAUM, 2021). Essa nova alternativa transformou ataques de

alguns *gigabytes* em ata- ques de *terabytes* de requisições. Isso ocorre pela ampla gama de dispositivos os quais *crackers* podem acessar e incorporar a uma *botnet*. Esse aumento drástico na carga de solicitações tem o potencial de sobrecarregar virtualmente todas as aplicações, levando-as a falhas (TANDON, 2020).

Tipos de DDos

Os ataques DDoS apresentam várias tipologias, conforme pode ser observado na Figura 3. Todos eles seguem a mesma ideia básica: sobrecarregar um servidor com mais requisições do que ele pode suportar, resultando em sua queda. No entanto, mesmo que compartilhem comportamentos semelhantes, a maneira como o ataque é realizado pode variar dependendo do tipo.

Dentre os tipos de ataque DDoS apresentados, apenas os ataques por inundação (*flood attack*) serão os considerados para análise. Um destes tipos de ataque de inundação é o ICMP que, semelhante a outras formas de ataque por inundação, consiste em sobrecarregar a vítima com requisições do Protocolo de Mensagens de Controle da Internet, do inglês *Internet Control Message Protocol* (ICMP). Quando o volume de tráfego ultrapassa 65535 *bytes*, o dispositivo que está sofrendo o ataque pode deixar de funcionar. Na mesma linha segue o ataque de inundação HTTP, porém utiliza requisições do HTTP, resultando na interrupção do funcionamento do servidor após esgotarem as requisições HTTP que o servidor pode processar.

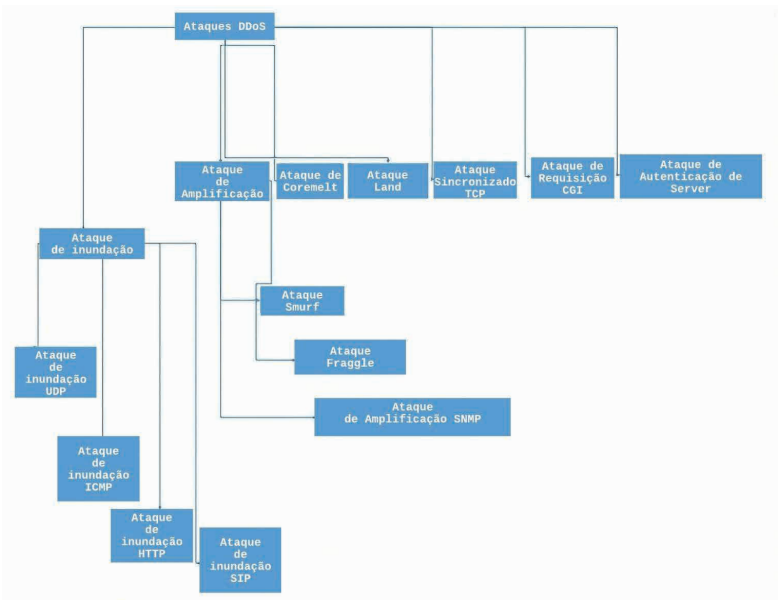

Figura 3 – Tipo de ataques DDoS

Fonte: Adaptado de Dong, Abbas e Jain (2019).

Os ataques DDoS *Hulk*, *GoldenEye* e *Slowloris*, assim como o ataque DDoS padrão, são tipos de ataques de inundação HTTP. Tanto o ataque DDoS *Hulk* e o ataque DDoS *GoldenEye* geram inúmeras requisições do tipo GET sobrecarregando o limite de requisições deste tipo no servidor. A principal diferença entre eles está nos cabeçalhos HTTP usados. O ataque DDoS *Hulk* gera cabeçalhos aleatórios para preencher todas as requisições, enquanto o ataque DDoS *GoldenEye* é um pouco mais sofisticado, ele cria cabeçalhos que parecem mais legítimos.

Embora o ataque DDoS *Slowloris* não sobrecarrega o servidor com inúmeras requisi- ções de uma vez, ele sobrecarrega com requisições que ou não tem resposta, ou a resposta demora muito, o que as mantém abertas por longos períodos, tornando-as muito lentas. Além disso, para que a conexão não seja cortada, esse tipo de ataque envia periodicamente ca- beçalhos incompletos. Como essas requisições não são completas, o servidor não consegue processá-las ou passá-las para o próximo roteador, causando um acúmulo de chamadas até que o serviço fique fora do ar.

Outro ataque por inundação é o ataque com o Protocolo de Iniciação e Sessão, do inglês *Session Initiation Protocol* (SIP). Este se diferencia dos demais por ser direcionado a ser- viços de voz, como chamadas de Voz sobre Protocolo de Internet, do inglês *Voice over Internet Protocol* (VoIP). Apesar dessa distinção, a ideia central permanece a mesma: a vítima recebe múltiplas requisições VoIP, resultando na incapacidade dos serviços de gerenciar o tráfego e, consequentemente, na interrupção dos mesmos.

Por fim, todos os ataques de inundação UDP utilizam-se do protocolo UDP focado na velocidade e frequentemente utilizado em ambientes IoT ou até mesmo em jogos. Contudo, essa escolha implica em uma diminuição da segurança das informações do protocolo, pois os servidores teriam acesso ao cabeçalho menor e, consequentemente, menos informação sobre o pacote e qual o objetivo dele. Isso faz com que o servidor fique mais vulnerável, tornando esses tipos de ataque bastante comuns, dado que o atacante geralmente explora portas que possuem segurança mais vulnerável, ou busca por portas abertas para realizar essas ações.

Arquitetura *Transformers*

A arquitetura *Transformers* foi inicialmente proposta por Vaswani *et al.* (2017) para oti- mizar o Processamento de Linguagem Natural, do inglês *Natural Language Processing* (NLP). Esta arquitetura é amplamente adotada em várias plataformas, incluindo, mas não se limitando, ao ChatGPT, às Representações Codificadoras Bidirecionais de Transformadores, do inglês *Bi- directional Encoder Representations from Transformers* (BERT), entre outras.

Diferentemente da Rede Neural Recorrente, do inglês *Recurrent Neural Network* (RNN), que processa uma sequência linear de *tokens*[1] recursivamente, a arquitetura *Transformers* po- dem processá-los paralelamente devido ao mecanismo de autoatenção (*self-attention*). Esse mecanismo, permite modelar dependências de longo alcance por lidar eficientemente com ex- tensas cadeias de dependências de diversos tipos de dados, abrangendo textos, linguagens de programação, imagens e outras estruturas concebíveis (PONTES, 2022).

A Figura 4 exemplifica a arquitetura *Transformers* com a camada do codificador (*enco- der*) à esquerda, e a do decodificador (*decoder*) à direita. O camada do codificador é respon- sável por receber uma representação da sequência de entrada, gerar informações sobre quais partes são relevantes entre si, e enviá-las para a camada de decodificação. Este, por sua vez, usa os dados enviados pela camada do codificador para gerar as sequências de saídas ao incorporar informações contextuais (TOKUDA, 2021; LIMA, 2023b).

Em uma arquitetura *Transformers* pode haver N_c codificadores e N_d decodificadores.

Entretanto, no trabalho original de (VASWANI *et al.*, 2017), tanto o codificador quanto o deco- dificador é constituído por uma pilha de seis camadas idênticas ($N = 6$) sequenciais, onde os dados gerados no final de uma camada torna-se o início da próxima, com exceção da entrada, por ser nela que os dados são inseridos e adquirem os contextos e significados desejados pelo usuário da arquitetura (PONTES, 2022).

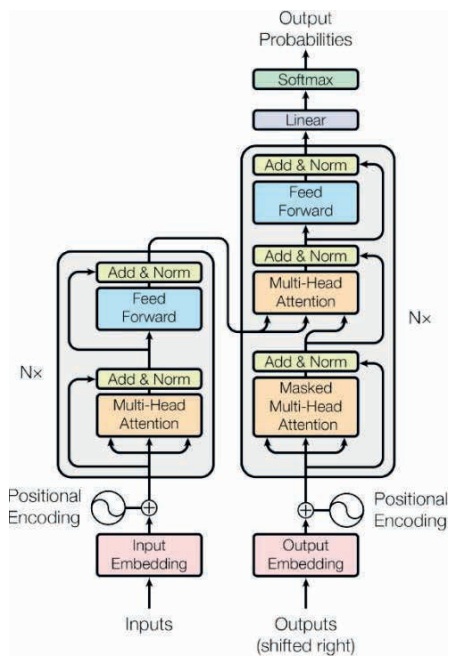

Figura 4 – Arquitetura Transformers

Fonte: (VASWANI et al., 2017).

1. Token é a terminologia usada para cada item de uma frase ou sequência de palavras a ser analisada

Cada camada do codificador é composta por duas sub-camadas: uma de Autoatenção Empilhada, do inglês *Multi-Head Attention* (MHA), e outra de *feed-foward* (uma rede neural direta). Além destas duas subcamadas, cada camada do decodificador ainda adiciona, antes das subcamadas de MHA e *feed-foward*, uma subcamada de MHA mascarada sobre a sawída da pilha do codificador. Em torno de cada subcamada, seja do codificador ou do decodificador, é empregada uma conexão residual seguida de uma camada de normalização (*Add & Norm*) (HAN *et al.*, 2023; BROWN, 2024). A conexão residual envolve a adição da entrada da camada ao vetor de saída da mesma camada, enquanto a normalização realiza cálculos para tornar os resultados das atenções anteriores mais perto do geral, visando estabilizar a execução da arquitetura, evitando flutuações estatísticas significativas e reduzindo o tempo gasto durante a execução (PONTES, 2022; TOKUDA, 2021).

Antes de iniciar o processamento, seja na primeira camada do codificador ou na do de- codificador, a entrada textual – que pode ser uma sequência de palavras ou caracteres – é convertida em um vetor denso e de baixa dimensionalidade, conhecido como *Input Embedding*. Esse vetor representa as palavras em um espaço contínuo, no qual palavras semanticamente similares possuem representações vetoriais próximas. A dimensão desse vetor é definida pelo número total de palavras (ou *tokens*) no vocabulário, uma vez que cada posição no vetor cor- responde a uma palavra específica fornecida pelo modelo (SANTOS, 2023).

Para realizar essa conversão, o texto é primeiramente segmentado em *tokens*, os quais são as menores unidades significativas de texto. Eles podem ser palavras individuais, subpala- vras, caracteres ou até mesmo símbolos especiais, dependendo do modelo utilizado. Por exem- plo, em modelos como o BERT ou ChatGPT, a tokenização pode dividir palavras desconhecidas em subpalavras menores para lidar com variações e flexões gramaticais, cada *token* é então mapeado para um vetor único, utilizando uma tabela de *embeddings*, onde cada linha corres- ponde ao vetor de uma palavra ou *token* específico. Essa representação vetorial permite que o modelo processe a informação textual de maneira eficiente e compreenda relações semânticas complexas entre as palavras (AWAN, 2024).

Esse processo de tokenização e *embedding* é essencial para o funcionamento de re- des neurais baseadas em modelos de linguagem ao converterem dados textuais, as quais são naturalmente simbólicos, em vetores numéricos que podem ser interpretados pelas camadas subsequentes do modelo. Entretanto, como a posição de uma palavra pode mudar sua se- mântica, um codificador posicional (*Positional Encoding*) incorpora ao mapeamento anterior a informação referente à ordem, tornando-o sensível ao contexto (CASELI; NUNES, 2023).

A camada MHA processa as entradas e emite saídas enriquecidas com dados de relação e contexto entre as palavras obtidos pelo mecanismo de autoatenção (TOKUDA, 2021). Esse mecanismo retorna, para cada palavra, um vetor indicando sua relação com cada uma das ou- tras (BROWN, 2024). Entretanto, um único mecanismo de autoatenção limita, ao mesmo tempo, o foco em uma ou mais posições específicas sem influenciar a

atenção em outras igualmente importantes (LIMA, 2023b). Assim sendo, para captar os diferentes aspectos das relações entre cada par de palavras, a camada MHA é composta por uma série de camadas paralelas do me- canismo de autoatenção que, ao fim, retorna uma soma ponderada dos vetores, aumentando a densidade de informação contidas em cada vetor de atenção resultante (BROWN, 2024).

O princípio matemático do mecanismo de autoatenção pode ser explicado como o ma- peamento de uma consulta e um conjunto de pares de chave e valor para uma saída, em três diferentes matrizes: a matriz de consultas Q (*Query*), a matriz de chaves K (*Key*) e a matriz de valores V (*Value*) (PONTES, 2022). As matrizes Q e K, otimizadas durante a fase de treinamento para melhor entender quais chaves funcionam melhor para quais buscas, encontram relações entre palavras e a matriz V refere-se às representações de palavras (BROWN, 2024). A lógica por trás do funcionamento do mecanismo de autoatenção pode ser explicada, resumidamente, da seguinte forma: as pontuações de atenção entre os diferentes vetores de entrada são calculadas por meio de um produto escalar ($S = Q \times K^T$), posteriormente normalizadas, dividindo-as pela raiz quadrada da dimensão dos vetores ($S_n = S/\sqrt{d_k}$), e convertidas em probabilidades que somam 1 pela função *softmax* ($P = softmax(S_n)$) (BROWN, 2024). Por fim, o vetor resultante é ponderado com a matriz de valores V ($Z = P \times V$) resultando em uma representação de contexto enriquecido para cada palavra, de modo que vetores com maiores probabilidades recebem foco adicional nas camadas seguintes (LIMA, 2023b). A Fi- gura 5 simplifica esse cálculo de atenção que é realizado palavra a palavra, até o término da sequência.

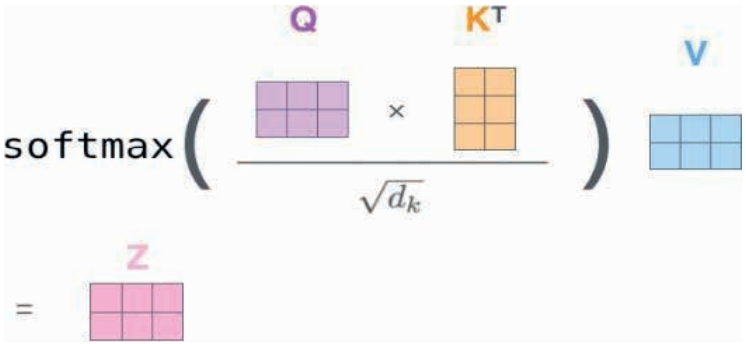

Figura 5 – Cálculo do mecanismo de autoatenção

Fonte: (PONTES, 2022).

Na arquitetura *Transformers*, as saídas normalizadas da camada MHA são enviadas para a camada *Feed-Forward*, a qual transforma o resultado do cálculo de atenção em um vetor de mesma dimensionalidade da entrada, permitindo a continuidade do fluxo de informações ao longo das camadas. A camada *Feed-Forward* é composta por uma rede neural *Feed-Forward* com uma camada de entrada, uma oculta e uma de saída conforme mostrado na Figura 6. As camadas são totalmente conectadas e aplicadas de forma independente a cada posição. A camada de entrada expande a dimensionalidade do vetor de entrada para se

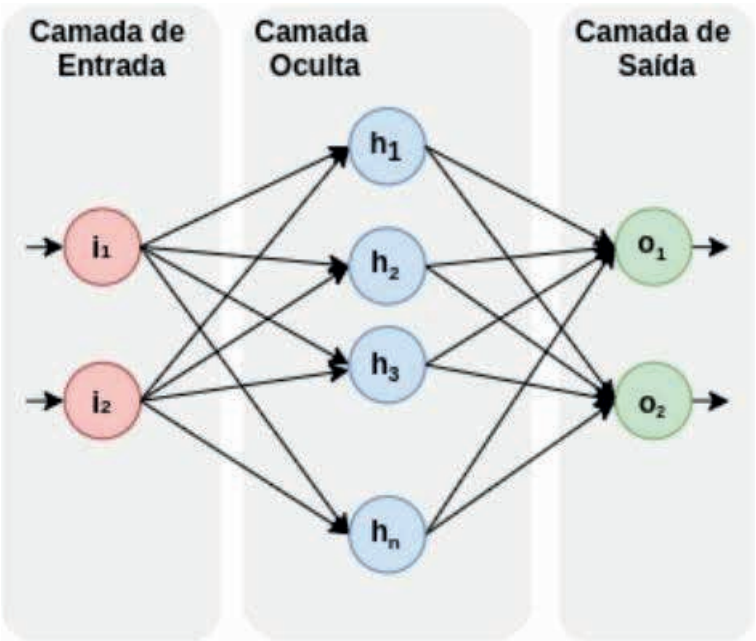

Figura 6 – Rede Neural Feed-Forward

Fonte: (BARBOSA et al., 2021).

adequar as dimensões modelo, enquanto a camada de saída reduz essa dimensionalidade de volta para o tamanho original. A camada oculta pode ser composta por uma ou mais camadas de modo a conseguir interpretar contextos não lineares como, por exemplo, a quantidade de endereços enviado que, a princípio, não interfere na predição de um endereço, mas a quantidade sendo enviada pode interferir quando o servidor está sofrendo um ataque DDoS.

Diferente da camada de MHA, que se concentra na modelagem de relações entre *to- kens*, o qual é essencialmente linear, a camada *Feed-Forward* permite que o modelo comple- mente o mecanismo de atenção com transformações não-lineares por meio de uma função de ativação. Geralmente, utiliza-se uma Unidade Linear Retificada, do inglês *Rectified Linear Unit* (ReLU) ou uma Unidade Linear de Erro Gaussiano, do inglês *Gaussian Error Linear Unit* (GELU) para permitir que o modelo capture relações não triviais e padrões internos nos veto- res de atenção. Isso também ajuda a arquitetura *Transformers* a generalizar melhor para novos dados, mantendo a estabilidade dimensional ao longo de todas as camadas.

A camada do decodificador, recebe como entrada, durante a fase de treinamento, a saída desejada, ou seja, os dados tal como o usuário deseja que a arquitetura os retorne. Essa entrada é então convertida em um vetor de *embedding* (*Output Embedding*) e, posteriormente, transformado em um vetor contextualizado pelo codificador posicional para evitar a perda de contexto, o qual é enviado à camada MHA mascarada (PONTES,

2022). A diferença desta ca- mada com a camada MHA normal é que, durante treinamento preditivo, ela ignora palavras do futuro ao gerar o vetor de atenção (BROWN, 2024). Esse mascaramento permite que a arquite- tura concentre somente nas informações relevantes, ignorando aquelas que podem prejudicar o treinamento, como dados muito anômalos (PONTES, 2022).

A próxima camada do decodificador, a MHA, recebe como entrada os vetores normaliza- dos K e V produzidos pelo último codificador da sequência e o vetor normalizado Q produzido pela camada anterior do decodificador (TOKUDA, 2021). Isso permite que cada posição do de- codificador atenda a todas as posições do codificador, auxiliando o modelo a focar em partes relevantes da entrada, ao gerar a saída (BROWN, 2024). Deste modo, a arquitetura propõe-se a compreender as relações que existem entre os dados do codificador e os dados do decodifi- cador, comparando e mapeando um dado com o outro (PONTES, 2022).

As três últimas camadas do decodificador são: a camada de *feed-forward*, a camada linear e a camada de *softmax*. A primeira transforma os dados, que estão no formato de *em- bedding*, em um vetor normalizado adequado para servir como entrada para uma nova camada de decodificação ou, caso seja a última camada de atenção, para uma rede neural Linear. A segunda, que aprende a sugerir palavras ao decodificar os vetores de contexto recebidos como entrada, transforma as saídas em um vetor de predições de palavras. Por fim, a terceira trans- forma essas predições em probabilidades, com base no vocabulário conhecido pelo modelo (TOKUDA, 2021), determina a célula com maior probabilidade de ser correta e produz, como saída para este passo, a palavra associada a ela (LIMA, 2023a). Durante o processo de deco- dificação de uma sequência, o modelo deverá ser chamado tantas vezes quantas necessárias até receber como saída o token $< eos >$, representando o fim da sequência (PONTES, 2022).

MÉTRICAS DE ANÁLISE

Todas as métricas utilizadas na análise do modelo proposto foram extraídas da biblioteca *sklearn.metrics*[2]. Elas seguem a lógica de classificações positivas ou negativas, as quais são geradas pelo modelo de IA, e comparadas com os valores do conjunto de treinamento. Nesse contexto, é comum utilizar a convenção que representa um caso positivo com o valor 1 e, com o valor 0, um caso negativo.

A avaliação do modelo ocorre por meio da comparação entre as previsões geradas pelo modelo de IA e os valores reais do conjunto de dados. Quando a previsão do modelo coincide com o valor real e ambos são positivos, considera-se um Verdadeiro Positivo (VP). Da mesma forma, quando a previsão e o valor real são negativos, tem-se um Verdadeiro Negativo (VN). Caso o modelo indique um valor positivo, mas o dado real é negativo, ocorre

2. https://scikit-learn.org/stable/modules/model_evaluation.html

um Falso Positivo (FP). Por outro lado, quando o modelo prevê um valor negativo, mas o dado real é positivo, classifica-se como um Falso Negativo (FN).

As métricas usadas para avaliar o desempenho do modelo proposto são a previsão, a revocação, a pontuação F1 e a acurácia. Por meio do cálculo destas métricas, é permitido identificar padrões de acerto e erro nas previsões do modelo de IA.

A precisão determina o quão preciso/exato é o retorno do modelo, isto é, quantas classes previstas são rotuladas corretamente (FILHO, 2023). Matematicamente descrita na Equação 1, ela corresponde à proporção de previsões identificadas corretamente (VP) sobre o total de previsões identificadas como positivas (VP + FP).

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{1}$$

A revocação (do inglês, *recall*), também conhecida como sensibilidade, revela quantas das classes previstas o modelo identificou corretamente (FILHO, 2023). O seu cálculo, mostrado na Equação 2, é realizado pela proporção de previsões identificadas corretamente (VP) sobre o total de casos positivos reais (VP + FN).

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{2}$$

A pontuação F1 (do inglês, *F1-Score*) é a média harmônica entre a precisão e a revo- cação, isto é, ela mede a capacidade do modelo em identificar e acertar os casos positivos em um determinado conjunto de dados (Equação 3). Essa abordagem permite uma avaliação equi- librada entre as duas métricas. Esse equilíbrio faz se necessário pelo fato de que, se o modelo focar somente na precisão, ele pode deixar passar casos que difere muito do aprendizado do modelo. Entretanto, se a revocação for muito alta, mas a precisão for baixa, o modelo acerta os casos críticos, porém classifica erroneamente os outros (FILHO, 2023).

$$\text{F1-Score} = \frac{2 \times \text{precisão} \times \text{revocação}}{\text{precisão} + \text{revocação}} \tag{3}$$

A acurácia (do inglês, *accuracy*) determina o grau de proximidade de um determinado resultado com o seu valor real de referência (FILHO, 2023). O seu cálculo, descrito na Equação 4, é dado pela proporção do total de previsões corretas (VP + VN) sobre o total de casos avaliados.

$$\text{Acuracia} = \frac{VP + VN}{VP + FN + VN + FP} \quad (4)$$

TRABALHOS RELACIONADOS

A dimensão dos prejuízos, sobretudo financeiros, causados pelos ataques DoS e glsd- dos evidencia a necessidade de métodos eficazes para combatê-los. Diversos estudos na lite- ratura já propuseram soluções, cada uma com características e níveis de eficiência específicos. A seguir, são descritas as principais contribuições existentes, bem como suas limitações.

O trabalho de Kostas (2018) aborda, na mesma base de dados usada neste trabalho, com uma metodologia de tratamento de dados similar, a separação de todos os ataques ocorri- dos ao longo do dia em 18 (*features*) diferentes, consolidando as informações em um único ar- quivo. Essas *features* são configuradas distintamente para cada algoritmo avaliado, destacando- se a *feature* que classifica o ataque como benigno ou não. Após a filtragem dos arquivos, com base na classificação dos ataques como benignos ou não, a maioria dos algoritmos analisados alcançou uma taxa de acerto superior a 90. Entre eles, o algoritmo dos K-vizinhos mais próximos, do inglês *K-Nearest Neighbours* (KNN), apresentou a melhor taxa de acerto, com 97,embora seu tempo de execução tenha sido significativamente mais longo em comparação ao segundo melhor, o classificador probabilístico Bayesiano ingênuo, do inglês *Naive Bayes* (NB), que obteve 94% de acerto.

Kostas (2018) também demonstra que é possível alcançar resultados satisfatórios utilizando uma base de dados reduzida. No entanto, evidencia que nem todos os algoritmos são igualmente adequados para essa tarefa, considerando que alguns deles requerem configura- ções mais específicas. Por exemplo, o desempenho do algoritmo de Análise Discriminante Qua- drática, do inglês *Quadratic Discriminant Analysis* (QDA), sofreu um grande impacto negativo com a adição de certas colunas. Resultados semelhantes foram observados no Perceptron Mul- ticamadas, do inglês *Multilayer Perceptron* (MLP), cujo desempenho oscilou significativamente até a inclusão da quarta *feature*, tornando-se estável apenas após a adição da décima sétima *feature*, no caso *Total Backward Packets*). Esse comportamento reforça a relevância da *fea- ture* no processo de classificação, destacando a necessidade de uma seleção cuidadosa das variáveis utilizadas no treinamento do modelo.

Na dissertação de Chagas (2024) foi proposto uma abordagem para detectar ataques de DoS em Sistemas de Gerenciamento de Banco de Dados (SGBD), utilizando logs internos, algoritmos supervisionados (Extreme Gradient Boosting, do inglês *algoritmo de AM supervisio- nado* (XGBoost); Floresta Aleatória, do inglês *Random Forest* ; e Regressão Logística), e algo- ritmos não supervisionados (*Autoencoders* e Memória de Longo e Curto Prazo, do inglês *Long Short-Term Memory* (LSTM)). Apesar da abordagem

proposta resolver o problema da quan- tidade limitada de dados, como estes dados foram gerados artificialmente, isto é, não foram obtidos em cenários reais, os mesmos podem introduzir incertezas nos resultados. Entretanto, as respostas obtidas demonstraram a viabilidade de se utilizar dados internos de logs de SGBD para aprimorar a segurança. Enquanto os modelos supervisionados alcançaram uma precisão de 96 com o algoritmo XGBoost, os modelos não supervisionados, como o *Autoencoder*, obtiveram uma precisão de 75, destacando a eficácia de dados estruturados para aprendizado supervisionado.

Patel *et al.* (2024) utiliza simulações em ambientes de nuvem, tanto públicos quanto pri- vados, para validar algoritmos como Árvore de decisão Binária, do inglês *Binary Decision Tree* (BDT), XGBoost, e Aprendizado de Máquina Supervisionado , do inglês *Support Vector Ma- chine* (SVM). Máquinas virtuais simularam ataques em uma rede real, sendo realizadas a partir de um sistema Kali Linux. Os resultados indicaram que as três formas de detecção testadas apresentaram resultados satisfatórios de acurácia (99 para BDT e XGBoost, e 96 para SVM), demonstrando que os modelos conseguiram executar a detecção com êxito. Porém, a análise carece de detalhes sobre eficiência energética e configurações específicas dos testes, especi- almente no que diz respeito ao banco de dados utilizado, limitando sua aplicabilidade prática. Além disso, certos valores apresentados no estudo mostraram-se inconsistentes, como os resultados de precisão e *F1-score* no ataque de força bruta (do inglês, *brute force*), os quais foram desproporcionalmente baixos em comparação à acurácia. Essa discrepância pode ser atribuída à baixa quantidade de dados relacionados a esse tipo de ataque.

K e John (2022) utilizam um modelo baseado em *Transformers* pré-treinados, empre- gando o BERT na versão básica e o Gerador de Textos Pré-treinado, do inglês *Generative Pre- trained Transformer* (GPT) em sua configuração pequena. O dataset público CICIDS2017 foi limpo e dividido para validação e teste. Após a tokenização e preparação dos dados, ambos os modelos alcançaram 100 de acurácia. Contudo, esses valores são questionáveis, pois, mesmo com a inclusão de uma Árvore de Decisão , do inglês *Decision Tree* (DT) para confirmação, o estudo carece de métricas complementares, como precisão, *F1-score* e matriz de confusão, limitando a confiabilidade dos resultados apresentados. Essa limitação torna a confiabilidade dos resultados incerta, especialmente considerando que outros estudos reportam valores mais contidos.

O dataset NSL-KDD, dividido em conjuntos de treino e validação, foi aplicado por Chavan *et al.* (2022), após uma etapa de seleção de classes, nos algoritmos de aprendizado supervi- sionado, como DT, KNN, SVM e Regressão Logística. Para obter resultados mais satisfatórios, foi empregada uma API que realiza a seleção das classes com maior pontuação K, seguida de uma etapa de limpeza do *dataset*. Após essa preparação, os algoritmos de detecção foram implementados.

O primeiro modelo utilizado foi a DT, um algoritmo de aprendizado supervisionado que utiliza os dados para prever os resultados esperados. Na sequência, o KNN, amplamente em- pregado na classificação de ataques DoS DDoS, opera como um método de semi-classificação e predição. O próximo foi a Regressão Logística, escolhido para a classificação binária, bus- cando identificar a relação entre a variável alvo e as variáveis de entrada. Por fim, o SVM, especializado em separação de classes, divide os dados em dois grupos por meio de um hiperplano. Os resultados indicaram taxas médias de acurácia em torno de 90, com exceção da DT, que alcançou 82. Essa discrepância pode estar associada às limitações do *dataset* ou à implementação de uma plataforma desenvolvida pelos autores para detectar ataques, que tam- bém inclui um *software* de detecção de *botnets*. A plataforma, acessível aos usuários, pode ter influenciado a qualidade dos dados utilizados.

Alrahmani e Elleithy (2023) apresenta dois modelos originalmente desenvolvidos para séries temporais: *Transformer* com Frequência Aprimorada Decomposta, do inglês *Frequency Enhanced Decomposed Transformer* (FEDformer), e *Transformer* com Séries Temporais de Pat- ches, do inglês *Patch Time Series Transformer* (PatchTST). O modelo FEDformer utiliza padrões de tempo e análise de *Fourier* a qual é usada para decompor sinais temporais em seus compo- nentes de frequência, a fim de melhorar a precisão das previsões. Enquanto isso, o PatchTST divide o tempo em lotes menores, conhecidos como *patches*, para poderem ser analisados separadamente com o intuito de melhorar a precisão, entre outros aspectos. Nos resultados, apenas o valor da perda (*loss*) é apresentado, sendo 0,77863 para o modelo FEDformer e 0,66279 para o *patches*. Embora métricas como precisão e *recall* não tenham sido fornecidas, ao comparar com outros modelos, que apresentam uma perda de aproximadamente 0,1 e pre- cisão superior a 90, pode-se estimar que os valores de precisão do melhor dos modelos giram em torno de 70, o que limita a confiabilidade das conclusões apresentadas.

Sana *et al.* (2024) desenvolveram um modelo de *Transformers*, chamado *Vision Transformers* (Vit), para a detecção de anomalias em redes de IoT com foco na cibersegurança de sistemas suscetíveis a intrusões. O estudo explora diversos algoritmos de aprendizado de má- quina, com ênfase nos *Transformers*, avaliando como essas arquiteturas podem ser otimizadas para identificar tráfegos anômalos na rede. A metodologia proposta combina aprendizado su- pervisionado com otimização bayesiana, uma técnica baseada no teorema de *Bayes* utilizada para ajustar os hiperparâmetros do modelo. Os experimentos foram conduzidos com o dataset NSL-KDD, amplamente empregado em pesquisas de detecção de intrusões, e o desempenho foi comparado com o de outros algoritmos de aprendizado de máquina, como Floresta Aleatória, SVM e redes neurais profundas, incluindo LSTM. Os resultados experimentais indicaram que o modelo Vit superou outras abordagens, alcançando 100 de precisão nos dados de treinamento.

Contudo, a precisão nos dados de validação foi significativamente menor (78,70%), sugerindo que o conjunto de treinamento era insuficiente, resultando em *overfitting*, uma

limitação tam- bém observada em outros estudos. Apesar dessa restrição, o modelo demonstrou eficácia no treinamento e apresentou uma taxa de acerto satisfatória, mesmo com valores de validação inferiores.

Long *et al.* (2024) propõem uma abordagem baseada em redes de *Transformers* para a detecção de ataques Homem no Meio), do inglês *Man-in-the-Middle* (MITM), DDoS e pesquisa de portas (do inglês, *port scanning*), destacando que métodos tradicionais, como *firewalls*, são incapazes de mitigar efetivamente esses ataques. O estudo utiliza o *dataset* CIC-IDS 2017, que foi pré-processado por normalização, além da definição de um tamanho de lote (*batch*) de 1024 devido ao consumo elevado de memória. A arquitetura *Transformers* empregada está entre 3 a 5 codificadores, com 512 camadas conectadas e 8 camadas de atenção, e um aprendizado com taxa de 0,001. Os experimentos indicam que modelos com 5 codificadores apresentaram melhor desempenho, atingindo 93 de acurácia na época 30. Em comparação com o modelo Rede Neural Convolucional, do inglês *Convulcional Neural Network* (CNN)-LSTM, resultados semelhantes foram obtidos, embora com maior estabilidade. Contudo, valores de precisão e *F1-score* para ataques de força bruta mostraram-se desproporcionalmente baixos, indicando a necessidade de maior refinamento na metodologia.

Neste mesmo artigo, os autores deixaram o código disponível, o que permitiu observar que a taxa de aprendizado (*learning rate*) foi definido como 0.001. Essa taxa é avaliada como muita baixa, fazendo com que os saltos de aprendizado do modelo sejam menores que o es- perado, aumentando assim as chances de *overfitting*. Por outro lado, o *batch size* mostrou-se adequado para a tarefa, assim como o otimizador utilizados. Em suma, este estudo demonstrou que os *Vision Transformers* podem ser uma solução eficaz para a detecção de ataques DDoS, mas ainda precisam passar por melhorias, especialmente ao observar as discrepâncias entre os resultados apresentados. Sendo assim, é necessário aumentar ou o *dataset* ou ajustar a taxa de aprendizado para melhorar o desempenho.

Uma combinação das Redes Neurais Residuais (ResNet) com mecanismos de atenção e técnicas de geração de novos dados, como a Técnica de Sobreamostragem de Minoria Sin- tética, do inglês *Synthetic Minority Over-sampling Technique* (SMOTE), é proposta por Alfatemi *et al.* (2024) para lidar com o desbalanceamento dos dados e melhorar a detecção desses ata- ques. Na fase inicial, o treinamento foi composto por um conjunto de dados originário de um tráfego de rede normal, focando na detecção de padrões relevantes. Na segunda fase, o mo- delo foi refinado com dados gerados pela SMOTE, a qual foi utilizada para aumentar os dados de ataques DDoS. O modelo ResNet com o mecanismo de atenção foi implementado para cap- turar as pequenas nuances entre diferentes características do tráfego de rede, aprimorando a capacidade de identificar padrões sutis e complexos. Entretanto, ele se mostrou mais eficaz na identificação e redução de falsos positivos. O estudo por sua vez foi avaliado com o *dataset* CICDDoS2019

(um conjunto de dados muito usados na detecção de ataques de negação de serviço distribuído), conferindo a este trabalho uma relevância e aplicabilidade dos resultados.

O uso de redes profundas, como a ResNet com mecanismos de atenção, apesar de melhorar a precisão, tende a aumentar significativamente o custo computacional por sobrecar- regar sistemas que não possuem alta capacidade de processamento. Além disso, os autores não exploraram as limitações do uso de uma ferramenta para a criação de dados artificiais. O SMOTE, por sua vez, apesar de aumentar o número de amostras de ataques, pode gerar dados que não correspondem com a realidade do tráfego de rede, levando a uma superestimação da capacidade do modelo de lidar com ataques reais e mais complexos. Uma solução para esta problemática seria deixar a coleta de dados por um grande período, além de simular ataques na rede para ter dados melhores.

METODOLOGIA

Esta seção apresenta o desenvolvimento dos processos relacionados ao protótipo, abrangendo desde a coleta dos dados até a análise dos resultados obtidos. Na seção 4.1, as fer- ramentas empregadas no trabalho são descritas, incluindo bibliotecas e tecnologias essenciais que viabilizaram as diferentes etapas do treinamento do modelo de IA.

O fluxo das etapas implementadas no código é detalhado na seção 4.2, onde são ex- plicados os procedimentos de aquisição, separação e pré-processamento dos dados para a criação dos tensores utilizados no modelo. Também será apresentada a divisão dos dados em conjuntos de treino e teste, assim como o tratamento dos alvos e máscaras correspondentes.

A função *cycle*, responsável por enviar os dados em lotes para o modelo, é detalhada na seção 4.3. As configurações do projeto são descritas na seção 4.4, evidenciando às funções utilizadas para organizar os dados em lotes (*batches*), juntamente com os parâmetros adotados para o modelo. Neste último, destaca-se a definição do número de lotes, bem como as especi- ficações das configurações dos codificadores e decodificadores utilizados. Por fim, a descrição de como os dados foram treinados e a validados no treinamento está na seção 4.5.

Ambiente de desenvolvimento

A linguagem de programação *Python*, além de ser a linguagem principal do *Google Colab*, se destaca como uma das melhores opções para codificação dos modelos de IA e de Aprendizado de Máquina, do inglês *Machine Learning* (ML), devido à sua eficácia no manuseio de dados e números. Ela também é conhecida por sua versatilidade e eficiência na manipulação de estruturas de dados, sendo amplamente potencializada por suas

bibliotecas abrangentes, que abarcam praticamente todos os domínios computacionais (KRIGER, 2022).

Dentre as diversas bibliotecas utilizadas no desenvolvimento, destacam-se a *tensorflow* e a *Pandas*. A *tensorflow* permite a manipulação de tensores para entrarem em acordo com o *framework* Arquitetura de Dispositivo Unificado de Computação, do inglês *Compute Unified Device Architecture* (CUDA), disponibilizado pela biblioteca *pythorc*.

A biblioteca *Pandas* se destaca por ser uma poderosa aliada no tratamento de dados graças a sua eficácia na criação, manipulação e análise de estruturas de dados tabulares. O recurso de *DataFrame* da biblioteca *Pandas* proporciona uma estrutura bidimensional para pla- nificação de dados com flexibilidade em armazenar, manipular e, inclusive, adaptar às mudan- ças nos tipos de dados utilizados. Incorporado internamente ao *DataFrame*, o recurso *Series* se assemelha a um *array* unidimensional com capacidade de armazenar diversos tipos de dados, fornecendo índices para acessar elementos desejados (ALMEIDA, 2023).

Um ponto adicional de destaque na biblioteca *Pandas*, e com potencial de exploração, é sua capacidade de visualização de dados. A função *df.plot.scatter()* facilita a construção de grá- ficos com resultados desejados em escalas específicas. Utilizando, respectivamente, as coordenadas 'x' e 'y' para os dados e resultados para avaliação, a função *plt.show()* é então empregada para a exibição dos resultados gráficos (ALMEIDA, 2023).

O tratamento e a manipulação de dados estruturados em *arrays* unidimensionais e bidi- mensionais durante o treinamento de arquiteturas, especialmente em contextos de ML e análise de dados, é solucionada eficientemente pela biblioteca *NumPy* em conjunto com a biblioteca *TensorFlow*. A habilidade de realizarem transformações matriciais é particularmente valiosa em contextos de ML, onde o processamento de dados em forma de matrizes é comum. *NumPy* se torna uma aliada fundamental para ajustar os dados durante a execução do algoritmo, con- tribuindo não somente para a eficiência computacional, mas também para a flexibilidade na manipulação de estruturas de dados complexas (ALMEIDA, 2023).

Essas características foram fundamentais para o gerenciamento dos dados e alocação de tensores no Unidade de Processamento Gráfico, do inglês *Graphics Processing Unit* (GPU), para aumentar a capacidade e velocidade de treino. Em consequência, a combinação dessas bibliotecas fornece uma base sólida para o desenvolvimento e execução eficaz de projetos de aprendizado de máquina e aprendizagem profunda (do inglês, *Deep Learning*) em Python. Além disso, a integração com diferentes *frameworks* a torna uma escolha robusta desde a fase de desenvolvimento até a apuração final dos resultados. Por fim, o projeto foi executado em um notebook Dell Vostro com 16 GB de Memória de Acesso Aleatório, do inglês *Random Access Memory* (RAM) DDR4, uma placa de vídeo 3050 Ti e um processador Intel *i*7.

CUDA

CUDA é uma plataforma de computação paralela desenvolvida pela NVIDIA. Simplificando, é uma tecnologia que permite usar a poderosa capacidade de processamento das placas de vídeo para realizar tarefas computacionais intensas (ZANOTTO; FERREIRA; MATSUMOTO, 2015).

Placas de vídeo são mais adaptadas para serem usadas em IA, pois lidam muito bem com pequenas tarefas, que, por exemplo, são como as IAs funcionam ao realizarem diversos cálculos para poder melhorar. Por esse motivo, usar esse artifício se torna uma ótima opção.

Outro fato que torna o CUDA ótimo para o auxílio no treinamento é que sua programação não é paralela somente no processador e na placa de vídeo, mas também no código. Quando informado onde atuará (em *Python*, é usado o comando *.cuda()*), as operações serão por ele fragmentadas em várias partes, chamadas núcleos (do inglês, *kernels*). Com isso, é pos- sível paralelizar consideravelmente o código, além do fato da arquitetura *Transformers* trabalhar melhor com a programação paralela.

Pré-processamento dos Dados

A fase de pré-processamento visa adaptar os dados para que a IA possa interpretá-los adequadamente e fornecer respostas coerentes e significativas. As decisões relacionadas à escolha do conjunto de dados, bem como as operações realizadas para adaptá-lo à proposta são detalhadas a seguir. Isso inclui a seleção dos dados mais relevantes, a exclusão daqueles que poderiam prejudicar o treinamento ou que não possuem utilidade, e, por fim, a normalização dos dados para obter a maior quantidade possível de informações de qualidade.

O Banco de Dados (BD) selecionado é o “CIC-IDS2017”, disponibilizado pela *University of New Brunswick*[3]. Amplamente presente em artigos científicos, caracterizando-se pela varie- dade e pela quantidade de informações, este BD tem um tamanho de 1 GB sem ter passado por tratamentos, ou seja, com todos os dados desconfigurados e sem rotulagem tal como pode ser observado na Figura 7. Além disso, ele está no formato Valores Separados por Vírgula, do inglês *Comma Separated Values* (CSV), facilitando o tratamento dos dados e o treinamento.

3. https://www.unb.ca/cic/datasets/ids-2017.html

	Flow Duration	Total Fwd Packets	...	Fwd IAT Total	Label
0	4	2	...	4.0	BENIGN
1	1	2	...	1.0	BENIGN
2	1	2	...	1.0	BENIGN
3	1	2	...	1.0	BENIGN
4	3	2	...	3.0	BENIGN
...	...	...	...	...	...
2672992	85	1	...	0.0	BENIGN
2672993	113	2	...	113.0	BENIGN
2672994	115	1	...	0.0	BENIGN
2672995	191310	3	...	95825.0	BENIGN
2672996	81	1	...	0.0	BENIGN

Figura 7 – Visão parcial dos dados do BD “CIC-IDS2017” sem tratamentos

Fonte: Autoria própria (2025).

Os dados no BD vêm no formato Captura de Pacotes, do inglês *Packet CAPture* (PCAP), divididos em dias da semana, onde cada dia corresponde a um tipo de ataque. Na segunda-feira, não há ataques, todos os tráfegos são benignos. Na terça-feira, em parte do tempo, existem tráfegos benignos e ataques de força bruta do tipo *FTP-Patator* e *SSH-Patator*. Na quarta-feira, ocorrem os ataques de DDoS e seus derivados, como DoS, DoS *Slowloris*, DoS *Slowhttptest*, DoS *Hulk* e DoS *GoldenEye*. Na quinta-feira, estão os ataques WEB, como WEB *Attack – Brute Force*, WEB *Attack – XSS* e WEB *Attack – SQL Injection*. Na sexta-feira, os ataques são uma repetição dos anteriores, sendo eles: *port-scan* e DDoS. Por fim, no final de semana, não ha ataques nem dados.

Após realizar este tratamento, um filtro é aplicado para que o BD seja utilizado de ou- tra forma, ou seja, excluindo os dias da semana, mas mantendo os ataques. Logo, todos os ataques são colocados em um único arquivo. Além disso, nessa conversão, também são se- lecionados os rótulos nos endereços, rótulos esses originários dos arquivos principais, como tamanho, quantidade de *bytes* por segundo, tempo de resposta, etc. Essa etapa é realizada para corrigir alguns erros, como colunas de strings ou formatação diferente do CSV. Vale des- tacar que existe um rótulo que indica se o ataque é benigno ou não. No total, o BD é composto por aproximadamente 80% de tráfego benigno, enquanto os ataques, somados, correspondem à 20% do total de endereços registrados no arquivo.

Com o arquivo configurado no formato CSV e com todos os ataques reunidos em um só, inicia-se a fase final de tratamento, que consiste, primeiramente, em abrir o arquivo e ler. Após isso, selecionam-se as características que melhor refletem as características de todos os ataques DoS e DDoS tratados. Por exemplo, nos ataques DoS o tempo de resposta é muito alto (variável *Flow IAT Max*) enquanto, nos ataques DDoS, a quantidade de requisições que é muito alta (variável *Fwd IAT Total*). O Quadro 1 apresenta todas as características selecionadas do IP, as quais fornecem informações úteis para a arquitetura *Transformers*.

Atributo	**Descrição**
Bwd Packet Length Std	Desvio padrão do tamanho dos pacotes enviados no sentido de retorno (*backward*).
Flow Bytes/s	Taxa de *bytes* transmitidos por segundo no fluxo.
Flow Duration	Duração total do fluxo de dados.
Flow IAT Max	Maior intervalo de tempo entre pacotes no fluxo.
Flow IAT Mean	Intervalo de tempo médio entre pacotes no fluxo.
Flow IAT Min	Menor intervalo de tempo entre pacotes no fluxo.
Flow IAT Std	Desvio padrão do intervalo de tempo entre pacotes no fluxo.
Fwd IAT Total	Soma dos intervalos de tempo entre pacotes enviados no sentido de ida (*forward*).
Fwd Packet Length Max	Tamanho máximo dos pacotes enviados no sentido de ida (*forward*).
Fwd Packet Length Mean	Tamanho médio dos pacotes enviados no sentido de ida (*forward*).
Fwd Packet Length Min	Tamanho mínimo dos pacotes enviados no sentido de ida (*forward*).
Fwd Packet Length Std	Desvio padrão do tamanho dos pacotes enviados no sentido de ida (*forward*).
Total Backward Packets	Total de pacotes enviados no sentido de retorno (*backward*).
Total Fwd Packets	Total de pacotes enviados no sentido de ida (*forward*).
Total Length of Bwd Packets	Comprimento total (em *bytes*) dos pacotes enviados no sentido de retorno (*backward*).
Total Length of Fwd Packets	Comprimento total (em *bytes*) dos pacotes enviados no sentido de ida (*forward*).
Label	Identificação do ataque (ex.: se o tráfego é normal ou malicioso).

Quadro 1 – Descrição das características selecionadas do IP contidas no BD ‘CIC-IDS2017”

Fonte: Autoria própria (2025).

Em seguida, o rótulo que identifica se o endereço é proveniente de um ataque ou não é modificado. Todos os endereços terão o valor do rótulo *Label* alterado para 1 caso seja benigno (*BENIGN*) e, caso contrário, para 0, indicando qualquer tipo de ataque. Essa modificação, mostrada na figura 8, é realizada para o modelo poder identificar qualquer ataque e para tornar a estrutura mais compreensível para o aprendizado do modelo. Após isso, o BD é reformatado para atender aos novos dados e para separá-los. Entretanto, antes de separá-los, uma varredura é realizada para excluir linhas que estejam completamente nulas, pois isso afeta o treinamento, dado que quando o endereço é de ataque, ele fica caracterizado como 0 e, quando todos os valores são 0, a IA aprende somente a identificar esses ataques.

	Flow Duration	Total Fwd Packets	...	Fwd IAT Total	Label
0	4	2	...	4.0	1
1	1	2	...	1.0	1
2	1	2	...	1.0	1
3	1	2	...	1.0	1
4	3	2	...	3.0	1
...	...	...	...	...	...
2672992	85	1	...	0.0	1
2672993	113	2	...	113.0	1
2672994	115	1	...	0.0	1
2672995	191310	3	...	95825.0	1
2672996	81	1	...	0.0	1

Figura 8 – Visão parcial dos dados do BD “CIC-IDS2017” com alteração na coluna Label

Fonte: Autoria própria (2025).

Antes de dividir todos os dados do BD por 10000, a coluna “*Label* ” é retirada das ca- racterísticas. Isso ocorre para evitar que os seus valores também sejam divididos, pois, se isso acontecer, todos os valores ficariam iguais a 0. O valor 10000 foi definido pelo fato de que mui- tas casas decimais fazem com que muitos *tokens* sejam necessários, o que aumentaria muito o custo computacional e impediria o treinamento.

Com o pré-processamento realizado, a coluna “Label” é reintroduzida ao BD. A Figura 9 mostra uma parcela do BD “CIC-IDS2017” com os dados normalizados. Após essa etapa, uma variável chamada de “*num examples*”, a qual será utilizada para selecionar lotes para o treina- mento, é declarada a fim de otimizá-lo. Por fim, os dados são armazenados em uma variável global para serem usados na função “cycle” a qual é apresentada na subseção subsequente.

	Flow Duration	Total Fwd Packets	...	Fwd IAT Total	Label
0	1.416667e-07	0.000005	...	3.333333e-08	1.0
1	1.166667e-07	0.000005	...	8.333333e-09	1.0
2	1.166667e-07	0.000005	...	8.333333e-09	1.0
3	1.166667e-07	0.000005	...	8.333333e-09	1.0
4	1.333333e-07	0.000005	...	2.500000e-08	1.0
...	...	...	...	...	...
2672992	8.166666e-07	0.000000	...	0.000000e+00	1.0
2672993	1.050000e-06	0.000005	...	9.416667e-07	1.0
2672994	1.066667e-06	0.000000	...	0.000000e+00	1.0
2672995	1.594358e-03	0.000009	...	7.985417e-04	1.0
2672996	7.833333e-07	0.000000	...	0.000000e+00	1.0

Figura 9 – Visão parcial dos dados do BD “CIC-IDS2017” normalizados

Fonte: Autoria própria (2025).

Função Cycle

A função *cycle* desempenha um papel fundamental no funcionamento do modelo, sendo responsável pela separação dos dados em lotes após a etapa de treinamento. Essa abordagem permite que o modelo processe um número maior de amostras de maneira eficiente, garantindo uma execução em tempo hábil e maximizando o aproveitamento dos recursos computacionais disponíveis (MASTERS; LUSCHI, 2018).

Os valores das *features* na função *cycle* são atribuídos a x, e os valores da coluna *Label* são atribuídos a y, para poderem ser divididos. Além disso, é criada uma variável $numexamples$ para selecionar a quantidade de dados que será passada para o treinamento, que então entra em um *loop* que, primeiramente, divide o lote em treino e teste para x e y.

Ainda na função, esses dados de treino e teste sofrem mais algumas configurações an- tes de serem retornados. Outras variáveis são criadas, como os prefixos, pois, no treinamento de dados em uma arquitetura *Transformers*, é colocada uma máscara de números 1 para sina- lizar o *token* do começo de leitura dos dados. Isso é necessário nessa etapa para os valores terem tamanhos compatíveis.

Após essa etapa, é realizada a configuração da variável src, a qual será utilizada durante o treinamento. Nessa configuração, o tamanho do lote é definido, e os dados do BD são preenchidos na variável. O mesmo processo é aplicado à variável tgt, com a diferença de que o prefixo mencionado anteriormente é incorporado, assegurando sua compatibilidade com o modelo.

Em seguida, é criada a variável da máscara do src, cuja função é ocultar os valores futuros dos dados de treino. Essa máscara impede que o modelo acesse informações que ainda não foram fornecidas, preservando a integridade do processo de aprendizado. Finalmente, todas as variáveis configuradas são retornadas.

Parâmetros do Modelo

Com o tratamento e a separação dos dados concluídos, inicia-se a etapa de treinamento. Entretanto, antes de iniciá-la, alguns parâmetros devem ser definidos. Nesta seção, os principais parâmetros e dimensões do modelo, essenciais para garantir a eficiência do treinamento, serão abordados. É importante considerar que a definição dos melhores valores para esses parâme- tros depende diretamente das limitações de memória e processamento disponíveis. Reduzir os valores excessivamente pode comprometer a precisão do modelo, enquanto valores muito altos podem inviabilizar a execução do *software*.

O primeiro parâmetro a ser definido é a quantidade de lotes de treinamento ou validação processados pelo *software*. Após o processamento de todos os lotes, o treinamento ou a valida- ção é considerado concluído. Cada lote possui um tamanho específico, denominado tamanho do lote (do inglês, *batch size*). Na configuração atual, o valor é de 19, representando o número de um endereço IP e de colunas que vão ser analisadas. Assim, o modelo analisa os dados de um endereço IP por vez, identificando as intenções associadas a um todo de 19 endereços.

Os tamanhos de entrada para o codificador e o decodificador são configurados para corresponder ao tamanho do lote. Essa configuração garante que os dados sejam transmitidos eficientemente entre as cabeças do codificador e do decodificador. No modelo atual, cada valor de entrada corresponde a uma coluna de um endereço IP, sendo a última coluna responsável por indicar se a requisição é ou não um ataque. A previsão dessa coluna constitui o objetivo principal do modelo.

Outro parâmetro relevante é a taxa de aprendizado (do inglês, *learning rate*), a qual define o valor utilizado para ajustar os pesos do modelo durante o processo de aprendizado. Na configuração presente, a taxa de aprendizado é de 0,1, um valor relativamente elevado, que permite à IA identificar padrões de forma rápida, otimizando o uso de memória e tempo. Isso resulta em uma quantidade maior de lotes processados no período disponível.

O número de *tokens* usado atualmente é de 330000, resultado de vários testes para inferir qual valor geraria a maior acurácia. Este valor atende melhor à quantidade de caracteres do BD além de ser o maior valor que o *hardware* suporta processar ao longo do treinamento.

A quantidade de camadas por nó que o modelo terá é definido pelo número de dimensões do modelo. Atualmente é igual a 512. Nesse ponto, não só o fator *hardware* influencia, pois muitas dimensões por nó aumenta proporcionalmente o custo computacional, mas

também o fator de acerto. Ao definir muitas dimensões, a IA tende a ser muito branda, deixando passar vários IPs com intenções maliciosas. Quando é diminuído mais que isso, o *software* acerta ba- sicamente de forma aleatória. Mesmo com a taxa de acerto sendo de mais de 50%, muitos ataques maliciosos acabam passando, sendo o oposto do desejado.

O modelo tem um valor de dimensões tanto para a camada dos codificadores quanto para a dos decodificadores. Cada uma destas dimensões é composta por uma quantidade de cabeças. Estas cabeças correspondem a quantos codificadores e decodificadores tem em cada dimensão. Como o aumento de dimensões é custoso, no modelo proposto foi definida 8 dimen- sões, cada uma com 8 cabeças. Esses valores foram determinados a partir de quais se saíram melhores em relação ao tempo de treinamento e aos resultados.

Por fim, algumas das variáveis declaradas anteriormente, as quais podem ter os mesmos valores, foram reutilizadas, como o número de *tokens*, previamente declarado, tanto para o codificador quanto para o decodificador. Além disso, algumas outras variáveis *booleanas* que valem destacar são a de retorno do $loss$ do tgt (*target*) e o entrelaçamento de *tokens*, indicando que o que foi visto previamente influência no resultado atual. Afinal, os ataques DDoS são feitos em massa, então ter um contexto antes da resposta se mostrou eficaz na detecção.

Treinamento

O treinamento se inicia ao escolher o otimizador Estimativa de Momento Adaptativo, do inglês *Adaptive Moment Estimation* (Adam), para que os saltos de aprendizado inicialize grande e, em seguida, fique menor, evitando, assim, que os resultados obtidos se desviem do esperado. Após essa etapa, os valores retornados da função *cycle* serão os dados de treinamento de x ey, bem como o valor da máscara para acompanhar o tamanho dos tensores. O modelo é, então, ajustado para treinamento e, ao fazer isso, os pesos dentro de cada nó se modificarão.

Essa mudança é feita com base no quanto o modelo se distancia do esperado com suas respostas. Para que esse cálculo seja feito, é utilizado os valores de treino de x, como fonte (src), os valores de treino de y, como alvo (tgt), e, por fim, a máscara. Utilizando a equação matemática de perda (*loss*), como *Cross-Entropy Loss*, foi possível aferir resultados importante para que a função de *backward* percorra o modelo mudando seus pesos.

Validação no Treinamento

Antes de inicializar o treinamento, duas listas são declaradas para armazenar os resul- tados e, posteriormente, estimar as métricas descritas na seção 2.4. Uma das listas contém o valor *Label* (1 para benigno e 0 para ataque) e, a outra, o valor retornado do modelo (1 para benigno e 0 para ataque).

A validação se inicia com os valores de teste sendo passados para o modelo, não sem antes conferir se o rótulo é ou não benigno. Esses valores são armazenados em um vetor de valores positivos, o qual permitirá, futuramente, a geração da matriz de confusão. Ao passar os valores para o modelo, ele retornará um *array* com o tamanho do lote, sendo a última coluna (que seria a *Label*) o valor o qual a IA determinou para o tipo de ataque em questão (1 para benignos e 0 para malignos).

Para que o modelo não necessite inicializar o treinamento sempre do zero para verificar os resultados, os pesos de cada nó do modelo são salvos, permitindo, assim, a geração das mé- tricas e, ainda, treinar o mesmo modelo mais de uma vez. Com o modelo salvo, a mesma lógica da validação é utilizada durante o treino. Com o modelo treinado e declarado, as dimensões são ajustadas para ficarem conforme o modelo salvo. Em seguida, apenas valores de teste, adquiridos pela função *cycle*, são enviados para o mesmo.

Os valores a serem avaliados são encaminhados para o modelo já treinado, buscando somente a previsão assertiva. Durante esse processo, os valores de cada nó não serão altera- dos, o que fará com que o modelo não demore a validar os dados e não acabe comprometendo sua eficiência ao ser confrontado com os dados de treino.

UTILIZAÇÃO DA ARQUITETURA *TRANFORMERS*

O modelo da arquitetura *Transformers* utilizado é inspirado no modelo GPT, o qual pos- sui uma camada de codificador e outra de decodificador. Essas camadas ajudam com a detec- ção, pois se considera tanto o que o codificador quanto o que o decodificador produziram, dado que umas das camadas de atenções processa os dados das duas camadas.

O codificador possui 8 cabeças que recebem os lotes. Nas primeiras iterações, não há muito resultado, pois os valores gerados são basicamente aleatórios. Mas como a taxa de aprendizado é alta, os saltos são maiores levando ao retorno de valores esperados, 1 ou 0.

A primeira camada do codificador é a de atenção, aonde o modelo começa a entender o que deve fazer. Após essa camada, tem a de normalização, a qual garante que os dados não se distanciem do que se interessa. Vale ressaltar que os valores que não passaram pela atenção também são enviados para a normalização. Após isso, os valores e pesos são enviados adiante no modelo.

Após a última normalização do codificador, o modelo envia os pesos para o decodifi- cador. Nele, os pesos passarão por mais uma camada de atenção e normalização, além da camada de *feed-forward* e normalização novamente. Finalmente, os valores serão linearizados e sofrerão *softmax*, que atuará de maneira semelhante à normalização, mantendo os pesos em um gradiente de valor para que os valores futuros gerados não fiquem desbalanceados e o modelo aprenda da maneira correta.

Na maioria, o funcionamento do codificador e do decodificador é parecido, mas o se- gundo tem uma camada a mais de atenção, que seria a MHA mascarada, a qual serve para o modelo tentar prever o valor sem ver, futuramente, qual seria esse valor. No caso do modelo proposto, ele tentará identificar, com base no que ele aprendeu, se o valor do IP é de ataque ou não. Essa autorregressão é muito importante para o modelo e possibilita ter as respostas esperadas.

Com o progresso do treino, a lógica não muda muito. A diferença é que, com os pesos mudando e melhorando o modelo, as saídas dos codificadores se tornam as entradas dos de- codificadores até que um novo lote de dados seja enviado. Isso é realizado somente no treino, de modo que o modelo se depare com endereços de IP em suas mais variadas formas. Após o modelo entender o que deve entregar, os pesos dos nós vão se especificando, fazendo com que o modelo comece a reparar nos detalhes para determinar com mais precisão e exatidão quais são os valores de cada coluna de IP que apontam que o pacote é benigno ou malicioso.

Mais para o fim do treinamento, que leva de 6 a 8 horas, os pesos dos nós quase já não se alteram, e o *loss* se mantém constante e bem baixo, dada a pouca diferença entre o peso esperado e o escolhido pelo modelo. As respostas são, na maioria, assertivas.

RESULTADOS E DISCUSSÃO

A figura 10 mostra a porcentagem da taxa de acerto ao longo das versões nas quais o protótipo foi desenvolvido. Nela é possível observar que a taxa de acerto iniciou-se em 68% e, após inúmeras modificações nos parâmetros do modelo, a mesma se elevou a 99%.

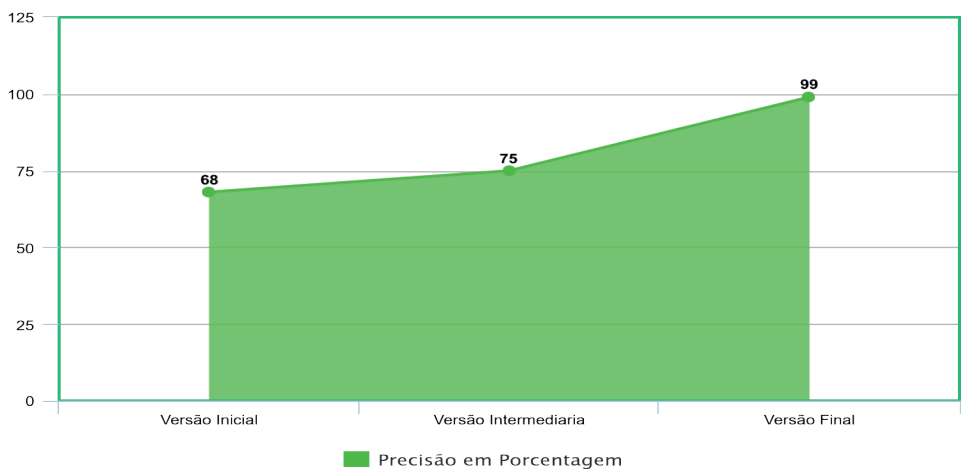

Figura 10 – Evolução da taxa de acerto da precisão em porcentagem por versão

Fonte: Autoria própria (2025).

O algoritmo, no início das versões, apresentava diversos problemas relacionados à me- mória RAM. As operações consumiam toda a memória disponível devido à quantidade de dados e dos tensores gerados. Quando estabilizado, após reduzir a quantidade de casas dos atribu- tos, dado que muitas das casas após a vírgula não agregavam tanto valor na detecção, o mo- delo começou a funcionar propriamente, com uma taxa de acerto de 68%. Entretanto, mesmo quando as primeiras predições começaram a fazer sentido, ainda havia muita confusão sobre quais eram os endereços de ataque e quais não eram, ou seja, o modelo estava adivinhando os resultados, não aprendendo.

As primeiras mudanças implementadas, principalmente na regulação de alguns parâ- metros (como o tamanho do lote ajustado para 1024, número de *tokens* para 120000, entre outros), permitiram à IA treinar de forma mais eficiente. Entretanto, o *software* ainda falhava em suas decisões, especialmente ao identificar endereços maliciosos que se assemelhavam aos benignos.

O salto de 68% para 75% foi alcançado ao alterar a manipulação dos tensores. Separados para teste e treino em proporção de 20% e 80% respectivamente, os tensores foram incorporados para serem usados em conjunto com a função *cuda()* do *tensorflow*. Entretanto, essa manipulação tornava os lotes para análise muito grandes, gerando uma sobrecarga na memória, sendo necessário adotar um número de lotes menor. No entanto, a quantidade de *to- kens* pôde ser aumentada de 120000 para 330000 removendo somente 3 casas decimais para evitar ao máximo a perca de informação.

Na versão intermediária do modelo, os parâmetros já estavam bem refinados, mas al- guns endereços maliciosos, que fugiam do padrão, ainda passavam despercebidos. Além disso, muitos dados continham somente lacunas (zeros) os quais foram excluídos, melhorando os re- sultados, mas criando outro problema: a IA aprendeu a identificar somente endereços de IP muito distintos, o que levou à estagnação dos resultados.

Para solucionar essa estagnação, foi adotada a estratégia de normalização dos dados, a qual permitiu à rede neural perceber pequenas diferenças entre os endereços. Com isso, o *software* conseguiu atingir mais de 95% de acerto. Após alguns ajustes finais nos parâmetros,como taxa de aprendizado de 0.1 e dimensão de 600 camadas de codificador e 8 de decodificador, a IA alcançou 99% de precisão. O Quadro 2 resume a evolução das versões indicando tanto a taxa de acerto como a característica do comportamento dos resultados que ele obtivera

Versão	Taxa de Acerto (%)	Característica
Versão Inicial	68	Previsões quase aleatórias com pouco aprendizado.
Versão Intermediária	75	Primeiras previsões confiáveis, mas falhando na detecção de endereços mais difíceis.
Versão Final	99	Previsões bem feitas detectando grande parte dos ataques.

Quadro 2 – Evolução da Taxa de Acerto e Características do Modelo

Os resultados obtidos na versão final são produtos da análise de 3998 endereços IP obtidos do mesmo BD do treino. Essa validação levou 10 horas para ser realizada gerando uma precisão de 99%, um recall de 97%, uma acurácia e*F1 score* de 98%. Estes valores mostram uma eficácia considerável do modelo em detectar IPs provenientes de um ataque.

A figura 11 apresenta a matriz de confusão. Pode-se observar que, dentre os 3998 ende- reços IP, o modelo acertou 3971 (3415 ataques previstos pelo modelo realmente como ataques, e 556 não ataques previstos corretamente pelo modelo como não ataques). Entretanto, o mo- delo classificou 27 endereços como ataques mesmo eles não sendo. Deste modo, nota-se que o modelo não teve erros ao detectar os ataques, dado que não deixou nenhum endereço malcioso ser classificado como benigno.

A figura 12 representa a mediana do tempo de detecção, a qual corresponde à 0,93 segundos. Este valor significa que, para cada endereço de IP analisado, o modelo demorou em torno de 1 segundo para ser detectado.

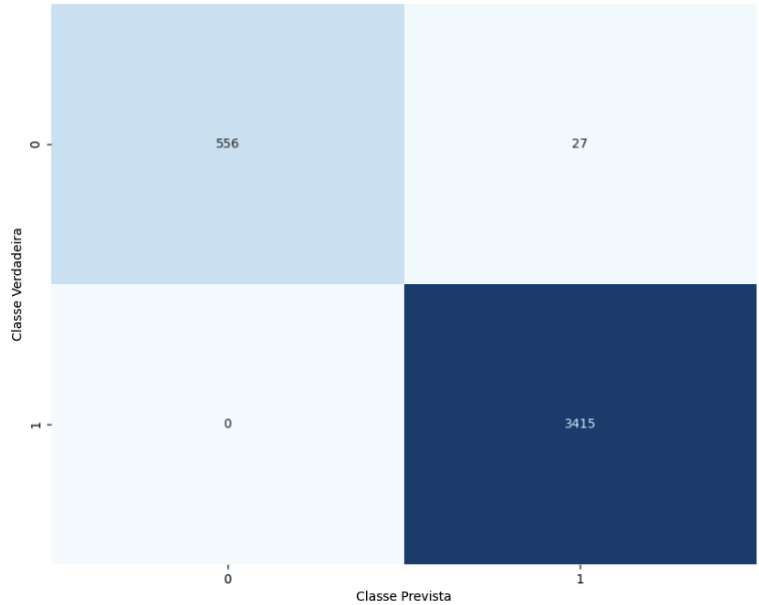

Figura 11 – Matriz de confusão resultante da validação do modelo proposto

Fonte: Autoria própria (2025).

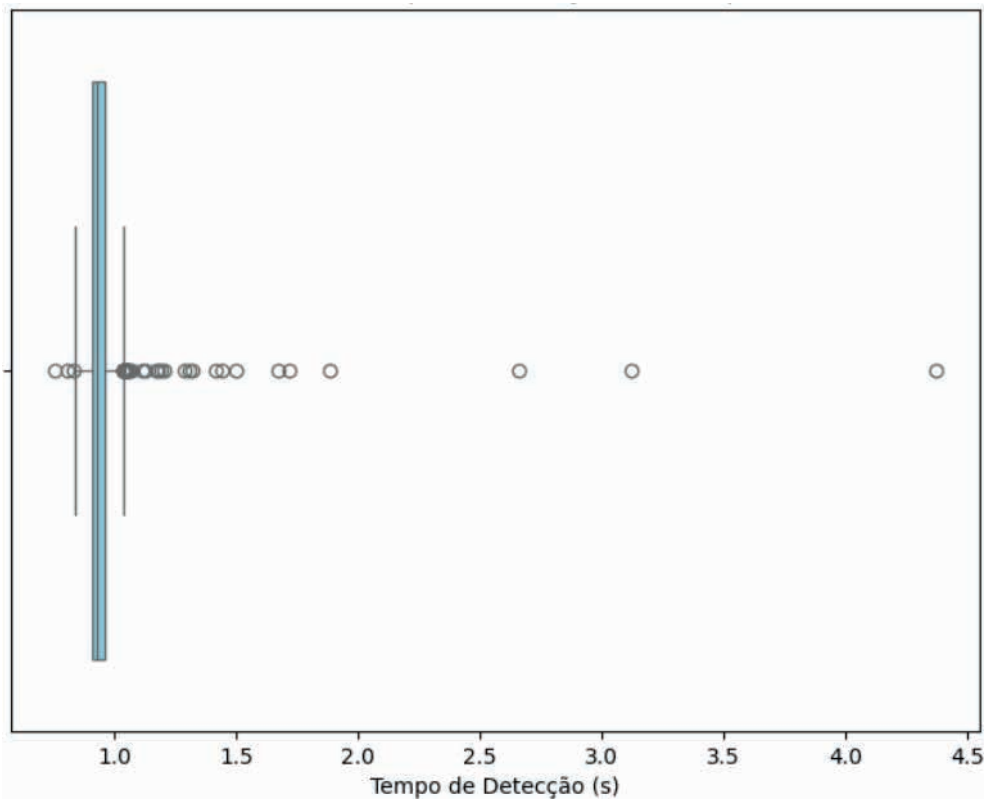

Figura 12 – Tempo gasto na detecção de cada ataque pelo modelo proposto

Fonte: Autoria própria (2025).

CONCLUSÃO

O emprego da arquitetura *Transformers* na detecção de ataques DDoS e DoS, como demonstrado nos resultados deste trabalho e em outros trabalhos relacionados, se mostra efi- ciente. Entretanto, confeccionar essas IAs requer um alto grau de dificuldade, tendo como um dos maiores desafios a capacidade de processar os dados computacionalmente. Tal desafio foi evidenciado neste trabalho quando a memória RAM foi totalmente ocupada durante o pro- cessamento dos dados, sendo solucionado após ajustes nos parâmetros e na seleção de lotes menores.

Mesmo após o ajuste para evitar o estouro de *buffer*, o modelo proposto ainda possui limitações em relação ao tempo e ao custo computacional. Durante a fase de treino, há uma demanda significativa de *hardware*, além disso, o tempo para o processamento dos dados é superior ao ideal. Isso demostra que alguns parâmetros ainda podem ser melhor ajustados e otimizados de modo que os processos possam ser melhor divididos exigindo menos do *hard- ware*.

Ao filtrar os dados para o treino, o modelo proposto obteve um melhor desempenho no treinamento, melhorando, assim, sua taxa de acerto. Entretanto, mesmo que um *dataset* modi- ficado para conter somente os dados selecionados, limpos e rotulados de ataques ao *host*, seja extremamente eficaz, o mesmo pode causar alguns problemas quando a arquitetura proposta for aplicada para detectar os ataques em ambientes reais. Isso ocorre devido à existência de variáveis e dados que somente são possíveis com tráfego real.

Por fim, após a adequação do modelo aos diferentes desafios encontrados no decorrer do desenvolvimento, o modelo proposto está operando normalmente com uma taxa de preci- são de 99%. Essa taxa permite utilizá-lo para assegurar a segurança dos dados pessoais dos usuários, proporcionando tranquilidade e privacidade a eles. Para o futuro, espera-se conseguir adaptá-lo o suficiente para realizar previsões em tempo real no tráfego de um servidor.

REFERÊNCIAS

ALFATEMI, A. *et al.* **Advancing DDoS Attack Detection: A Synergistic Approach Using Deep Residual Neural Networks and Synthetic Oversampling**. arXiv, 2024. Disponível em: https://arxiv.org/abs/2401.03116.

ALMEIDA, M. **Pandas: o que é, para que serve e como instalar**. Alura, 2023. Disponível em: https://www.alura.com.br/artigos/pandas-o-que-e-para-que-serve-como-instalar. Acesso em: 7 dez 2023.

ALRAHMANI, Z. A.; ELLEITHY, K. Ddos attack forecasting using transformers. *In*: **2023 IEEE Intl Conf onyberSciTech)**. IEEE, 2023. p. 0911–0915. Disponível em: http://dx.doi.org/10.1109/DASC/PiCom/CBDCom/ Cy59711.2023.10361388.

ANTONAKAKIS, M. *et al.* Understanding the mirai botnet. *In*: **26th USENIX Security Symposium (USENIX Security 17)**. Vancouver, BC: USENIX Association, 2017. p. 1093–1110. ISBN 978-1-931971-40-9. Disponível em: https://www.usenix.org/conference/usenixsecurity17/ technical-sessions/presentation/antonakakis.

ARORA, M. *et al.* **Como a Cloudflare mitigou automaticamente um ataque DDoS recorde mundial de 3,8 Tbps**. Blog da Cloudflare, 2024. Disponível em: https://blog.cloudflare.com/ pt-br/how-cloudflare-auto-mitigated-world-record-3-8-tbps-ddos-attack/. Acesso em: 24 fev 2025.

ASIMOV, I. **I, Robot**. United States of America: Gnome Press, 1950.

AWAN, A. A. **O que é tokenização?** DataCamp, 2024. Disponível em: https://www.datacamp. com/pt/blog/what-is-tokenization. Acesso em: 24 fev 2025.

BACH, S. L. **CONTRIBUIÇÃO DO HACKER PARA O DESENVOLVIMENTO TECNOLÓGICODA INFORMÁTICA**. 2001. 135 p. Dissertação (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Santa Catarina (UFSC), Florianópolis - Santa Catarina, 2001. Disponível em: https://repositorio.ufsc.br/xmlui/handle/123456789/82176. Acesso em: 20 fev 2025.

BARBOSA, G. *et al.* Segurança em redes 5g: Oportunidades e desafios em detecção de anomalias e predição de tráfego baseadas em aprendizado de máquina. *In*: **Minicursos do XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**. SBC, 2021. p. 145–189. Disponível em: http://dx.doi.org/10.5753/sbc.7165.8.4. Acesso em: 25 fev 2025.

BROWN, J. P. D. **Chattronics: utilizando GPTs para auxílio a projeto de sistemas de aquisição de sinais**. 2024. 150 f. Monografia (TCC) — Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, 2024. Disponível em: http://hdl.handle.net/10183/274294. Acesso em: 11 out 2024.

CASELI, H. d. M.; NUNES, M. d. G. V. **Processamento de linguagem natural: conceitos, técnicas e aplicações em português**. São Carlos, SP: BPLN, 2023. Disponível em: https://brasileiraspln.com/livro-pln/. Acesso em: 24 fev 2025.

CHAGAS, D. A. de M. **Detecção de Ataques de Negação de Serviço em SGBDs a Partir de Logs Internos Usando Abordagens Supervisionada e Não Supervisionada**. 2024. 68 p. Dissertação (Mestrado) — Universidade de Brasília, Brasília, DF, 2024. Disponível em: http://icts.unb.br/jspui/bitstream/10482/48503/1/2024_DaniloAndersonDeMouraChagas_ DISSERT.pdf. Acesso em: 11 nov 2024.

CHAVAN, N. *et al.* Ddos attack detection and botnet prevention using machine learning. *In*: **2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)**. IEEE, 2022. p. 1159–1163. Disponível em: http://dx.doi.org/10.1109/ICACCS54159. 2022.9785247.

COMER, D. E. **Redes de Computadores e Internet**. 6º. ed. São Paulo, SP: Bookman, 2016. ISBN: 978-8582603727.

COSTA, W. L.; PORTELA, A. L. de C.; GOMES, R. L. Análise de características do tráfego de rede para detecção de ataques ddos em ambientes iot. *In*: **Anais do XII Computer on the Beach - COTB'21**. Universidade do Vale do Itajaí, 2021. (COTB, v. 12), p. 217–224. Disponível em: https://periodicos.univali.br/index.php/acotb/article/view/17404.

DONG, S.; ABBAS, K.; JAIN, R. A survey on distributed denial of service (ddos) attacks in sdn and cloud computing environments. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 7, p. 80813–80828, 2019. ISSN 2169-3536. Disponível em: http://dx.doi.org/10.1109/ACCESS.2019.2922196.

FERNANDES, N. O. C. **Segurança da Informação**. 1a edição. ed. ProEdu, 2018. 104 p. Disponível em: http://proedu.rnp.br/handle/123456789/1538. Acesso em: 12 out 2023.

FILHO, M. **Precisão, Recall e F1 Score em Machine Learning**. Mario Filho - Machine Learning, 2023. Disponível em: https://mariofilho.com/precisao-recall-e-f1-score-em-machine-learning/. Acesso em: 25 fev 2025.

HAN, K. *et al.* A survey on vision transformer. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Institute of Electrical and Electronics Engineers (IEEE), v. 45, n. 1, p. 87–110, jan. 2023. Disponível em: http://dx.doi.org/10.1109/tpami.2022.3152247.

K, A.; JOHN, A. A deep learning based intrusion detection system using transformers.**SSRN Electronic Journal**, Elsevier BV, 2022. ISSN 1556-5068. Disponível em: http://dx.doi.org/10.2139/ssrn.4294593.

KOSTAS, K. **Anomaly Detection in Networks Using Machine Learning**. 2018. 70 p. Dissertação (Dissertação) — Computer Science and Electronic Engineering - University of Essex, Colchester, Inglaterra, 2018. Disponível em: https://www.researchgate.net/publication/ 328512658_Anomaly_ Detection_in_Networks_Using_Machine_Learning. Acesso em: 11 nov 2023.

KRIGER, D. **O que é Python, para que serve e por que aprender**. Kenzie, 2022. Disponível em: https://blog-next.kenzie.com.br/o-que-e-python. Acesso em: 18 fev 2024.

KUROSE, J. F.; ROSS, K. W. **Computer networking: a top-down approach**. 7. ed. Amherst, MA: Pearson, 2016. 864 p. ISBN 978-0133594140.

LIMA, A. C. de M. **Um método de dois estágios para detecção de pólipos em imagens de colonoscopia usando aprendizado profundo**. 2023. 156 f. Tese (Doutorado) — Universidade Federal do Maranhão - UFMA, São Luis - Maranhão, 2023. Disponível em: https://tedebc.ufma.br/jspui/handle/tede/5052. Acesso em: 11 out 2024.

LIMA, D. L. S. **Classificação de Imagens de Exames de Endoscopia por Cápsula Utilizando Transformers**. 2023. 57 p. Dissertação (Mestrado) — Universidade Federal do Maranhão - UFMA, São Luís - Maranhão, 2023. Disponível em: https://tedebc.ufma.br/jspui/handle/tede/ 4635. Acesso em: 11 set 2024.

LONG, Z. *et al.* A transformer-based network intrusion detection approach for cloud security. **Journal of Cloud Computing**, Springer Science and Business Media LLC, v. 13, n. 1, jan. 2024. ISSN 2192-113X. Disponível em: http://dx.doi.org/10.1186/s13677-023-00574-9.

MASTERS, D.; LUSCHI, C. **Revisiting Small Batch Training for Deep Neural Networks**. arXiv, 2018. Disponível em: https://arxiv.org/abs/1804.07612. Acesso em: 25 fev 2025.

MENDONÇA, G. *et al.* Detecção de ataques ddos usando correlação espaço-temporal bayesiana. *In*: **Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2022)**. Sociedade Brasileira de Computação (SBC), 2022. (SBRC 2022),p. 224–237. Disponível em: http://dx.doi.org/10.5753/sbrc.2022.222296.

MPF. **Crimes Cibernéticos**. [*S.l.*], 2018. Disponível em: https://memorial.mpf.mp.br/nacional/ vitrine-virtual/publicacoes/crimes-ciberneticos-coletanea-de-artigos. Acesso em: 18 out 2023.

NETO, F. **Modelo TCP/IP - 4 ou 5 camadas?** Redes Brasil, 2018. Disponível em: https://www.redesbrasil.com/modelo-tcp-ip-4-ou-5-camadasp/. Acesso em: 11 nov 2023.

OLIVEIRA, J. S. de *et al.* **As Tecnologias da Informação e Comunicação na Gestão Administrativa e Operacional da Segurança Pública**. São Paulo: Blucher Open Access, 2016. v. 1. 43-54 p.

PATEL, M. *et al.* Ddos attack detection model using machine learning algorithm in next generation firewall. **Procedia Computer Science**, Elsevier BV, v. 233, p. 175–183, 2024. ISSN 1877-0509. Disponível em: http://dx.doi.org/10.1016/j.procs.2024.03.207.

PONTES, L. B. L. **Investigação de modelos neurais baseados na arquitetura Transformer para sumarização automática de código-fonte**. 2022. 71 p. Dissertação (Dissertação (Mestrado em Computação Aplicada)) — Instituto Federal do Espirito Santo (IFES), Serra - Espírito Santo, 2022. Disponível em: https://repositorio.ifes.edu.br/handle/123456789/2993.Acesso em: 11 nov 2023.

RODRIGUES, M. **MafiaBoy: hacker símbolo dos ataques DDoS em 2000 fala sobre segurança**. TecMundo, 2017. Disponível em: https://www.tecmundo.com.br/seguranca/ 121123-mafiaboy-hacker-simbolo-ataques-ddos-2000-fala-seguranca.htm. Acesso em: 24 fev 2025.

SANA, L. *et al.* Securing the iot cyber environment: Enhancing intrusion anomaly detection with vision transformers. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 12, p. 82443–82468, 2024. ISSN 2169-3536. Disponível em: http://dx.doi.org/10.1109/ACCESS.2024.3404778.

SANTOS, S. R. dos. **O uso de aprendizado profundo para predição de diagnósticos de doenças cardiovasculares**. 2023. 533 f. Monografia (TCC) — Universidade Federal do Ceará - UFC, Quixadá, Ceará, 2023. Disponível em: http://repositorio.ufc.br/handle/riufc/75923. Acesso em: 11 out 2024.

SOLHA, L.; TEIXEIRA, R.; PICCOLINI, J. **Tudo que você precisa saber sobre os ataques DDoS**. 2000.

STALLINGS, W. **Criptografia E Segurança De Redes: PRINCÍPIOS E PRÁTICAS**. 6º. ed. [*S.l.*]: Pearson Universidades, 2014. 560 p. ISBN 9788543005898.

TANDON, R. **A Survey of Distributed Denial of Service Attacks and Defenses**. arXiv, 2020. Disponível em: https://arxiv.org/abs/2008.01345. Acesso em: 25 fev 2025.

TANENBAUM, A. S. **Redes de Computadores**. 6. ed. São Paulo, SP: Pearson Prentice Hall, 2021. 593 p.

TAVARES, A. C. J. **Detecção de Ataques DDoS Através da Lógica Paraconsistente Anotada**. 2021. 120 f. Monografia (TCC) — Universidade Tecnológica Federal do Paraná, Santa Helena, Paraná, 2021. Disponível em: http://repositorio.utfpr.edu.br/jspui/handle/1/26462. Acesso em: 12 out 2023.

TELECOMUNICAÇÕES, W. **Quais prejuízos sua empresa tem com ataques DDoS e como a WCS previne para que não isso aconteça**. 2023. Disponível em: https://pt.linkedin.com/ pulse/quais-preju%C3%ADzos-sua-empresa-tem-com-ataques-ddos-e-como/. Acesso em: 01 nov 2023.

TOKUDA, N. H. O. **Análise de sentimento por meio de deep learning aplicada à mineração de argumentos**. 2021. 19 f. Monografia (TCC) — Universidade Presbiteriana Mackenzie, Higienópolis, São Paulo, 2021. Disponível em: https://dspace.mackenzie.br/items/ f4acbfc0-5c03-49e7-8a1e-89c7ece199fe.

VASWANI, A. *et al.* Attention is all you need. *In*: GUYON, I. *et al.* (Ed.). **Advan- ces 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.

YOACHIMIK, O.; PACHECO, J. **DDoS threat report for 2023 Q4**. Cloud Flare, 2024. Disponível em: https://blog.cloudflare.com/ddos-threat-report-2023-q4/. Acesso em: 11 nov 2023.

ZANOL, M. V. **Integração de ferramentas contra ataques DDoS e malware**. 2018. 151 f. Monografia (TCC) — Universidade de Caxias do Sul, Caxias do Sul - RS, 2018. Disponível em: https://repositorio.ucs.br/11338/3930.

ZANOTTO, L.; FERREIRA, A.; MATSUMOTO, M. Arquitetura e programação de gpu nvidia. **ProQuest Number: INFORMATION TO ALL USERS**, 2015.