Universidade Estadual de Londrina

GUSTAVO HIDEYUKI KITAMURA NISHIKAWA

DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES UTILIZANDO ARQUITETURA HÍBRIDA BASEADA EM TRANSFORMADOR E AUTOCODIFICADOR VARIACIONAL

LONDRINA

2025

GUSTAVO HIDEYUKI KITAMURA NISHIKAWA

DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES UTILIZANDO ARQUITETURA HÍBRIDA BASEADA EM TRANSFORMADOR E AUTOCODIFICADOR VARIACIONAL

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Mario Lemes Proença Jr.

LONDRINA
2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

N724d Nishikawa, Gustavo Hideyuki Kitamura.

Detecção de anomalias em redes de computadores utilizando arquitetura híbrida baseada em transformador e autocodificador variacional / Gustavo Hideyuki Kitamura Nishikawa. - Londrina, 2025.

82 f. : il.

Orientador: Mario Lemes Proença Jr..

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Graduação em Ciência da Computação, 2025.

Inclui bibliografia.

1. Segurança de redes - TCC. 2. Transformador - TCC. 3. Autocodificador Variacional - TCC. 4. Aprendizado profundo - TCC. I. Proença Jr., Mario Lemes. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Graduação em Ciência da Computação. III. Título.

CDU 519

GUSTAVO HIDEYUKI KITAMURA NISHIKAWA

DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES UTILIZANDO ARQUITETURA HÍBRIDA BASEADA EM TRANSFORMADOR E AUTOCODIFICADOR VARIACIONAL

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Mario Lemes Proença Jr.
Universidade Estadual de Londrina

Prof. Dr. Elieser Botelho Manhas Jr.
Universidade Estadual de Londrina

Vinícius Ferreira Schiavon
Universidade Estadual de Londrina

Londrina, 25 de novembro de 2025.

Dedico este trabalho à minha família, que sempre se preocupou comigo e me forneceu suporte para continuar.

AGRADECIMENTOS

Aos meus pais, por todo o esforço e sacrifício que sempre dedicaram para me apoiar em cada etapa da minha vida. Ao meu irmão e meu gato, pela presença constante.

Aos meus amigos, pela análise crítica figurativa.

Ao Professor Dr. Mario Lemes Proença Jr., por toda a orientação, paciência, ensinamentos, e pela oportunidade de crescimento acadêmico e pessoal proporcionados ao longo desta jornada.

Aos membros do grupo Orion, por todo auxílio no meu aprendizado e por sempre estarem dispostos a ajudar.

Aos professores do curso Ciência da Computação, pelo apoio no meu desenvolvimento profissional.

Aos meus colegas do curso, por compartilharem esta jornada comigo.

Aos servidores da UEL, pelo esforço diário que possibilita as atividades da universidade.

"O que eu não posso criar, eu não entendo."

Richard Feynman

NISHIKAWA, G. H. K.. **Detecção de Anomalias em Redes de Computadores Utilizando Arquitetura Híbrida Baseada em Transformador e Autocodificador Variacional**. 2025. 81f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2025.

RESUMO

A importância central das redes de computadores na sociedade moderna impulsiona a necessidade de sistemas de proteção robustos, como os Sistemas de Detecção de Intrusão em Redes. O Aprendizado Profundo é uma ferramenta promissora neste contexto pela sua capacidade identificar e aprender padrões. O presente trabalho realiza um estudo sobre os fundamentos do Aprendizado Profundo, com foco nas arquiteturas do Autocodificador Variacional e do Transformador. É implementado um modelo não supervisionado que integra ambas as arquiteturas estudadas. O modelo é testado em um conjunto de dados sintético de tráfego de rede, e a avaliação de sua eficácia foi conduzida por meio das métricas de Precisão, Revocação, *F1-score*, Coeficiente de Correlação de Matthews e Área Sob a Curva Característica de Operação do Receptor. O modelo demonstrou robustez, obtendo valores superiores a $0,97$ em todas as métricas testadas.

Palavras-chave: Segurança de redes. Detecção de anomalias. Detecção de intrusões. Aprendizado Profundo. Autocodificador Variacional. Transformador.

NISHIKAWA, G. H. K.. **Anomaly Detection in Computer Networks using a Hybrid Architecture based on Transformer and Variational Autoencoder**. 2025. 81p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2025.

ABSTRACT

The central importance of computer networks in modern society drives the need for robust protection systems, such as Network Intrusion Detection Systems. Deep Learning is a promising tool in this context due to its ability to identify and learn patterns. This work conducts a study on the fundamentals of Deep Learning, focusing on the Variational Autoencoder and Transformer architectures. An unsupervised model that integrates both studied architectures is implemented. The model is tested on a synthetic network traffic dataset. The model's effectiveness was evaluated using Precision, Recall, *F1-score*, Matthews Correlation Coefficient, and Area Under the Receiver Operating Characteristic Curve metrics. The model demonstrated robustness, achieving values greater than 0.97 for all tested metrics.

Keywords: Network security. Anomaly detection. Intrusion detection. Deep Learning. Variational Autoencoder. Transformer.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura das Redes Definidas por *Software*. Fonte: próprio autor. . . 20
Figura 2 – Rede Neural Artificial. Fonte: próprio autor. 24
Figura 3 – Aproximação da função seno com ANN. Fonte: adaptado de Prince [1]. 27
Figura 4 – Matriz de Confusão. Fonte: próprio autor. 32
Figura 5 – Arquitetura do Autocodificador. Fonte: próprio autor. 34
Figura 6 – Arquitetura do Autocodificador Variacional em alto nível. Fonte: próprio autor. 35
Figura 7 – Arquitetura do Autocodificador Variacional completa. Fonte: próprio autor. 40
Figura 8 – Arquitetura do Transformador. Fonte: adaptado de Vaswani *et al.* [2]. . 43
Figura 9 – Características do primeiro dia. Fonte: próprio autor. 59
Figura 10 – Características do segundo dia. Fonte: próprio autor. 59
Figura 11 – Características do terceiro dia. Fonte: próprio autor. 60
Figura 12 – Arquitetura do Sistema Proposto. Fonte: próprio autor. 61
Figura 13 – AnomT-VAE. Fonte: próprio autor. 63
Figura 14 – Matrizes de confusão para as quatro configurações avaliadas. Fonte: próprio autor. 68
Figura 15 – Curvas ROC para as funções de perda MSE e KLD. Fonte: próprio autor. 69

LISTA DE TABELAS

Tabela 1 – Principais Bibliotecas. Fonte: próprio autor. 66
Tabela 2 – Hiperparâmetros do Modelo e Treinamento. Fonte: próprio autor. . . . 66
Tabela 3 – Resultados das métricas de desempenho separadas por abordagem. Fonte: próprio autor. 67
Tabela 4 – Resultados a Nível de Fluxo do Sistema (Detecção + Mitigação). Fonte: próprio autor. 70
Tabela 5 – Resultados da Mitigação. Fonte: próprio autor. 70

LISTA DE ABREVIATURAS E SIGLAS

AE	*Autoencoder*
ANN	*Artificial Neural Network*
AUROC	*Area Under the Receiver Operating Characteristic Curve*
BCE	*Binary Cross Entropy*
BERT	*Bidirectional Encoder Representations from Transformers*
`[CLS]`	*Classification token*
ConVAE	*Convolutional Variational Autoencoder*
DDoS	*Distributed Denial of Service*
DKL	Divergência de Kullback-Leibler
DL	*Deep Learning*
ELBO	*Evidence Lower Bound*
FFN	*Feed-Forward Network*
FN	Falso Negativo
FP	Falso Positivo
FPR	*False Positive Rate*
GPT	*Generative Pre-trained Transformer*
i.i.d.	independentes e identicamente distribuídos
K	*Key*
LDM	*Latent Diffusion Model*
LLM	Large Language Model
MCC	*Matthews Correlation Coefficient*
MHA	*Multi-Head Attention*
MLE	*Maximum Likelihood Estimation*
MLP	*Multilayer Perceptron*

MSA-CBAM	*Multi-Scale Adaptive Convolutional Block Attention*
MSE	*Mean Square Error*
MVAE	*Multidistributed Variational Autoencoder*
NIDS	*Network Intrusion Detection System*
NLL	*Negative Log Likelihood*
PLN	Processamento de Linguagem Natural
Q	*Query*
ReLU	*Rectified Linear Units*
RNN	*Recurrent Neural Network*
ROC	*Receiver Operating Characteristic*
RTIDS	*Robust Transformer-based Intrusion Detection System*
RUIDS	*Robust Unsupervised Intrusion Detection System*
SDN	*Software Defined Network*
SGD	*Stochastic Gradient Descent*
T-NAE	*Transformer-Based Network Traffic Autoencoder*
T-VAE	*Transformer-Based Variational Autoencoder*
TPR	*True Positive Rate*
UEL	Universidade Estadual de Londrina
V	*Value*
VAE	*Variational Autoencoder*
ViT	*Vision Transformer*
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

LISTA DE SÍMBOLOS

ϕ	Conjunto de parâmetros aprendíveis (Decodificador VAE e redes neurais)
θ	Conjunto de parâmetros aprendíveis (Codificador VAE)
η	Taxa de aprendizado
μ	Média
σ	Desvio-padrão
σ^2	Variância
ϵ	Amostra de ruído
α	Peso de atenção
τ	Limiar de detecção
∇	Gradiente
∂	Derivada parcial
$\sum$	Somatório
$\prod$	Produtório
$\subset$	Contido em
$\odot$	Multiplicação elemento a elemento
$arg\ min$	Argumento que minimiza a função
$arg\ max$	Argumento que maximiza a função
$softmax$	Função Softmax
$LayerNorm$	Normalização de Camada
$L(\phi)$	Função de perda
$E_{x \sim P}$	Valor esperado sobre a distribuição P
$D_{KL}(P\|\|Q)$	Divergência de Kullback-Leibler
P_{data}	Distribuição de probabilidade real dos dados

P_{model}	Distribuição de probabilidade do modelo
$Q(z\|x)$	Distribuição variacional (Codificador VAE)
$P(x\|z)$	Modelo generativo (Decodificador VAE)
$\mathcal{N}(\mu, \sigma)$	Distribuição Normal (Gaussiana)
$H(X)$	Entropia de Shannon
x	Vetor de entrada
y	Vetor de saída
$\hat{y}$	Predição do modelo
z	Vetor (resultado linear ou vetor latente)
W	Matriz de pesos
b	Vetor de viés
a	Vetor de ativação
f_ϕ	Função computada pela rede neural
S	Conjunto de dados
D	Universo de dados
X	Matriz de dados de entrada
X^*	Dados reconstruídos
I	Matriz identidade
J	Dimensão do espaço latente
R	Conjunto dos números reais
T	Transposta de matriz/vetor
q	Vetor de Consulta
k	Vetor de Chave
v	Vetor de Valor
Q	Matriz de Consultas
K	Matriz de Chaves

V	Matriz de Valores
d_{model}	Dimensão de embedding
h	Número de cabeças de atenção
pos	Posição em uma sequência

SUMÁRIO

1 INTRODUÇÃO . . . 18

2 FUNDAMENTAÇÃO TEÓRICA . . . 20
2.1 Redes Definidas por *Software* . . . 20
2.2 Anomalias em Redes de Computadores . . . 21
2.3 Sistemas de Detecção de Intrusão em Redes de Computadores 22
2.4 Aprendizado Profundo . . . 23
2.4.1 Redes Neurais . . . 23
2.4.2 Aprendizado com Dados . . . 28
2.4.3 Funções de Perda . . . 28
2.4.4 Otimização . . . 30
2.4.5 Métricas de Avaliação . . . 31

3 AUTOCODIFICADORES VARIACIONAIS . . . 34
3.1 Limite Inferior da Evidência . . . 35
3.2 Implementação Computacional . . . 38
3.3 Aplicações Do Autocodificador Variacional . . . 40
3.3.1 Geração de Dados . . . 40
3.3.2 Componente em Modelos de Difusão Latente . . . 41
3.3.3 Detecção de Anomalias . . . 41

4 TRANSFORMADORES . . . 42
4.1 Mecanismos do Transformador . . . 42
4.1.1 Camada de Incorporação . . . 42
4.1.2 Codificação Posicional . . . 44
4.1.3 Atenção . . . 44
4.1.3.1 Atenção por Produto Escalar com Escala . . . 46
4.1.3.2 Atenção Multi-Cabeça . . . 46
4.1.4 Rede de Alimentação Direta . . . 47
4.1.5 Adição e Normalização . . . 47
4.2 Montagem da Arquitetura . . . 48
4.2.1 Codificador do Transformador . . . 48
4.2.2 O Bloco Decodificador . . . 49
4.2.3 Camada de Saída Final . . . 51
4.3 Aplicações do Transformador . . . 51
4.3.1 Processamento de Linguagem Natural . . . 51

4.3.2 Visão Computacional 52
4.3.3 Detecção de Anomalias em Séries Temporais 52

5 TRABALHOS RELACIONADOS 53

6 ESTUDO DE CASO 56
6.1 Conjunto de Dados 56
6.1.1 Análise Exploratória de Dados 58
6.2 Sistema de Detecção e Mitigação de Intrusão de Redes 61
6.2.1 Coletor de Tráfego e Pré-processador de Dados 61
6.2.2 Detector de Anomalias 62
6.2.2.1 AnomT-VAE 62
6.2.2.2 Pontuação e Limiar 64
6.2.3 Mitigador de Anomalias 65
6.3 Experimentos e Resultados 66
6.3.1 Detecção de Anomalias 66
6.3.2 Mitigação de Anomalias 70
6.4 Discussão 70

7 CONCLUSÃO 72

REFERÊNCIAS 73

1 INTRODUÇÃO

As redes de computadores representam um dos elementos mais importantes na sociedade contemporânea, permitindo a comunicação em tempo real e o compartilhamento de recursos digitais a longas distâncias [3]. A evolução destas redes transformou o modo como indivíduos e organizações interagem, tornando-se não apenas um meio de entretenimento, mas uma infraestrutura crítica para setores como saúde, educação, finanças e segurança nacional [4]. Com todos esses benefícios, empresas, governos, instituições financeiras e outros serviços essenciais passaram a depender dessas redes para suas operações diárias [4]. Esta dependência transformou as redes de computadores em recursos fundamentais, cuja falha pode gerar grandes prejuízos financeiros, comprometer serviços essenciais e até mesmo ameaçar a segurança nacional [5, 6].

Os dados que trafegam por essas redes comumente possuem valor para as organizações envolvidas. Essa situação criou um cenário favorável para o surgimento de ameaças cibernéticas, que empregam técnicas para descobrir e explorar vulnerabilidades das redes. Esses ataques têm como objetivo desde o roubo de dados com valor até a interrupção de serviços de organizações [6]. Diante desse cenário, torna-se necessário o desenvolvimento de medidas para combater o crescente problema de segurança cibernética. Métodos tradicionais de detecção de intrusão baseados em assinaturas, embora eficazes contra ameaças conhecidas [7], não conseguem identificar ataques do tipo *zero-day* que exploram vulnerabilidades desconhecidas ou variações específicas de ataques existentes [5]. A necessidade de sistemas mais adaptativos e inteligentes tornou-se imprescindível para proteger adequadamente as infraestruturas de rede modernas.

A revolução digital tem impulsionado o crescimento no volume de tráfego de rede que, segundo estimativas, ultrapassará dezenas de zettabytes nos próximos anos [8]. O crescimento no volume de tráfego, combinado com a crescente sofisticação dos ataques cibernéticos [9, 10], representa um desafio significativo para os sistemas de segurança convencionais. Diante das limitações das abordagens tradicionais para lidar com essa complexidade, técnicas de Aprendizado Profundo surgem como ferramentas promissoras, capazes de identificar padrões que seriam imperceptíveis aos métodos tradicionais [11, 12, 13]. As arquiteturas de Aprendizado Profundo permitem o reconhecimento e a modelagem de padrões em dados de altas dimensões, viabilizando a detecção de anomalias em grandes volumes de dados de rede e adaptando-se continuamente aos novos padrões de tráfego e ameaças [14, 15, 16].

Entre as arquiteturas de Aprendizado Profundo, o Transformador [2] tem despertado uma notoriedade significativa na área de detecção de intrusões em redes [17]. Esta arquitetura se destaca por seu mecanismo de autoatenção, que captura eficientemente

relações entre características de tráfego de rede temporalmente distantes, superando limitações encontradas em arquiteturas como as Redes Neurais Recorrentes [18]. Algumas dessas limitações incluem a dificuldade em reter informações de longo prazo e o custo computacional elevado para processamento em sequência [19]. Os Autocodificadores Variacionais [20] se destacam por aproximar a distribuição probabilística dos dados, conferindo um caráter não supervisionado ao modelo [21]. Essa característica permite que os sistemas detectem intrusões sem depender de conjuntos de dados rotulados. Isso viabiliza a detecção de ataques *zero-day* em cenários reais, onde a anotação manual e contínua representa um desafio significativo [22].

A integração de Transformador e Autocodificadores Variacionais emerge como uma abordagem inovadora para a detecção de intrusão em redes [23]. Ao combinar a capacidade dos Transformadores de capturar relações de longo alcance com a habilidade dos Autocodificadores Variacionais de aproximar distribuições probabilísticas, espera-se obter um Sistema de Detecção de Intrusão em Rede eficaz. Este trabalho propõe realizar um estudo dos modelos Transformador e Autocodificador Variacional e implementar uma arquitetura híbrida que integra as duas abordagens para a detecção de intrusão em redes.

Este trabalho está estruturado da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica, abordando conceitos de Redes Definidas por *Software*, anomalias em redes, Sistemas de Detecção de Intrusão e os fundamentos do Aprendizado Profundo. Em seguida, o Capítulo 3 foca em detalhar a arquitetura e os princípios dos Autocodificadores Variacionais, enquanto o Capítulo 4 se aprofunda nos mecanismos dos Transformadores. O Capítulo 5 revisa os trabalhos relacionados na literatura. O Capítulo 6 descreve o estudo de caso conduzido, detalhando o conjunto de dados, o sistema de detecção implementado, os experimentos, os resultados obtidos e a discussão. Por fim, o Capítulo 7 apresenta as conclusões e considerações finais deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Redes Definidas por *Software*

As redes de computadores consolidaram-se como uma infraestrutura essencial da sociedade contemporânea, suportando operações críticas em setores como saúde, finanças e segurança nacional [3, 4]. Dada essa dependência, a integridade e a disponibilidade desses sistemas tornaram-se críticas, uma vez que falhas podem resultar em prejuízos financeiros severos e comprometer serviços essenciais [5, 6]. No entanto, as arquiteturas de rede tradicionais, com sua inteligência descentralizada e rigidamente acoplada ao hardware, impõem barreiras ao gerenciamento eficiente desses riscos e à escalabilidade da infraestrutura [24].

Para superar essas limitações, com o objetivo de obter uma rede mais programável, automatizada e com custos de gerenciamento reduzidos, surgiu o paradigma das Redes Definidas por *Software* (SDN, do inglês *Software Defined Network*). A principal inovação desta arquitetura reside na separação entre o plano de controle e o plano de dados, estruturando a rede em três camadas: Plano de Aplicação, Plano de Controle e Plano de Dados [24]. A comunicação entre as camadas é feita por interfaces denominadas *Northbound* e *Southbound*. A Figura 1 ilustra essa organização em camadas e suas interações.

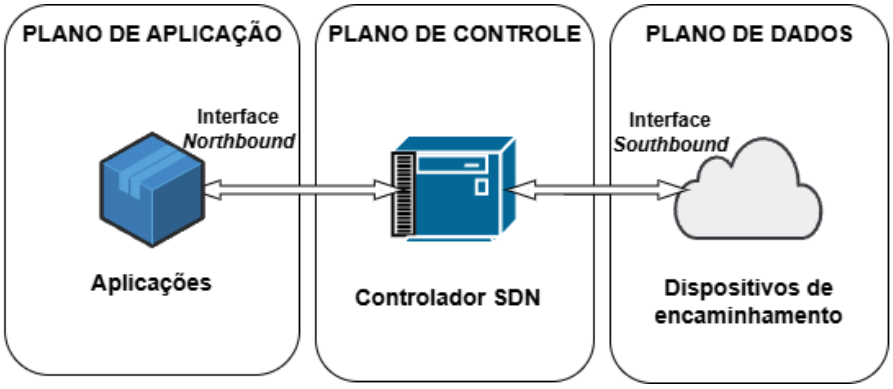


Figura 1 – Arquitetura das Redes Definidas por *Software*. Fonte: próprio autor.

No topo da hierarquia, encontra-se o Plano de Aplicação, onde residem as aplicações de negócios e gerenciamento, como os Sistemas de Detecção de Intrusão. Através da interface *Northbound*, essas aplicações comunicam seus requisitos de alto nível ao controlador no plano de controle [25]. Essa comunicação permite a implementação dinâmica de políticas de segurança e roteamento, sem a necessidade de reconfigurar manualmente o hardware físico.

O núcleo da arquitetura é o Plano de Controle, consolidado em uma entidade de *software* centralizada denominada Controlador SDN. O controlador possui uma visão

global da rede e é responsável por traduzir as políticas definidas pelas aplicações em regras de fluxo. Essa abstração transforma a rede em um sistema flexível, onde a lógica de decisão é desacoplada dos dispositivos de encaminhamento [24].

O Plano de Dados é composto pelos dispositivos de infraestrutura, como *switches* e roteadores. Nesta arquitetura, esses equipamentos são elementos de encaminhamento que operam sob o comando do controlador por meio da interface *Southbound* [25]. O protocolo OpenFlow se estabelece como o padrão predominante para esta interface, fornecendo as instruções necessárias para que o controlador programe as tabelas de fluxo nos dispositivos, definindo ações como encaminhar, descartar ou modificar pacotes [26].

2.2 Anomalias em Redes de Computadores

De modo geral, as anomalias são definidas como padrões que desviam do esperado [27]. No contexto de redes de computadores, o comportamento esperado é definido como tráfego normal, representando os padrões legítimos de comunicação entre dispositivos, aplicações e usuários. Este comportamento regular manifesta-se através de características como volumes de tráfego, distribuição de protocolos, durações de conexão e outras características [28]. Desvios significativos desses padrões estabelecidos são classificados como anomalias, que podem se originar tanto de falhas operacionais quanto de atividades maliciosas [29].

Alguns dos principais tipos de ataques cibernéticos que geram padrões anômalos em redes são: Ataques de Negação de Serviço (*DDoS*, do inglês *Distributed Denial of Service*) e ataques de Varredura de Porta (*Port Scan*) [30]. Ambos os ataques serão definidos a seguir:

- **Ataques de Negação de Serviço Distribuído:** O objetivo desse ataques é esgotar os recursos de um alvo, tornando-o indisponível. Isso é alcançado através da inundação do alvo com um volume massivo de tráfego (pacotes) originado de múltiplas fontes. O padrão anômalo gerado é um aumento abrupto no volume de tráfego convergindo para um único destino (alvo).

- **Varredura de Porta:** Esta é uma técnica de reconhecimento utilizada por atacantes para descobrir serviços vulneráveis em uma máquina alvo. O atacante envia requisições de conexão para um único IP de destino, direcionadas a um grande número de portas diferentes. A análise das respostas permite identificar quais portas estão abertas e os serviços em execução. O padrão anômalo manifesta-se como um IP de destino com alta diversidade de portas acessadas em um curto intervalo de tempo.

A existência de padrões comportamentais nesses ataques permite a detecção automatizada desses ataques. Para identificar tais desvios de comportamento, é necessário construir uma linha de base (*baseline*) que caracterize o perfil de tráfego normal do segmento de rede [31]. Variações significativas em relação ao *baseline* são caracterizadas como anomalias. Utilizando essa premissa como base, são desenvolvidos Sistemas de Detecção de Intrusão em Redes [5].

2.3 Sistemas de Detecção de Intrusão em Redes de Computadores

Devido à crescente dependência das redes de computadores, a prevenção de danos causados por ataques tornou-se uma prioridade. Para prevenir os danos causados por essas ameaças, os Sistemas de Detecção de Intrusão em Redes (NIDS, do inglês *Network Intrusion Detection Systems*) [5] são componentes essenciais da arquitetura de segurança. Esses sistemas operam monitorando o tráfego da rede em tempo real. O objetivo é analisar os dados em busca de padrões que indiquem atividades maliciosas [32]. Quando uma anomalia ou assinatura de ataque é detectada, o NIDS gera um alerta, que é então encaminhado a um administrador de sistema ou a um componente de resposta automatizada. Historicamente, as abordagens de detecção dividem-se em duas categorias principais [33]:

- **Detecção baseada em assinaturas:** Esta abordagem compara o tráfego de rede com um banco de dados de padrões de ataques já conhecidos, as assinaturas. Apesar de essa abordagem apresentar baixas taxas de falsos positivos para ameaças catalogadas, ela é incapaz de detectar ataques desconhecidos (*zero-day*).
- **Detecção baseada em anomalias:** Esta abordagem foca em modelar estatisticamente o comportamento normal da rede. Qualquer desvio significativo desse padrão estabelecido é classificado como uma anomalia. Esta metodologia é capaz de identificar ataques *zero-day*. No entanto, ela tende a apresentar taxas elevadas de falsos positivos e falsos negativos [11].

A comunidade científica empreende esforços significativos para o avanço desta área, embora o campo ainda necessite de desenvolvimento contínuo para superar desafios persistentes [32, 34, 11, 5, 35, 36, 33]. O grupo de pesquisa Orion, da Universidade Estadual de Londrina (UEL), é um dos grupos que contribuem ativamente para este domínio, produzindo pesquisas que visam impulsionar o avanço da área [37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]. Entre os trabalhos publicados, pode-se citar Ruffo *et al.* [15] que propuseram um sistema não supervisionado de detecção e mitigação de anomalias de volume em ambientes SDN. O propósito era modelar o comportamento legítimo da rede e a metodologia utiliza uma f-AnoGAN. Outro trabalho do grupo de Ruffo *et al.*

[11], em que os autores apresentam uma revisão sistemática da literatura sobre NIDS baseados em Aprendizado Profundo para ambientes SDN. O trabalho de Lent *et al.* [14] propôs um sistema de detecção de anomalias para ambientes SDN, onde a metodologia adota uma abordagem baseada em anomalia, utilizando Redes de Unidades Recorrentes Fechadas. Nos últimos anos, o grupo têm se focado em Aprendizado Profundo, por sua capacidade de aprender representações e padrões a partir de grandes volumes de dados, o que possibilita a detecção de anomalias [11].

2.4 Aprendizado Profundo

O Aprendizado Profundo (DL, do inglês *Deep Learning*) representa um ramo do aprendizado de máquina. Ele tem revolucionado diversas áreas da tecnologia, como exemplo: visão computacional, processamento de linguagem natural, diagnóstico médico e segurança cibernética [11, 50]. Diferente dos algoritmos convencionais de aprendizado de máquina, que eram limitados pela necessidade de extração manual de características, o Aprendizado Profundo tem a capacidade de processar os dados em sua forma bruta, identificando padrões complexos através de múltiplas camadas de abstração [51, 50].

2.4.1 Redes Neurais

A base do Aprendizado Profundo é a Rede Neural Artificial (ANN, do inglês *Artificial Neural Network*). Inspirada na estrutura do cérebro humano, uma ANN é composta por unidades de processamento simples, os neurônios, organizados em camadas interconectadas: uma camada de entrada, que recebe os dados; uma ou mais camadas ocultas, onde ocorre o processamento principal; e uma camada de saída, que produz o resultado [51]. A Figura 2 ilustra uma ANN, que possui n entradas (x_1 a x_n) alimentando K camadas ocultas ($h^{(1)}$ a $h^{(K)}$) para processamento, que geram m saídas (y_1 a y_m).

Modelos de Aprendizado Profundo são essencialmente Redes Neurais Artificiais com múltiplas camadas ocultas. Existem diversas arquiteturas de Aprendizado Profundo, tais como Redes Neurais Convolucionais, Autocodificadores Variacionais e Transformadores. Essas arquiteturas são projetadas para melhorar a eficácia do aprendizado para diferentes objetivos, incluindo a classificação de imagens, a geração de texto e a detecção de anomalias. Todas essas arquiteturas compartilham o mesmo objetivo: aproximar uma função que modela padrões ou relações estruturais.

A capacidade de aproximar funções vem da computação realizada em cada camada da rede, que combina duas etapas: uma transformação linear, seguida por uma função de ativação não linear. Primeiro, cada neurônio de uma camada calcula uma soma ponderada das saídas da camada anterior, adicionando um termo que representa o viés de ativação do neurônio [51]. Matematicamente, para uma camada inteira, essa operação linear pode

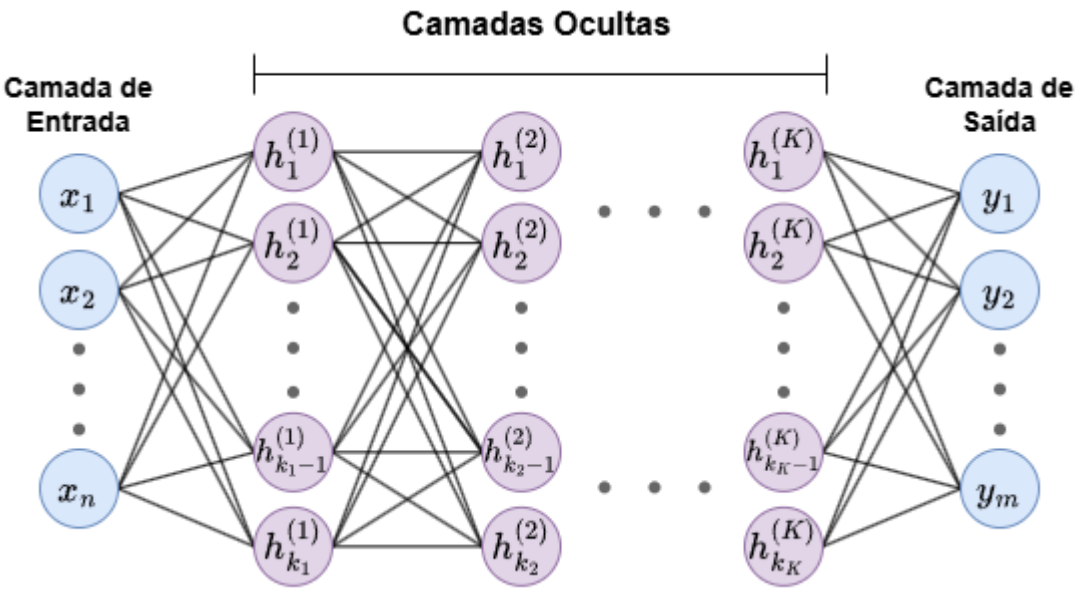


Figura 2 – Rede Neural Artificial. Fonte: próprio autor.

ser representada da seguinte forma:

$$z = Wx + b \tag{2.1}$$

- x é o vetor de entrada (dados de entrada ou saídas da camada anterior).
- W é a matriz de pesos, que representa o peso das conexões entre os neurônios.
- b é o vetor de viés, que permite um ajuste constante na saída.
- z é o resultado da transformação linear.

Se as redes neurais utilizassem apenas essa operação, elas estariam limitadas a aprender apenas relações lineares, pois somar várias camadas lineares resultaria apenas em outra relação linear. Para modelar padrões mais complexos, é necessária outra etapa: a aplicação de uma função de ativação não linear $g(z)$ [1]. A saída final da camada a é então dada por:

$$a = g(z) = g(Wx + b) \tag{2.2}$$

A Unidade Linear Retificada (ReLU, do inglês *Rectified Linear Units*) é uma das funções de ativação mais populares, em grande parte por facilitar a aproximação da função objetivo [52]. Ela opera da seguinte forma: retorna 0 para entradas negativas ou a própria entrada para valores não negativos ($ReLU(z) = max(0, z)$). Outras funções comuns incluem a Sigmoide e Tangente Hiperbólica.

Uma rede neural é formada pelo encadeamento de várias dessas camadas. A saída a de uma camada se torna a entrada x para a camada seguinte. Dessa forma, a rede inteira pode ser vista como uma única função, parametrizada pelos seus pesos e vieses [1]. A combinação de transformações lineares e ativações não lineares, repetidas por múltiplas camadas, permite que as redes neurais modelem o mapeamento entre as entradas e saídas de forma a atingir um objetivo específico [53, 1]. A função computada pela rede neural é representada como:

$$\hat{y} = f_\phi(x) \tag{2.3}$$

Onde:

- x é a entrada inicial da rede neural.
- ϕ representa o conjunto de todos os parâmetros aprendíveis da rede (todos os pesos W e vieses b de todas as camadas).
- f_ϕ é a função que a rede implementa.
- $\hat{y}$ é a predição do modelo dado o valor de entrada x.

A capacidade de representação de relações das redes neurais é formalizada pelo Teorema da Aproximação Universal. Este teorema afirma que uma rede neural com apenas uma camada oculta, contendo um número suficiente de neurônios, pode aproximar qualquer função contínua com um grau arbitrário de precisão [53]. Na prática, isso significa que, dada uma rede neural com quantidade suficiente de parâmetros, ela é capaz de modelar relações complexas entre entradas e saídas, como exemplo a que distingue um tráfego de rede normal de um anômalo.

Pode-se aprimorar a compreensão intuitiva deste teorema observando como uma rede neural com uma camada oculta aproxima uma função, ilustrada na Figura 3. Ela demonstra uma rede neural com três neurônios na camada oculta que tenta aproximar a função alvo $y = \text{sen}(\pi x)$. Será analisada a construção passo a passo:

1. **Transformação linear da camada oculta:** O primeiro passo em cada neurônio é aplicar uma transformação linear ($z_i = W_i x + b_i$) à entrada x. Os gráficos (a), (b) e (c) ilustram o resultado desta operação para cada um dos três neurônios. O resultado de cada um é uma função linear simples, uma reta. Observa-se que, somando todos os resultados, geraria uma função com apenas uma região linear.

2. **Aplicação da camada de ativação:** Em seguida, a saída linear de cada neurônio é passada através de uma função de ativação não linear, $\text{ReLU}(z)$. Os gráficos (d),

(e) e (f) mostram o resultado dessa transformação. As retas são convertidas em funções não lineares (a_1, a_2, a_3). Nota-se que, com essas ativações, quatro regiões lineares distintas podem ser estabelecidas pela soma das funções: (($a_1 = 0$, $a_2 = 0$, $a_3 = 0$), ($a_1 > 0$, $a_2 = 0$, $a_3 = 0$), ($a_1 > 0$, $a_2 > 0$, $a_3 = 0$), ($a_1 > 0$, $a_2 > 0$, $a_3 > 0$)).

3. **Ponderação pela camada de saída:** A camada de saída recebe essas três funções como entrada e aplica uma nova ponderação, multiplicando cada uma por um peso de saída ($W_{\text{out},1}$, $W_{\text{out},2}$, $W_{\text{out},3}$). Os gráficos (g), (h) e (i) mostram essas funções já ponderadas. Nota-se no gráfico (h) que um peso de saída negativo multiplica a_2, permitindo que a rede modele comportamentos decrescentes.

4. **Aproximação final:** Por fim, a saída final da rede (a linha preta contínua no gráfico (j)) é a soma de todos esses componentes ponderados da etapa anterior, se aproximando da função seno para o intervalo $[0, 2]$. Percebe-se que, com esse número de regiões lineares, não é possível aproximar a função seno com exatidão, nem para este pequeno intervalo $[0, 2]$. Caso houvesse mais neurônios seria possível ter mais regiões lineares, o que permitiria uma aproximação mais precisa dessa função.

O teorema pode ser compreendido intuitivamente a partir do fato de que, ao posicionar, escalar e combinar um número suficiente de regiões lineares, uma rede neural consegue construir uma aproximação para qualquer função contínua. O número de regiões pode ser aumentado por duas abordagens: adicionando mais neurônios a uma camada, o que aumenta a largura da rede, ou mais camadas ocultas, o que incrementa a profundidade da rede [1].

No contexto deste trabalho, não há uma função alvo explícita, como a função seno descrita no exemplo. Em vez disso, a rede neural aprende a construir uma função que modela a distribuição dos dados de tráfego normais. Isso torna possível a identificação de dados anômalos [15, 11]. Os mecanismos que permitem a rede neural construir tal função serão estudados nas próximas seções.

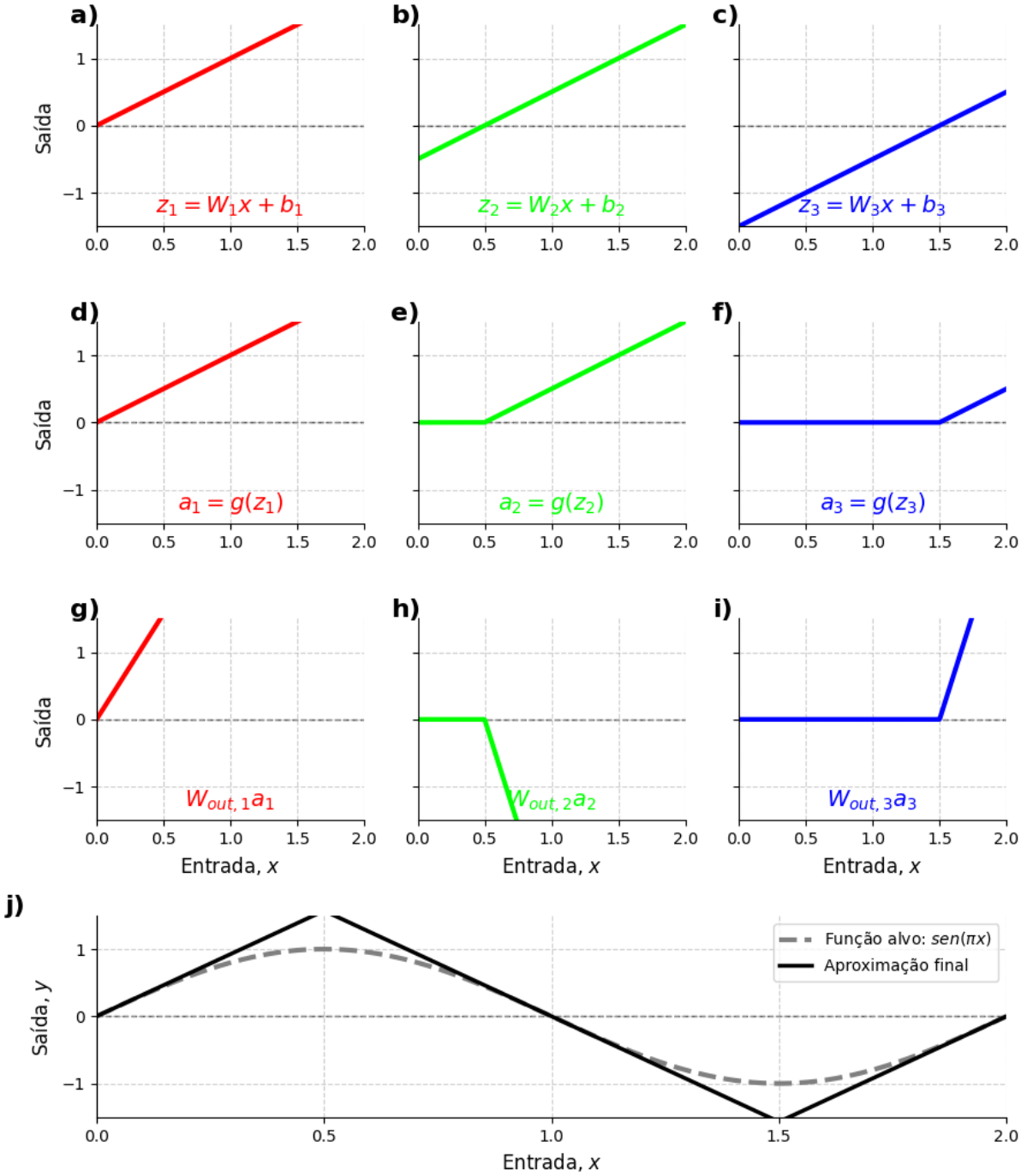


Figura 3 – Aproximação da função seno com ANN. Fonte: adaptado de Prince [1].

2.4.2 Aprendizado com Dados

Na seção anterior, foi estabelecido que uma Rede Neural Artificial, em teoria, pode aproximar qualquer função contínua. No entanto, o desafio fundamental reside em determinar os parâmetros que permitem a rede neural aproximar a função desejada.

O ponto de partida para a modelagem de uma função objetivo é o conjunto de dados, do qual a ANN extrairá informações para aproximá-la. Este conjunto consiste em exemplos que incorporam os padrões ou relações estruturais que a função objetivo deve modelar. A natureza deste conjunto define o paradigma de aprendizado, como o supervisionado e o não supervisionado [54]. Em notação matemática, a composição dos conjuntos de dados desse paradigma é dada por:

- **Conjunto de dados supervisionados:** $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- **Conjunto de dados não supervisionados:** $S = \{x_1, x_2, \ldots, x_n\}$

Na teoria de Aprendizado de Máquina, o conjunto de dados S, independentemente do paradigma, representa uma amostra finita do universo de dados. Assume-se que essa amostra foi extraída de uma distribuição de dados verdadeira e desconhecida P_{data}, sob a condição de que os exemplos são independentes e identicamente distribuídos (*i.i.d.*). A suposição *i.i.d.* significa que cada ponto de dado foi coletado sem influenciar os outros (independência) e que todos os pontos se originam do mesmo fenômeno estatístico (identicamente distribuídos) [55]. Essa premissa é a base que permite generalizar os aprendizados obtidos com a amostra S para a distribuição completa. O universo de dados será denotado como D, de modo que $S \subset D$.

No aprendizado supervisionado, o conjunto S fornece pares de dados com rótulos. Assume-se a existência de uma função alvo $f_{\text{real}} : X \to Y$ que descreve a relação que se deseja aproximar. O objetivo é, então, estimar os parâmetros ϕ da função f_ϕ de modo que esta se aproxime da função f_{real}, com base nos exemplos fornecidos pelo conjunto S [56, 54].

No aprendizado não supervisionado, o conjunto S é formado apenas pelos dados de entrada, sem rótulos correspondentes. Nele, assume-se a existência de relações estruturais nos dados, descrita pela distribuição de probabilidade P_{data}. O objetivo é estimar os parâmetros ϕ da função f_ϕ de modo que esta modele as propriedades dessa distribuição a partir dos dados presentes no conjunto S [1, 57].

2.4.3 Funções de Perda

Nas seções anteriores foi estabelecido que a Rede Neural Artificial é uma função aproximadora f_ϕ, com o objetivo de encontrar os parâmetros ϕ que modelam os dados S.

O mecanismo para alcançar este objetivo é o processo de treinamento, que se baseia em dois componentes principais: a função de perda e o algoritmo de otimização.

A função de perda, denotada $L(\phi)$, é a métrica utilizada para quantificar a disparidade entre as predições do modelo f_ϕ e seu objetivo. O propósito do treinamento é encontrar o conjunto de parâmetros ϕ que minimiza o valor desta função de perda para o conjunto de dados S [54]. Dada uma função de perda viável para a tarefa do modelo, o problema de aprendizado com dados se transforma em um problema de otimização.

A teoria da informação e a estatística fornecem uma base para justificar a escolha das funções de perda [54]. Nessa perspectiva, a rede neural f_ϕ aprende a definir os parâmetros ϕ de uma distribuição de probabilidade $P_{\text{model}}(.;\phi)$ [1]. O objetivo é encontrar os parâmetros ϕ da rede neural, de modo que $P_{\text{model}}(.;\phi)$ modele efetivamente a distribuição real do conjunto de dados, P_{data} [55].

Dependendo do tipo de aprendizado, o modelo pode aprender a distribuição incondicional dos dados, $P_{\text{model}}(x;\phi)$, para aproximar $P_{\text{data}}(x)$, ou pode aprender a distribuição condicional, $P_{\text{model}}(x|y;\phi)$, para aproximar $P_{\text{data}}(x|y)$.

A função de perda $L(\phi)$ pode ser formulada como uma medida de divergência entre a distribuição real P_{data} e a distribuição do modelo $P_{\text{model}}(.;\phi)$. Uma métrica amplamente utilizada para esta finalidade é a Divergência de Kullback-Leibler (DKL) [58]. Esta métrica, derivada da teoria da informação, quantifica a discrepância entre duas distribuições probabilísticas .

Para o aprendizado não supervisionado, a divergência é definida como [58]:

$$D_{KL}(P_{\text{data}}(x) \parallel P_{\text{model}}(x;\phi)) = E_{x\sim P_{\text{data}}}\left[\log\left(\frac{P_{\text{data}}(x)}{P_{\text{model}}(x;\phi)}\right)\right] \tag{2.4}$$

O objetivo do modelo é encontrar os parâmetros ϕ que minimizam esta divergência:

$$L(\phi) = D_{KL}(P_{\text{data}}(x) \parallel P_{\text{model}}(x;\phi)) \tag{2.5}$$

$$\phi^* = \arg\min_{\phi} D_{KL}(P_{\text{data}}(x) \parallel P_{\text{model}}(x;\phi)) \tag{2.6}$$

Este termo pode ser simplificado da seguinte forma, ao expandir a definição da DKL:

$$D_{KL}(P_{\text{data}}(x) \parallel P_{\text{model}}(x;\phi)) = E_{x\sim P_{\text{data}}}[\log P_{\text{data}}(x)] - E_{x\sim P_{\text{data}}}[\log P_{\text{model}}(x;\phi)] \tag{2.7}$$

O primeiro termo, $E_{x\sim P_{\text{data}}}[\log P_{\text{data}}(x)]$, é uma constante em relação a ϕ. Portanto, minimizar a DKL é equivalente a minimizar a Log-Verossimilhança Negativa (NLL, do inglês *Negative Log Likelihood*) [54]:

$$\phi^* = \arg\min_{\phi} E_{x \sim P_{\text{data}}}[-\log P_{\text{model}}(x; \phi)] \tag{2.8}$$

De maneira similar, para modelos supervisionados, é possível obter a seguinte equação para a distribuição condicional dos dados [1]:

$$\phi^* = \arg\min_{\phi} E_{(x,y) \sim P_{\text{data}}}[-\log P_{\text{model}}(y \mid x; \phi)] \tag{2.9}$$

Em ambos os casos, (2.8) e (2.9), o problema de aprendizado se reduz a encontrar os parâmetros ϕ que maximizam a probabilidade de o modelo ter gerado conjunto de dados S. Este é o princípio da Máxima Verossimilhança (MLE, do inglês *Maximum Likelihood Estimation*) [54]. Utilizando o princípio da MLE, é possivel derivar outras funções de perda ao fazer suposições sobre P_{data} e P_{model}:

1. **Erro quadrático:** Se a tarefa for de regressão e $P_{\text{model}}(y \mid x; \phi)$ for assumida como uma distribuição Gaussiana com variância constante, minimizar a NLL é equivalente a minimizar o Erro Quadrático Médio (MSE, do inglês *Mean Square Error*) entre a predição e o alvo [54].

2. **Entropia cruzada binária:** Se a tarefa for de classificação e $P_{\text{model}}(y \mid x; \phi)$ for uma distribuição de Bernoulli ou Categórica, a NLL será a função de Entropia Cruzada Binária (BCE, do inglês *Binary Cross Entropy*) [1].

3. **Função de perda dos Autocodificadores Variacionais:** No contexto não supervisionado, o cálculo de $P_{\text{model}}(x; \phi)$ é muitas vezes intratável. Métodos como os Autocodificadores Variacionais otimizam um limite inferior da log-verossimilhança [20].

2.4.4 Otimização

Com a função de perda $L(\phi)$ definida, o problema de aprendizado se consolida como um problema de otimização: encontrar o conjunto de parâmetros ϕ^* que minimiza $L(\phi)$. A otimização é feita utilizando métodos baseados em gradiente, assumindo que $L(\phi)$ é diferenciável em relação a ϕ [51].

O gradiente da função de perda em relação aos parâmetros, $\nabla_\phi L(\phi)$, é um vetor que aponta na direção de maior crescimento da função. Ele é definido como o vetor das derivadas parciais de $L(\phi)$ em relação a cada parâmetro ϕ_i [1]:

$$\nabla_\phi L(\phi) = \left[\frac{\partial L(\phi)}{\partial \phi_1}, \frac{\partial L(\phi)}{\partial \phi_2}, \ldots, \frac{\partial L(\phi)}{\partial \phi_n}\right]^T \tag{2.10}$$

Para minimizar $L(\phi)$, o algoritmo de otimização ajusta iterativamente os parâmetros ϕ na direção oposta à do gradiente. Um dos algoritmos utilizados para este processo é o Gradiente Descendente, cuja regra de atualização é definida como [1]:

$$\phi_{t+1} = \phi_t - \eta \nabla_\phi L(\phi_t) \tag{2.11}$$

Onde ϕ_t representa os parâmetros na iteração t, e η é um hiperparâmetro positivo, denominado taxa de aprendizado, que controla o tamanho do passo dado em cada iteração.

Na prática, o algoritmo base utilizado é o Gradiente Descendente Estocástico (SGD, do inglês *Stochastic Gradient Descent*), que torna o processo eficiente ao estimar o gradiente utilizando partes do conjunto de dados (lotes) [59]. Para acelerar e estabilizar a convergência, variantes adaptativas do SGD, como Adam [60], AdamW [61] e RMSprop [59], são utilizadas, com o objetivo de ajustar η dinamicamente e suavizar a trajetória.

Para calcular eficientemente o gradiente $\nabla_\phi L(\phi)$ em uma rede neural profunda, que pode ser vista como a composição de várias funções, utiliza-se o algoritmo de retropropagação [62, 51]. A retropropagação é a aplicação da regra da cadeia do cálculo, que computa a contribuição de cada parâmetro em ϕ na perda final, propagando o gradiente da camada de saída até a camada de entrada [1]. Isso torna possível atualizar todos os parâmetros da rede neural de forma que a função de perda seja minimizada localmente.

Em suma, o processo de otimização em Aprendizado Profundo consiste em calcular o gradiente da função de perda via retropropagação e atualizar os parâmetros ϕ repetidamente, de modo que os parâmetros ϕ se ajustem para minimizar a função de perda $L(\phi)$. Minimizando a função de perda, a rede neural f_ϕ fica mais próxima de realizar a tarefa objetiva de acordo com o conjunto de dados S e a função de perda escolhida.

2.4.5 Métricas de Avaliação

Para comparar o desempenho dos modelos de Aprendizado Profundo no conjunto de dados S, no contexto de classificação binária, são utilizadas métricas fundamentadas na matriz de confusão, exibida na Figura 4. A avaliação baseia-se na comparação entre o rótulo verdadeiro da amostra e a predição realizada pelo modelo. Define-se a seguinte convenção de nomenclatura dos rótulos das classes:

- **Positivo (P):** Refere-se à presença do evento de interesse, ou seja, tráfego anômalo.
- **Negativo (N):** Refere-se à ausência de ameaças, ou seja, tráfego normal.

Com base nessas definições, os quatro resultados possíveis da classificação são descritos a seguir:

- **Verdadeiro Positivo (VP):** Amostra anômala classificada corretamente como anomalia.
- **Verdadeiro Negativo (VN):** Amostra normal classificada corretamente como normal.
- **Falso Positivo (FP):** Amostra normal classificada incorretamente como anomalia.
- **Falso Negativo (FN):** Amostra anômala classificada incorretamente como normal.

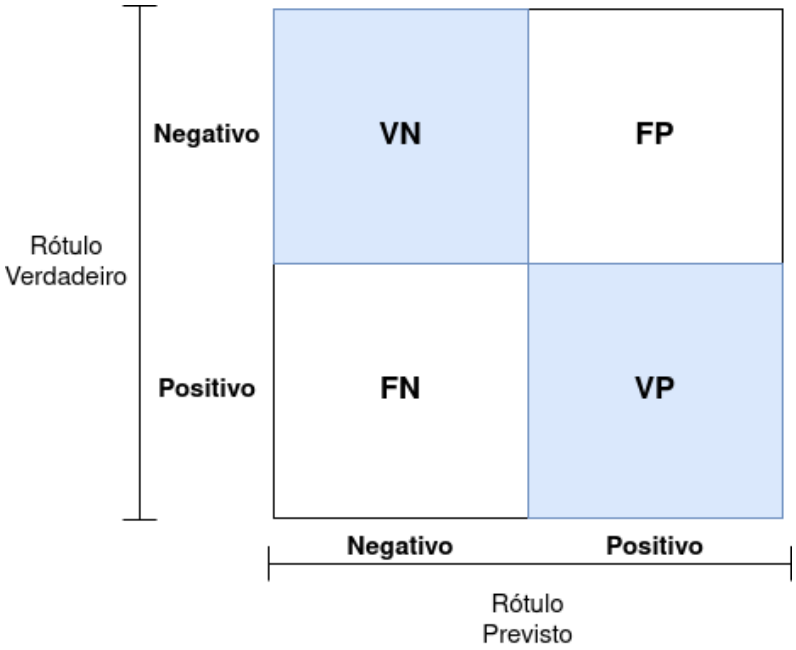


Figura 4 – Matriz de Confusão. Fonte: próprio autor.

A partir destes componentes, as seguintes métricas são derivadas [11]:

Acurácia: Mede a proporção de predições corretas (VP e VN) em relação ao número total de amostras.

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN}$$

Precisão: Mede a proporção de amostras classificadas como positivas que eram de fato positivas.

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

Revocação (*recall*): Também conhecida como Taxa de Verdadeiros Positivos (TPR, do inglês *True Positive Rate*), mede a proporção de amostras positivas reais que foram

corretamente identificadas pelo modelo.

$$\text{Revocação} = \frac{VP}{VP + FN}$$

***F1-score*:** É a média harmônica da Precisão e da Revocação. Esta métrica é robusta em cenários desbalanceados, pois penaliza modelos que otimizam apenas uma em detrimento da outra.

$$\textit{F1-score} = 2 \times \frac{\text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}} = \frac{2VP}{2VP + FP + FN}$$

Coeficiente de Correlação de Matthews (MCC, do inglês *Matthews Correlation Coefficient*): É considerada uma das métricas mais robustas para classificação binária desbalanceada. Ela mede a correlação entre as predições e os rótulos verdadeiros. Os valores variam de -1 (total desacordo) a +1 (acordo perfeito), onde 0 indica um desempenho aleatório.

$$\text{MCC} = \frac{(VP \times VN) - (FP \times FN)}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}}$$

Taxa de Falsos Positivos (FPR, do inglês *False Positive Rate*): Quantifica a proporção de amostras normais que foram incorretamente classificadas como positivas. É calculada pela razão entre os falsos positivos e o total de amostras negativas reais.

$$FPR = \frac{FP}{FP + VN}$$

Área Sob a Curva ROC (AUROC, do inglês *Area Under the Receiver Operating Characteristic Curve*): A Curva ROC (do inglês *Receiver Operating Characteristic*) representa graficamente a Taxa de Verdadeiros Positivos contra a Taxa de Falsos Positivos para diferentes limiares de classificação. A AUROC mede a área sob esta curva. O valor de 1.0 indica um classificador perfeito, enquanto 0.5 indica um classificador aleatório.

3 AUTOCODIFICADORES VARIACIONAIS

O Autocodificador Variacional (VAE, do inglês *Variational Autoencoder*), proposto por Kingma e Welling (2014) em [20], é um modelo generativo probabilístico cujo objetivo é aprender e aproximar a distribuição probabilística que gerou o conjunto de dados S, $P_{data}(x)$. Por realizar essa tarefa aprendendo diretamente a partir dos dados brutos, sem a necessidade de rótulos, ele se enquadra na categoria de aprendizado não supervisionado.

A arquitetura do VAE é baseada nos Autocodificadores (AE, do inglês *Autoencoder*), que são compostos por duas redes neurais: uma representando o codificador e a outra representando o decodificador. O codificador recebe um dado de entrada, como um vetor de características de um fluxo de rede, e o comprime em uma representação menor e mais simples. Em seguida, o decodificador recebe essa representação compacta e tenta reconstruir o dado original da forma mais precisa possível [63].

O VAE, assim como o AE, opera sob o princípio de codificar os dados de entrada em uma representação de menor dimensão, que é chamada de vetor latente, e esse vetor está contido em um espaço latente. No entanto, a diferença entre essas duas arquiteturas reside na maneira como esse vetor latente é gerado.

No AE convencional, a codificação é determinística: o codificador mapeia a entrada para um vetor latente (z) fixo e único. Para uma mesma entrada, a saída do codificador será sempre a mesma, como ilustrado na Figura 5. No VAE, em contrapartida, a codificação é probabilística. O codificador utiliza os dados de entrada para gerar parâmetros de uma distribuição probabilística, tipicamente uma distribuição normal, com média (μ) e variância (σ^2). O vetor latente z é então amostrado dessa distribuição, como representado na Figura 6.

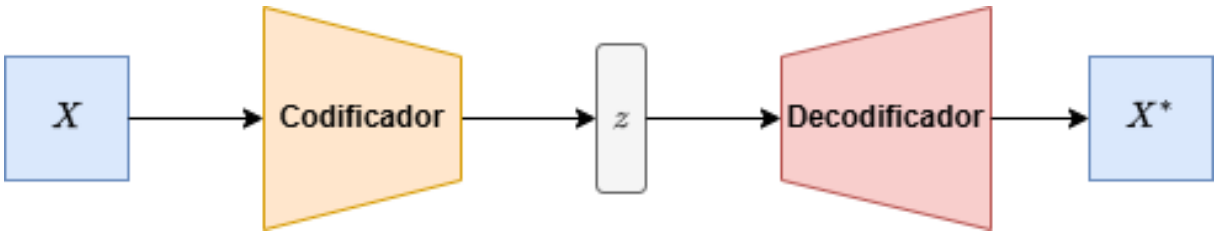


Figura 5 – Arquitetura do Autocodificador. Fonte: próprio autor.

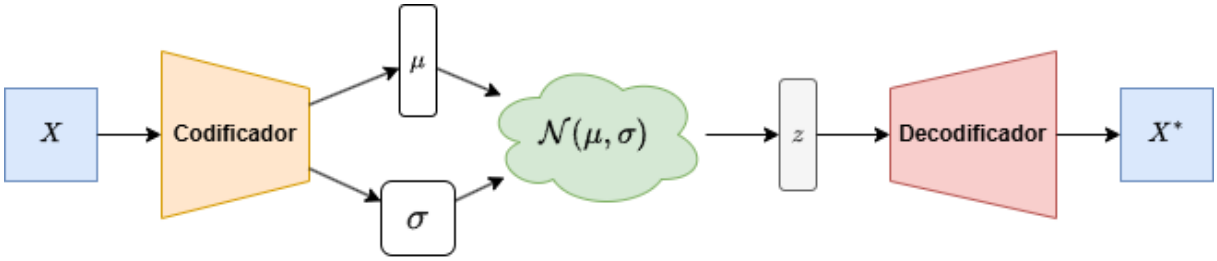


Figura 6 – Arquitetura do Autocodificador Variacional em alto nível. Fonte: próprio autor.

Formalmente, a arquitetura do VAE é definida por dois modelos probabilísticos parametrizados por redes neurais [20]:

1. O Codificador é a distribuição variacional, denotada $Q(z \mid x; \theta)$. É uma rede neural, cujos parâmetros são θ, que recebe x e define os parâmetros da distribuição probabilística do espaço latente para x (média $\mu_\theta(x)$ e variância $\sigma^2_\theta(x)$).

2. O Decodificador é o modelo generativo, denotado $P(x \mid z; \phi)$. É uma rede neural, cujos parâmetros são ϕ, que recebe a amostra latente z e define os parâmetros da distribuição sobre x^*.

O objetivo do treinamento será otimizar ambos os conjuntos de parâmetros ϕ e θ simultaneamente, através do algoritmo de retropropagação e da maximização do Limite Inferior da Evidência, que será derivado a seguir.

3.1 Limite Inferior da Evidência

O Limite Inferior da Evidência (ELBO, do inglês *Evidence Lower Bound*) é a função que será utilizada no processo de treinamento do VAE. Para derivá-lo, é utilizado o princípio da Máxima Verossimilhança, que afirma que a melhor estimativa para os parâmetros de um modelo estatístico é aquela que maximiza a probabilidade de ter gerado os dados observados (o conjunto de dados) 2.4.3 [64, 54].

Para aproximar a distribuição real P_{data}, define-se uma distribuição probabilística parametrizada $P_{\text{model}}(x; \phi)$. O método estima os parâmetros ϕ que maximizam a verossimilhança do modelo $L(\phi \mid S) \equiv P_{\text{model}}(S; \phi)$, que é a probabilidade de o modelo ter gerado o conjunto de dados S [1]. O objetivo desse modelo é:

$$\phi^* = \arg\max_{\phi} \left[\log P_{\text{model}}(S; \phi)\right], \text{ para } P_{\text{model}}(S; \phi) \equiv P_{\text{model}}(x_1, x_2, \ldots, x_n; \phi) \tag{3.1}$$

Dado que as amostras do conjunto de dados S são assumidas como *i.i.d.*, a log-verossimilhança pode ser reescrita como a soma das log-verossimilhanças individuais [64]:

$$\log P_{\text{model}}(S;\phi) = \log\left(\prod_{i=1}^{n} P_{\text{model}}(x^{(i)};\phi)\right) = \sum_{i=1}^{n} \log P_{\text{model}}(x^{(i)};\phi) \tag{3.2}$$

O objetivo da equação(3.1) é equivalente a maximizar esta soma, porém o desafio da Máxima Verossimilhança surge em sua implementaçào: para otimizar a soma, é necessário calcular cada termo $\log P_{\text{model}}(x^{(i)};\phi)$ individualmente. Para a maioria dos modelos, o cálculo direto da verossimilhança marginal $P_{\text{model}}(x;\phi)$ para uma única amostra é computacionalmente intratável, levando um tempo impraticável para ser calculado [1, 20], devido a um fenômeno chamado maldição da dimensionalidade [65].

Para contornar essa intratabilidade, a abordagem proposta por Kingma *et al.* [20] introduz uma variável adicional ao modelo: a variável latente z. Com a introdução da variável latente z, o modelo generativo $P_{\text{model}}(x;\phi)$ é definido pela marginalização da probabilidade conjunta $P_{\text{model}}(x,z;\phi)$. Esta probabilidade é fatorada em uma distribuição a priori sobre z, dada por $P_{\text{model}}(z;\phi)$, e a distribuição condicional $P_{\text{model}}(x \mid z;\phi)$. A log-verossimilhança que se deseja maximizar é a integral sobre todas as possíveis configurações de z:

$$\log P_{\text{model}}(x;\phi) = \log \int P_{\text{model}}(x \mid z;\phi) P_{\text{model}}(z;\phi) dz \tag{3.3}$$

Esta integral ainda é intratável. A solução para esse problema consiste em introduzir um segundo modelo, o codificador $Q(z \mid x;\theta)$, parametrizado por θ. A derivação do ELBO se baseia na relação entre $\log P_{\text{model}}(x;\phi)$ e a Divergência de Kullback-Leibler entre o codificador $Q(z \mid x;\theta)$ e o posterior verdadeiro $P_{\text{model}}(z \mid x;\phi))$. Este posterior verdadeiro descreve a distribuição probabilística ideal das variáveis latentes z dado um ponto x, que também é intratável [66].

Inicia-se com a definição da DKL [66]:

$$D_{KL}(Q(z \mid x;\theta) \parallel P_{\text{model}}(z \mid x;\phi)) = E_{z\sim Q}\left[\log \frac{Q(z \mid x;\theta)}{P_{\text{model}}(z \mid x;\phi)}\right] \tag{3.4}$$

Expandindo $\log P_{\text{model}}(z \mid x;\phi)$ utilizando a regra de Bayes ($P(z \mid x) = \frac{P(x,z)}{P(x)}$):

$$D_{KL}(Q \parallel P_{\text{model}}) = E_{z\sim Q}\left[\log Q(z \mid x;\theta) - \log\left(\frac{P_{\text{model}}(x,z;\phi)}{P_{\text{model}}(x;\phi)}\right)\right] \tag{3.5}$$

$$D_{KL}(Q \parallel P_{\text{model}}) = E_{z\sim Q}\left[\log Q(z \mid x;\theta) - \log P_{\text{model}}(x,z;\phi) + \log P_{\text{model}}(x;\phi)\right] \tag{3.6}$$

Como $\log P_{\text{model}}(x;\phi)$ não depende de z, o termo pode ser removido da expectativa. Reorganizando a equação para isolar a log-evidência $\log P_{\text{model}}(x;\phi)$ (o termo que deve ser maximizado):

$$\log P_{\text{model}}(x;\phi) = E_{z\sim Q}\left[\log P_{\text{model}}(x,z;\phi) - \log Q(z \mid x;\theta)\right] + D_{KL}(Q \parallel P_{\text{model}}) \tag{3.7}$$

O primeiro termo desta equação é definido como o ELBO, representado por $ELBO(\phi,\theta)$ [20]:

$$ELBO(\phi,\theta) = E_{z\sim Q(z|x;\theta)}\left[\log P_{\text{model}}(x,z;\phi) - \log Q(z \mid x;\theta)\right] \tag{3.8}$$

Substituindo $ELBO(\phi,\theta)$ na equação anterior, obtém-se a seguinte relação:

$$\log P_{\text{model}}(x;\phi) = ELBO(\phi,\theta) + D_{KL}(Q(z \mid x;\theta) \parallel P_{\text{model}}(z \mid x;\phi)) \tag{3.9}$$

Por definição, a Divergência KL é não negativa ($D_{KL} \geq 0$). Isso garante que $ELBO(\phi,\theta)$ é sempre menor ou igual à log-verossimilhança, formando o seguinte limite inferior:

$$\log P_{\text{model}}(x;\phi) \geq ELBO(\phi,\theta) \tag{3.10}$$

Por conta dessa inequaçào, a otimização do VAE consiste em maximizar o $ELBO(\phi,\theta)$, em relação a ambos os conjuntos de parâmetros (ϕ do decodificador e θ do codificador). Esta maximização tem dois efeitos:

1. Maximiza-se o próprio limite inferior, $ELBO(\phi,\theta)$, o que indiretamente eleva o valor da log-verossimilhança $\log P_{\text{model}}(x;\phi)$ (o objetivo original).
2. Minimiza-se a divergência $D_{KL}(Q(z \mid x;\theta) \parallel P_{\text{model}}(z \mid x;\phi)$, forçando a distribuição do codificador $Q(z \mid x;\theta)$ a se tornar uma aproximação precisa da posterior verdadeira $P_{\text{model}}(z \mid x;\phi)$.

Para a implementação e interpretação, o $ELBO(\phi,\theta)$, é decomposto. Ao aplicar a regra do produto à probabilidade conjunta, $P_{\text{model}}(x,z;\phi) = P_{\text{model}}(x \mid z;\phi)P_{\text{model}}(z;\phi)$, o $ELBO(\phi,\theta)$ é decomposto em dois termos:

$$ELBO(\phi,\theta) = E_{z\sim Q}\left[\log P_{\text{model}}(x \mid z;\phi) + \log P_{\text{model}}(z;\phi) - \log Q(z \mid x;\theta)\right] \tag{3.11}$$

$$ELBO(\phi, \theta) = \underbrace{E_{z \sim Q(z|x;\theta)} \left[\log P_{\text{model}}(x \mid z; \phi)\right]}_{\text{Termo de Reconstrução}} - \underbrace{D_{KL}(Q(z \mid x; \theta) \parallel P_{\text{model}}(z; \phi))}_{\text{Termo de Regularização}} \quad (3.12)$$

O objetivo $ELBO(\phi, \theta)$, que é maximizado, é composto pelos seguintes termos [1]:

1. **Termo de Reconstrução:** $E_{z \sim Q}[\log P_{\text{model}}(x \mid z; \phi)]$ Esta é a log-verossimilhança esperada da reconstrução dos dados. Este termo força o decodificador $P_{\text{model}}(x \mid z; \phi)$ a aprender a reconstruir a entrada x de forma precisa, a partir de uma amostra latente z fornecida pelo codificador Q. Na prática, este termo corresponde a uma função de perda de reconstrução, como o Erro Quadrático (para dados Gaussianos) ou Entropia Cruzada (para dados binários).

2. **Termo de Regularização:** $-D_{KL}(Q(z \mid x; \theta) \parallel P_{\text{model}}(z; \phi))$ Este termo é a divergência entre a distribuição do codificador $Q(z \mid x; \theta)$ e a distribuição a priori $P_{\text{model}}(z; \phi)$, frequentemente escolhida como uma Gaussiana padrão $\mathcal{N}(0, I)$. Maximizar este termo negativo equivale a minimizar a DKL, forçando o codificador a produzir distribuições latentes que se assemelhem à prior, $P_{\text{model}}(z; \phi)$. Esse termo atua como um regularizador, possibilitando que o espaço latente z seja contínuo e estruturado, o que permite a geração de novas amostras.

3.2 Implementação Computacional

A derivação do ELBO na seção anterior estabeleceu a função objetivo $ELBO(\phi, \theta)$ que deve ser maximizada. Esta função depende dos dois conjuntos de parâmetros da arquitetura: θ para o codificador $Q(z \mid x; \theta)$ e ϕ para o decodificador $P_{\text{model}}(x \mid z; \phi)$. No entanto, a otimização desta função com métodos baseados em gradiente ainda enfrenta dois desafios computacionais que precisam ser endereçados para a implementação do modelo.

O primeiro desafio é a amostragem estocástica. O Termo de Reconstrução do ELBO, $E_{z \sim Q}[\log P_{\text{model}}(x \mid z; \phi)]$, é uma expectativa. Na prática, esta expectativa é aproximada via Monte Carlo [54], amostrando um z da distribuição do codificador $Q(z \mid x; \theta)$ e calculando $\log P_{\text{model}}(x \mid z; \phi)$ para essa amostra. Além disso, a operação de amostragem $z \sim Q(z \mid x; \theta)$ não é derivável, impedindo que o gradiente da perda de reconstrução seja calculado para atualizar os parâmetros θ do codificador.

Para contornar este problema, a arquitetura VAE implementa o Truque da Reparametrização (*Reparameterization Trick*) [20], que torna a operação de amostragem z derivável. O codificador $Q(z \mid x; \theta)$ é treinado para gerar os parâmetros determinísticos de uma distribuição (tipicamente a média $\mu_\theta(x)$ e o desvio padrão $\sigma_\theta(x)$ de uma Gaussiana).

O vetor latente z é gerado por uma função determinística que combina esses parâmetros com um ruído ϵ, amostrado de uma distribuição Gaussiana padrão $\mathcal{N}(0, I)$. A geração de z torna-se:

$$z = \mu_\theta(x) + \sigma_\theta(x) \odot \epsilon, \text{ onde } \epsilon \sim \mathcal{N}(0, I) \tag{3.13}$$

Onde $\odot$ denota a multiplicação elemento a elemento de um vetor. Desta forma, z é derivável em relação a θ, permitindo o cálculo do gradiente e a retropropagação do modelo, ainda sendo uma amostra da distribuição desejada $Q(z \mid x; \theta)$.

O segundo desafio reside em como calcular os dois termos da perda. O Termo de Reconstrução ($E_{z\sim Q}[\log P_{\text{model}}(x \mid z; \phi)]$), como visto anteriormente, é implementado via amostragem. Na prática, ele se torna a Log-Verossimilhança Negativa da entrada x dada a saída x^* do decodificador, que corresponde a uma função de perda como a Entropia Cruzada (para dados binários) ou o Erro Quadrático Médio (para dados Gaussianos).

O Termo de Regularização ($D_{KL}(Q(z \mid x; \theta) \parallel P_{\text{model}}(z; \phi))$) também precisa ser tratável. Assumindo a prior $P_{\text{model}}(z; \phi)$ como uma Gaussiana padrão $\mathcal{N}(0, I)$ e o codificador $Q(z \mid x; \theta)$ como uma Gaussiana com covariância diagonal, esta DKL possui uma solução analítica fechada. Ou seja, é possível calculá-la diretamente a partir das saídas μ_θ e σ_θ do codificador para cada dimensão j do espaço latente (de tamanho J):

$$D_{KL}(Q(z \mid x; \theta) \parallel \mathcal{N}(0, I)) = \frac{1}{2} \sum_{j=1}^{J} \left(\mu_j^2 + \sigma_j^2 - \log(\sigma_j^2) - 1 \right) \tag{3.14}$$

Combinando as implementações práticas dos dois termos, a função de perda final (Negativo do ELBO) a ser minimizada pelo VAE é:

$$L(\theta, \phi) = \underbrace{L_{\text{Reconstrução}}(x, x^*)}_{\text{BCE ou MSE}} + \underbrace{\frac{1}{2} \sum_{j=1}^{J} \left(\mu_j^2 + \sigma_j^2 - \log(\sigma_j^2) - 1 \right)}_{\text{Termo de Regularização (DKL)}} \tag{3.15}$$

Onde x^* é a saída do decodificador $P_{\text{model}}(x \mid z; \phi)$ após o z ser obtido pelo truque da reparametrização.

O fluxo de dados da arquitetura durante o treinamento, ilustrado na Figura 7, é o seguinte:

1. A entrada x é passada pelo codificador $Q(z \mid x; \theta)$, que produz os vetores μ e σ.
2. Uma amostra $\epsilon \sim \mathcal{N}(0, I)$ é gerada.
3. O vetor latente z é criado pelo Truque da Reparametrização: $z = \mu + \sigma \odot \epsilon$.
4. O vetor z é passado pelo decodificador $P(x \mid z; \phi)$, que produz a reconstrução x^*.

5. A perda de reconstrução é calculada.
6. A perda DKL é calculada diretamente a partir de μ e σ.
7. A perda total é retropropagada para atualizar ϕ e θ.

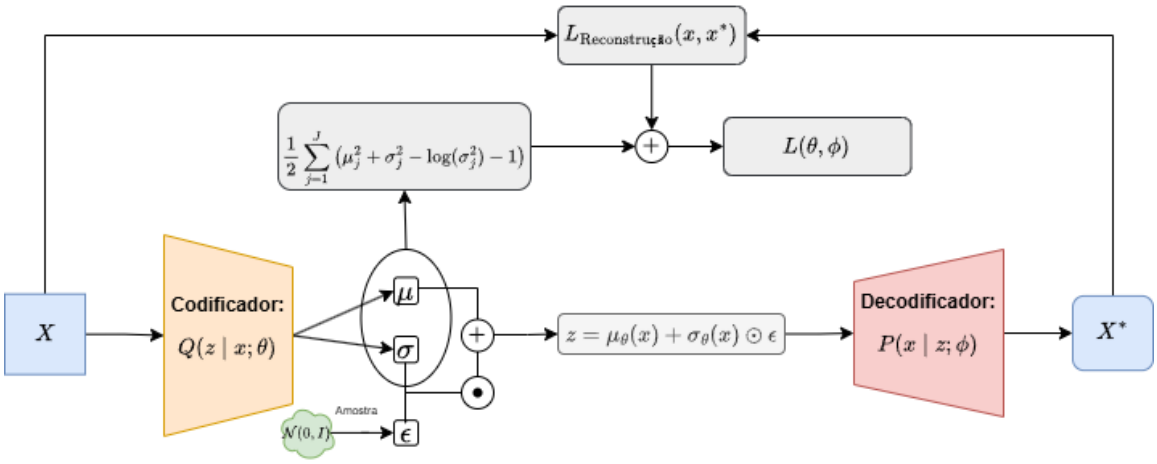


Figura 7 – Arquitetura do Autocodificador Variacional completa. Fonte: próprio autor.

3.3 Aplicações Do Autocodificador Variacional

Os Autocodificadores Variacionais apresentam uma grande versatilidade, com aplicações em múltiplos domínios que se beneficiam de sua capacidade de aprender representações latentes probabilísticas e estruturadas. Uma distinção em relação a outros modelos generativos é que o VAE, além de sua capacidade de geração de dados, impõe uma organização contínua ao espaço latente. Essa característica habilita seu uso em tarefas como a detecção de anomalias, e o modelo serve como base para o desenvolvimento de modelos mais recentes, como os modelos de difusão. A seguir, são exploradas algumas dessas aplicações.

3.3.1 Geração de Dados

Como um modelo generativo, a aplicação primária do VAE é a síntese de dados. O treinamento otimiza o decodificador $P(x \mid z; \phi)$ para que ele se torne um mapeamento da distribuição a priori $P(z)$ para a distribuição dos dados $P_{data}(x)$. Isso permite que novas amostras sejam geradas ao passar vetores z, amostrados de uma distribuição $\mathcal{N}(0, I)$, através da rede neural do decodificador [20]. Uma consequência da continuidade deste espaço latente é a capacidade de interpolar entre amostras de dados distintas. Ao decodificar a trajetória linear entre os vetores latentes z_a e z_b, correspondentes a duas amostras de dados x_a e x_b, obtém-se uma transição suave entre as amostras de dados, o que permite a exploração da variedade de dados aprendidas pelo modelo [67].

3.3.2 Componente em Modelos de Difusão Latente

Uma das aplicações recentes dos VAEs é servir como um componente para os Modelos de Difusão Latente (LDMs, do inglês *Latent Diffusion Models*) [68]. Os processos de difusão são computacionalmente custosos quando aplicados diretamente no espaço de pixels de alta dimensão. Para diminuir a complexidade computacional, os LDMs utilizam um VAE pré-treinado como um compressor perceptual. O codificador do VAE mapeia a imagem de alta resolução para um espaço latente compacto. O processo de difusão ocorre inteiramente nesse espaço latente de baixa dimensão, diminuindo o custo computacional. Ao final, o decodificador do VAE é usado para mapear a representação latente gerada de volta ao espaço de pixels, permitindo a síntese de imagens de alta fidelidade, por conta do modelo de difusão, com custo computacional reduzido.

3.3.3 Detecção de Anomalias

A capacidade do VAE de modelar a distribuição dos dados de treinamento $P_{data}(x)$ faz dele uma ferramenta eficaz para a detecção de anomalias [69, 70, 71]. A abordagem consiste em treinar o VAE exclusivamente em dados normais. O modelo aprende a comprimir e reconstruir eficientemente apenas as amostras que se assemelham à distribuição normal. Quando uma amostra anômala é apresentada, o modelo falha em duas frentes: o codificador não consegue mapear a entrada para uma região provável do espaço latente, resultando em alta DKL, e o decodificador falha em reconstruí-la com precisão, resultando em alto erro de reconstrução.

4 TRANSFORMADORES

O Transformador (*Transformer*), proposto por Vaswani *et al.* [2], é uma arquitetura de rede neural criada inicialmente para problemas de sequência a sequência, como a tradução automática. Esses problemas consistem em transformar uma sequência de entrada em uma nova sequência de saída, cujos comprimentos não precisam ser iguais [72]. A estrutura do Transformador revolucionou o campo de Processamento de Linguagem Natural (PLN) e se tornou a base para outros modelos de Aprendizado Profundo modernos em diversas áreas [73].

Antes do Transformador, a abordagem padrão utilizada para tarefas sequenciais era baseada na Rede Neural Recorrente (RNN, do inglês *Recurrent Neural Network*) e suas variantes, como LSTM (*Long Short-Term Memory*) e GRU (*Gated Recurrent Unit*) [74]. Essas arquiteturas processam os dados de forma sequencial, um elemento de cada vez, mantendo um estado interno que serve como memória do que foi visto até aquele ponto. Assim, o estado de um passo depende diretamente do resultado do passo anterior [75].

Na abordagem recorrente, o processamento é inerentemente sequencial. A informação flui passo a passo através da rede, o que cria um gargalo computacional que impede o paralelismo, gerando problemas de escalabilidade e eficiência [74]. Além disso, essa estrutura torna difícil para o modelo aprender relações entre elementos que estão muito distantes um do outro na sequência [76].

No Transformador, em contrapartida, o processamento é paralelo. A arquitetura abandona a recorrência e introduz um mecanismo chamado autoatenção (*self-attention*). A autoatenção permite que o modelo avalie a importância de todos os elementos em uma sequência simultaneamente, independentemente de sua posição. Ele calcula uma representação contextual para cada elemento com base em uma visão global dos dados de entrada [2].

4.1 Mecanismos do Transformador

Nesta seção, serão definidos os mecanismos do Transformador, conforme apresentado na Figura 8.

4.1.1 Camada de Incorporação

O processamento de dados no Transformador inicia-se pela Camada de Incorporação (*Embedding Layer*) [2]. Este componente tem a função de converter a sequência de entrada em uma representação vetorial, conhecida como incorporação (*embedding*). O

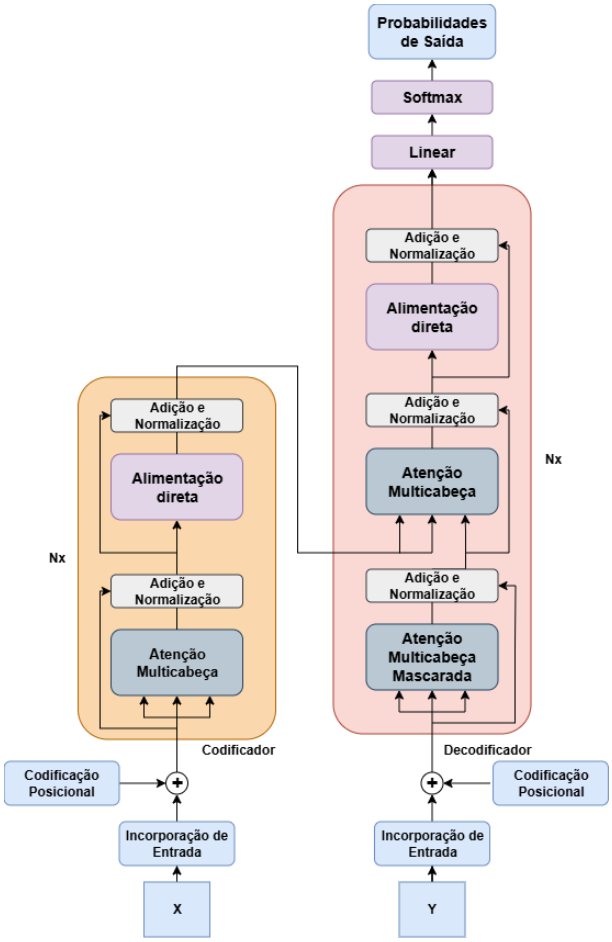


Figura 8 – Arquitetura do Transformador. Fonte: adaptado de Vaswani *et al.* [2].

Transformador exige que todo o processamento subsequente, como o cálculo da autoatenção e as redes de alimentação direta, opere sobre esses vetores, que devem possuir uma dimensão constante d_{model}. Esta camada funciona como um mapeamento aprendível, cuja implementação depende da natureza dos dados de entrada:

- Para dados categóricos, a camada é implementada como uma tabela de consulta. Esta tabela é uma matriz onde cada linha é um vetor de dimensão d_{model} e existe um vetor de dimensão d_{model} para cada tipo de dado categórico possível. O tipo de dado categórico é transformado em um índice, e a camada utiliza o índice da categoria de entrada para adquirir o vetor numérico correspondente [77].

- Para dados vetoriais contínuos, a camada é implementada por meio de uma projeção linear que mapeia a dimensão de entrada original para a dimensão d_{model} [78].

Esta etapa de projeção cumpre duas funções essenciais para o funcionamento do modelo. Primeiramente, ela garante a unificação dimensional, tornando os dados de entrada compatíveis com as operações matriciais da arquitetura. Em segundo lugar, ela

permite que o modelo aprenda, durante o treinamento, uma representação inicial dos dados que seja otimizada para a tarefa.

4.1.2 Codificação Posicional

O mecanismo de autoatenção processa os dados de entrada como um conjunto de vetores, sem considerar a ordem sequencial em que aparecem. Ou seja, se os vetores de entrada fossem embaralhados, a saída da camada de autoatenção seria correspondentemente embaralhada, mas os valores de saída em si seriam idênticos [2].

Para a maioria das tarefas sequenciais, como o processamento de linguagem ou séries temporais, a ordem dos elementos é fundamental. Para que o modelo utilize essa informação, é necessário injetar dados sobre a posição de cada elemento.

A arquitetura do Transformador utiliza a Codificação Posicional (*Positional Encoding*) para resolver esse problema. Este componente gera um vetor de dimensão d_{model} para cada posição na sequência. Este vetor posicional é calculado utilizando uma função fixa específica, que permite o modelo aprender a entender as posições relativas dos elementos da sequência. No artigo original do Transformador [2], os autores propuseram o uso de funções seno e cosseno de diferentes frequências:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{4.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{4.2}$$

Onde pos é a posição do elemento na sequência e i é o índice da dimensão dentro do vetor de incorporação (de 0 a $d_{\text{model}}/2 - 1$).

Este vetor de codificação posicional é somado ao vetor de *embedding*. O resultado desta soma é a entrada final para o primeiro bloco do Transformador, combinando assim a informação da *embedding* da sequência de entrada com a informação de ordem, dada pela codificação posicional [2].

4.1.3 Atenção

Após a soma da incorporação com a codificação posicional, a sequência de vetores X (onde $X \in R^{T \times d_{\text{model}}}$, sendo T o comprimento da sequência) está pronta para ser processada pelo componente central do Transformador: o mecanismo de Atenção.

A operação de Atenção baseia-se na interação entre três tipos de vetores, gerados a partir de cada vetor de entrada x_i (que possui dimensão d_{model}):

$$q_i = x_i W_Q, \quad k_i = x_i W_K, \quad v_i = x_i W_V$$

Onde $W_Q \in R^{d_{\text{model}} \times d_k}$, $W_K \in R^{d_{\text{model}} \times d_k}$ e $W_V \in R^{d_{\text{model}} \times d_v}$ são matrizes de pesos aprendíveis.

A intuição sobre esses vetores é análoga a um sistema de recuperação de informação [2]:

- **Consulta (q_i):** Atua como um vetor de busca. Ele representa o elemento i no contexto de uma procura por relevância nos demais elementos da sequência.
- **Chave (k_i):** Atua como o vetor de identificação. Ele representa o atributo do elemento i que será comparado por todas as consultas (q_j) da sequência para determinar pontuação de relevância entre os elementos i e j.
- **Valor (v_i):** Atua como o vetor de conteúdo. Ele representa a informação do elemento i que será agregada na saída final, com base na pontuação de relevância.

O mecanismo de atenção utiliza esses três vetores para calcular uma nova representação, z_i, para cada elemento de entrada x_i. Esta nova representação z_i é uma agregação de informações de toda a sequência, ponderada pela relevância.

O processo ocorre em três etapas para cada elemento i [2]:

1. **Cálculo de pontuação:** A Consulta q_i é usada para calcular uma pontuação de relevância com todas as Chaves k_j da sequência. Esta pontuação e_{ij} mede o quão relevante o elemento j é para a consulta do elemento i.
2. **Normalização de pesos:** As pontuações $e_{i1}, e_{i2}, \ldots, e_{iT}$ são normalizadas por meio de uma função de normalização, geralmente a *softmax*. Isso converte as pontuações em um conjunto de pesos de atenção α_{ij}, que são positivos e somam 1. O peso α_{ij} é uma maneira de quantificar a atenção que o elemento i deve dar ao elemento j.
3. **Soma ponderada:** O vetor de saída z_i é calculado como a soma de todos os vetores de Valor v_j da sequência, sendo cada v_j multiplicado pelo seu respectivo peso de atenção α_{ij}.

O vetor de saída z_i é uma mistura de todos os valores da sequência, onde os valores considerados mais relevantes, identificados por valores com maior peso α_{ij}, contribuem mais para a representação final de z_i. Esse vetor pode ser visto como a representação contextualizada do elemento i.

Para o processamento paralelo e eficiente das operações, esses vetores são processados em formato matricial. Os T vetores de entrada x_i são agrupados na matriz X. Esta matriz é projetada linearmente pelas mesmas três matrizes de pesos aprendíveis (W_Q, W_K, W_V) para criar as matrizes de Consulta (Q, do inglês *Query*), Chave (K, do inglês *Key*) e Valor (V, do inglês *Value*):

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

Nesta formulação, Q, K, V são as matrizes de dimensão $T \times d_k$. Onde a i-ésima linha da matriz Q é o vetor q_i, a i-ésima linha de K é o vetor k_i e a i-ésima linha da matriz V é o vetor v_i

4.1.3.1 Atenção por Produto Escalar com Escala

O mecanismo de atenção utilizado pelo Transformador é a Atenção por Produto Escalar com Escala (*Scaled Dot-Product Attention*) [2]. Esta operação combina as três etapas (pontuação, normalização, soma ponderada) em uma única expressão:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.3}$$

Analisando a equação 4.3 [2]:

- QK^T: Calcula a pontuação de todos elementos da sequência em uma operação de produto escalar. O resultado é uma matriz $T \times T$ onde a célula (i, j) contém o produto escalar $q_i \cdot k_j$.
- $\sqrt{d_k}$: Este é o fator de escala. A divisão por este fator é feita para dar estabilidade no treinamento. Sem ela, o produto escalar QK^T pode gerar valores grandes, saturando a função *softmax* e anulando os gradientes.
- $\text{softmax}(\left(\frac{QK^T}{\sqrt{d_k}}\right))$: Aplica a normalização de pesos em cada linha da matriz de pontuação, fazendo a soma dos pesos dessa linha ser igual a 1.
- $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$: Executa a soma ponderada dos pesos com os conteúdos de V, multiplicando a matriz de pesos de atenção pela matriz de V.

4.1.3.2 Atenção Multi-Cabeça

O Transformador aprimora este mecanismo através da Atenção Multi-Cabeça (MHA, do inglês *Multi-Head Attention*). Em vez de executar uma única função de atenção com vetores de dimensão d_{model}, o MHA executa h funções de atenção de dimensão $d_{\text{head}} = d_{\text{model}}/h$, chamadas "cabeças".

Para isso, o modelo utiliza h matrizes de projeção aprendíveis distintas (W_Q^i, W_K^i, W_V^i), um para cada cabeça i. Cada conjunto de matrizes projeta a entrada (X) para dimensões menores $(d_{\text{head}} = d_{\text{model}}/h)$, criando as matrizes Q_i, K_i, V_i específicas para cada cabeça [2]:

$$Q_i = XW_Q^i, \quad K_i = XW_K^i, \quad V_i = XW_V^i$$

A Atenção por Produto Escalar com Escala (Equação 4.3) é aplicada a cada uma dessas h cabeças:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$$

As saídas das h cabeças (cada uma com dimensão $T \times d_{\text{head}}$) são então concatenadas de volta em uma única matriz. Esta concatenação produz uma matriz de dimensão $T \times (h \cdot d_{\text{head}})$, que é equivalente à dimensão $T \times d_{\text{model}}$.

Esta matriz concatenada passa por uma projeção linear final, parametrizada por uma matriz de pesos $W^O \in R^{d_{\text{model}} \times d_{\text{model}}}$, para produzir a saída final da camada MHA. Esta saída é o resultado que será passado para a próxima subcamada, a de adição e normalização.

A vantagem do MHA é que cada uma das h cabeças aprende um conjunto distinto de matrizes de projeção (W_Q^i, W_K^i, W_V^i). Isso permite que cada cabeça se especialize em capturar diferentes tipos de informação na sequência. Por exemplo, uma cabeça pode focar em dependências de curta distância, enquanto outra foca em dependências de longa distância [2].

4.1.4 Rede de Alimentação Direta

Após o processamento pela camada de MHA, a arquitetura do Transformador aplica uma segunda camada: a Rede de Alimentação Direta (FFN, do inglês *Feed-Forward Network*).

A FFN é uma rede neural simples, composta por duas transformações lineares, implementadas por meio de duas camadas densas, com uma função de ativação não linear entre elas, tipicamente a Unidade Linear Retificada [2]:

$$\text{FFN}(x) = \text{ReLU}(0, xW_1 + b_1)W_2 + b_2 \tag{4.4}$$

Inicialmente, a camada expande a dimensionalidade da representação de d_{model} para uma dimensão interna da FFN d_{ff}. Na sequência, ela comprime de volta para o tamanho original d_{model}. A FFN permite a extração de características e a introdução de não linearidade no modelo, operando sobre a representação contextualizada fornecida pelo mecanismo de atenção.

4.1.5 Adição e Normalização

Para permitir o treinamento de uma arquitetura com múltiplas camadas, o Transformador emprega duas técnicas em torno de cada uma das camadas descritas anteriormente: conexões residuais (adição) e normalização de camada [2].

Cada camada $f(x)$ é envolvida por esta operação, definida como:

$$\text{Saída} = \text{LayerNorm}(x + f(x)) \tag{4.5}$$

Esse processo ocorre em duas etapas:

1. **Adição (Conexão residual):** A entrada x da subcamada é somada diretamente à saída $f(x)$ da subcamada. Esta técnica, conhecida como conexão residual [79]. Ela mitiga o problema do desaparecimento do gradiente, um problema que faz o modelo não aprender pelo gradiente calculado ficar próximo de 0. Esse problema aparece em arquiteturas muito profundas, porém esta técnica resolve isso, permitindo que o modelo aprenda arquiteturas muito mais profundas.

2. **Normalização de camada:** O resultado da soma $(x + f(x))$ passa por uma Normalização de Camada (*Layer Normalization*) [80]. Esta operação calcula a média e o desvio padrão dos valores de camada para cada amostra e normaliza esses valores, deixando-os com média 0 e variância 1. Isso ajuda a deixar o treino do modelo mais estável e a acelerar sua convergência.

4.2 Montagem da Arquitetura

Com os mecanismos do Transformador estabelecidos, é possível montar os blocos principais da arquitetura do Transformador, ilustrados pela Figura 8.

4.2.1 Codificador do Transformador

O Codificador do Transformador é a parte da arquitetura responsável por processar a sequência de entrada e gerar uma representação contextualizada dela. Ele é uma pilha de N blocos idênticos. Cada bloco do Codificador é composto por duas subcamadas principais, com as operações de Adição e Normalização aplicadas a cada uma [2]:

1. **Subcamada 1: MHA (Autoatenção):**

   - Primeiro, o bloco aplica o mecanismo de Atenção Multi-Cabeça.
   - Neste contexto, esta operação é uma autoatenção, pois as matrizes Q, K e V são todas derivadas da mesma entrada: a saída do bloco anterior. No caso do primeiro bloco, a entrada consiste na soma da codificação posicional com a entrada incorporada.
   - A saída desta subcamada é processada pela operação de Adição e Normalização:

     $$\text{Subcamada}_1 = \text{LayerNorm}(x + \text{MHA}(x))$$

     .

2. **Subcamada 2: Rede de Alimentação Direta:**

   - A saída da primeira subcamada é então passada para a Rede de Alimentação Direta (FFN).
   - A FFN processa cada posição de forma independente.
   - A saída da FFN passa pela segunda operação de Adição e Normalização:

$$Z = \text{LayerNorm}(\text{Subcamada}_1 + \text{FFN}(\text{Subcamada}_1))$$

.

A saída do bloco do Codificador i torna-se a entrada para o Bloco Codificador $i+1$. A saída da pilha inteira de N codificadores é uma representação vetorial Z da sequência de entrada, que captura as relações contextuais entre os dados da sequência de entrada. A saída Z será então utilizada pelo Decodificador do Transformador [2].

4.2.2 O Bloco Decodificador

O Decodificador do Transformador tem a função de gerar a sequência de saída um elemento por vez, no caso da inferência. Assim como o Codificador, ele é uma pilha de N blocos idênticos.

A estrutura do Bloco Decodificador é semelhante à do Codificador, a diferença é que ele possui três subcamadas, com as subcamadas de atenção modificadas. A entrada para o Decodificador é a sequência de saída gerada até o momento, que passa pela mesma Camada de Incorporação e Codificação Posicional do Codificador. Além disso, a forma como ele recebe dados de entrada difere entre o treinamento e a inferência:

- **Durante o treinamento (Paralelo):** Para um treinamento eficiente, o decodificador recebe a sequência de saída completa, denotada por Y na figura 8, de uma só vez. A sequência é deslocada para a direita para que a entrada na posição i seja o elemento y_{i-1}, e o modelo aprenda a prever y_i. A previsão é feita em paralelo para as todas as posições.
- **Durante a inferência (Autoregressiva):** O processo é sequencial, sendo produzido dado por dado. O modelo começa com um elemento que representa o início da sequência. A saída gerada y_1 é, então, alimentada de volta como entrada para o próximo passo, gerando y_2. O processo se repete até que um elemento que representa o fim da sequência seja produzido.

As subcamadas do Bloco Decodificador, onde X_{dec} é a entrada para o bloco, são projetadas para lidar com ambos os cenários [2]:

1. **Subcamada 1: Atenção Multi-Cabeça Mascarada (Autoatenção)**

   - O Decodificador primeiro aplica uma Atenção Multi-Cabeça em sua própria entrada (a sequência de saída do decodificador, ou elemento de entrada).
   - Para que o modelo de treinamento (paralelo) simule o comportamento da inferência (sequencial), é aplicado um mascaramento antes da função $softmax$. Esta máscara faz com que os pesos das posições futuras sejam zeradas. Isso impede que o cálculo de atenção para a posição i veja as posições futuras ($j > i$), garantindo um comportamento similar ao da inferência. Esse tipo de atenção é chamada Atenção Multi-Cabeça Mascarada, denotada como função por MaskedMHA.
   - A saída é processada por Adição e Normalização:

$$\text{Subcamada}_1 = \text{LayerNorm}(X_{\text{dec}} + \text{MaskedMHA}(X_{\text{dec}}))$$

2. **Subcamada 2: Atenção Cruzada (Codificador-Decodificador)**

   - Esta é a subcamada que conecta o Decodificador ao Codificador.
   - Ela implementa a Atenção Cruzada. A origem dos vetores Q, K e V utilizados para essa função é diferente:
     - A matriz de Consulta (Q) vem da subcamada anterior do Decodificador.
     - As matrizes de Chave (K) e Valor (V) vêm da saída final da pilha de Codificadores (a matriz Z contextualizada).
   - Isso permite que cada elemento no Decodificador consulte toda a sequência de entrada, dada pela matriz Z, para decidir qual informação é relevante para gerar o próximo elemento da saída.
   - A saída é processada por Adição e Normalização:

$$\text{Subcamada}_2 = \text{LayerNorm}(\text{Subcamada}_1 + \text{MHA}(Q = \text{Subcamada}_1, K = Z, V = Z))$$

3. **Subcamada 3: Rede de Alimentação Direta**

   - A saída da terceira subcamada é então passada para a Rede de Alimentação Direta.
   - A FFN processa cada posição de forma independente.
   - A saída da FFN passa pela terceira operação de Adição e Normalização:

$$\text{Subcamada}_3 = \text{LayerNorm}(\text{Subcamada}_2 + \text{FFN}(\text{Subcamada}_2))$$

4.2.3 Camada de Saída Final

Após a pilha de N blocos do Decodificador, a saída final (SubCamada$_3$ do último bloco) é processada para gerar a probabilidade do próximo elemento que será gerado [2]:

1. **Camada linear:** Uma camada linear projeta o vetor de saída de dimensão d_{model} para a dimensão do vocabulário. Essa dimensão final corresponde ao número total de elementos possíveis que o modelo pode prever.

2. **Função softmax:** A função $softmax$ é aplicada à saída da camada linear, convertendo-a em uma distribuição de probabilidade. O elemento com a maior probabilidade é selecionado como a previsão do modelo para aquele passo.

4.3 Aplicações do Transformador

Neste capítulo a arquitetura do Transformador foi apresentada conforme o trabalho original [2], como uma pilha completa de Codificador-Decodificador para tarefas de sequência-a-sequência, como a tradução. Contudo, a flexibilidade e a eficácia de seus componentes de captura e representaçào de contexto, com o mecanismo de atenção, levaram a adaptações da arquitetura [81].

Modelos subsequentes em diversas áreas passaram a utilizar apenas a pilha de Codificadores, apenas a pilha de Decodificadores ou combinações híbridas de arquiteturas. Esta modularidade permitiu que o Transformador se tornasse a base para múltiplas áreas, que vão muito além de sua aplicação inicial [81, 82].

4.3.1 Processamento de Linguagem Natural

O Processamento de Linguagem Natural foi o domínio onde o Transformador causou a revolução inicial. Inicialmente, modelos baseados na arquitetura completa Codificador-Decodificador, como o T5 (*Text-to-Text Transfer Transformer*) [83] e o BART (*Bidirectional and Auto-Regressive Transformers*) [84], refinaram tarefas de sequência-a-sequência, como sumarização de texto e tradução.

As arquiteturas parciais mudaram a necessidade de utilizar a arquitetura completa, demonstrando grande potencial:

- **Arquiteturas utilizando apenas Codificador:** Modelos como o BERT (*Bidirectional Encoder Representations from Transformers*) [85] utilizam apenas a pilha de Codificadores. Ao processar a sequência de entrada inteira de forma bidirecional, o BERT aprendeu representações contextuais profundas, revolucionando tarefas de compreensão de linguagem, como classificação de sentimento e reconhecimento de entidades nomeadas.

- **Arquiteturas utilizando apenas Decodificador:** Mais recentemente, os Grandes Modelos de Linguagem (LLM, do inglês *Large Language Model*), como a série GPT (*Generative Pre-trained Transformer*) [86], popularizaram o uso de arquiteturas de apenas Decodificador. Ao utilizar apenas a pilha de Decodificadores, esses modelos são autorregressivos e se tornaram o padrão para tarefas de geração de linguagem, como completude de texto, diálogo e escrita criativa.

4.3.2 Visão Computacional

O Transformador de Visão (ViT, do inglês *Vision Transformer*) [87] demonstrou que o Transformador pode ser aplicado diretamente à classificação de imagens, alcançando melhores resultados do que as Redes Neurais Convolucionais. O ViT trata a imagem como uma sequência. Ele divide a imagem em pequenas sub-regiões quadradas, aplica uma projeção linear a cada sub-região e adiciona informações posicionais. Esta sequência de vetores é processada por uma pilha de Codificadores do Transformador, que utiliza a autoatenção para encontrar relações entre diferentes partes da imagem.

4.3.3 Detecção de Anomalias em Séries Temporais

A capacidade do mecanismo de autoatenção de capturar dependências complexas e de longo prazo, independentemente da distância entre os pontos, tornou o Transformador uma ferramenta poderosa para a análise de Séries Temporais.

No contexto da detecção de anomalias, particularmente em dados multivariados como tráfego de rede ou telemetria industrial, arquiteturas baseadas no Transformador são utilizadas [82]. Modelos utilizando apenas Codificadores ou híbridos com Autocodificadores (como o VAE) são treinados em dados normais para aprender os padrões subjacentes da sequência.

O modelo aprende a reconstruir o comportamento normal da série. Quando uma amostra anômala é apresentada, a autoatenção falha em encontrar padrões familiares e o modelo é incapaz de reconstruí-la com precisão. Isso resulta em um alto erro de reconstrução, que serve como um indicador eficaz de que um comportamento anômalo foi detectado.

5 TRABALHOS RELACIONADOS

A literatura acadêmica recente mostra que a detecção de anomalias em redes utilizando Aprendizado Profundo é um campo de estudo proeminente [11, 5, 32, 33]. Dois tipos de modelos de Aprendizado Profundo têm sido aplicados neste contexto: os Transformadores [17] e os Autocodificadores Variacionais [88, 89]. Este capítulo irá abordar trabalhos que utilizam essa abordagem.

Um trabalho que integra ambas as arquiteturas é o T-VAE (*Transformer-Based Variational Autoencoder*), proposto por Li *et al.* (2025) [90]. O modelo é projetado para a percepção de anomalias em dados de séries temporais com múltiplas variáveis, semelhantes aos dados utilizados neste trabalho. A arquitetura T-VAE consiste em duas sub-redes otimizadas de forma conjunta: uma Rede de Representação e uma Rede de Memória. A Rede de Representação utiliza mecanismos de autoatenção, baseados nos Transformadores, para capturar a informação sequencial dos dados. A Rede de Memória emprega um Autocodificador Variacional para modelar distribuição dos dados normais.

Wu *et al.* [18] propuseram um sistema supervisionado robusto baseado em Transformadores, utilizando um mecanismo de autoatenção para detectar intrusões na rede. O sistema, chamado RTIDS (*Robust Transformer-based Intrusion Detection System*), utiliza uma arquitetura de Transformador codificador-decodificador para comprimir os dados de fluxo brutos de alta dimensão em representações latentes de baixa dimensão. Os autores avaliaram o desempenho do RTIDS nos conjuntos de dados *CIC-IDS2017* e *CIC-DDoS*2019. Eles compararam seu modelo com algoritmos de aprendizado de máquina como SVM (*Support Vector Machine*) e modelos de Aprendizado Profundo, incluindo RNN, FNN e LSTM. O trabalho relata que o RTIDS obteve um desempenho superior para os conjuntos de dados testados, alcançando um *F1-score* de $99,17\%$ para o *CIC-IDS2017* e $99,48\%$ para o *CIC-DDoS*2019.

Outro trabalho que utiliza o transformador como base no formato supervisionado é o de Manocchio *et al.* [91, 92], que introduz o FlowTransformer, um framework para implementar e avaliar sistematicamente sistemas de detecção de intrusão de rede. A estrutura permite a substituição modular de componentes-chave, como a codificação de entrada, o tipo de transformador e o cabeçalho de classificação, para testar diferentes arquiteturas. Os autores o utilizaram para avaliar arquiteturas como GPT 2.0 e BERT em três conjuntos de dados públicos: *CSE-CIC-IDS*, NSL-KDD e *UNSW-NB15*. Uma das principais contribuições do trabalho foi que a escolha do cabeçalho de classificação tem um impacto significativo no desempenho dos modelos com Transformador.

Para resolver as limitações de métodos NIDS que são puramente generativos ou

discriminativos, Wang *et al.* [93] propuseram o *MTRC*, um modelo auto-supervisionado. O sistema utiliza múltiplos Transformadores para criar múltiplas reconstruções dos dados de rede e aplica aprendizado contrastivo para aprender representações do tráfego normal. Essa técnica força o modelo a capturar características latentes dos dados normais, e a perda contrastiva funciona como pontuação de anomalia. O *MTRC* foi validado nos conjuntos de dados *CIC-IDS2017* e *UNSW-NB15*, obtendo um *F1-score* de $98,94\%$ e $96,51\%$, respectivamente.

Akkepalli e K [94] propuseram uma arquitetura NIDS com dois componentes principais: um Transformador regularizado por *Copula Entropy* com C^2 VAE para processamento de características e um modelo híbrido de Aprendizado Profundo para classificação de ataques. O primeiro componente visa extrair e selecionar características de forma robusta e eficiente, enquanto o segundo detecta ambos ataques conhecidos e *zero-day*. A avaliação experimental do modelo no conjuntos de dados *NID* demonstrou as seguintes métricas: acurácia $98,54\%$, *F1-score* $97,50\%$, revocação $97,30\%$, precisão $98,2\%$, taxa de detecção $97,50\%$.

O trabalho de Wang *et al.* (2023) [95] investigou a detecção de intrusão não supervisionada, focando na captura do contexto temporal dos dados de rede. O propósito central foi desenvolver um sistema que aprendesse representações robustas, menos sensíveis a dados anômalos infiltrados. A metodologia emprega o RUIDS (*Robust Unsupervised Intrusion Detection System*), que utiliza um módulo de reconstrução de contexto mascarado, uma ideia similar ao BERT [85]. Esta técnica ensina o modelo a reconstruir dados, tornando-o menos sensível a amostras contaminadas. Os resultados nos experimentos comparativos demonstraram a eficácia da abordagem, alcançando um valor de AUC de $88,02\%$ no *UNSW-NB15* e $98,01\%$ no *CIC-IDS-WED*, valores superiores aos métodos de base avaliados no estudo.

O trabalho de Kummerow *et al.* (2024) [96] propôs o modelo não supervisionado T-NAE (*Transformer-Based Network Traffic Autoencoder*), um Autocodificador baseado em Transformer de dois estágios para análise de tráfego de rede. O objetivo foi aprender representações do tráfego em nível de pacote e de sequência. O primeiro estágio usa a atenção nas características para fundir características discretas e contínuas de cada pacote. O segundo estágio utiliza a autoatenção para modelar a sequência de pacotes. Treinado de forma não supervisionada, o modelo detecta anomalias usando o erro de reconstrução, alcançando um *F1-score* de $95,36\%$ e uma taxa de falso positivo de $4,53\%$ no *CIC-IDS2017*. A arquitetura também permite explicações por meio da perturbação de atenção, o que torna o modelo mais compreensível.

Para superar as limitações dos NIDS baseados em fluxo e da necessidade de dados rotulados, Ghadermazi *et al.* [97] introduziram o GTAE-IDS (*Graph Transformer-Based Autoencoder Framework for Real-Time Intrusion Detection System*). O sistema gera dina-

micamente grafos a partir de sequências iniciais de pacotes e emprega um Autocodificador de Grafo com uma arquitetura de Transformador no codificador para modelar o tráfego normal. A detecção de anomalias baseia-se na capacidade do modelo de reconstruir os dados, onde anomalias apresentam maior erro de reconstrução ou desvios nas representações numéricas dos vetores do codificador. O sistema alcançou acurácia superior a $98,00\%$ nos conjuntos de dados *CIC-IDS2017/2018* e *ACI-IoT-2023* e demonstrou baixo tempo de inferência, superando abordagens estado da arte baseadas em grafos e pacotes.

O estudo de Vu *et al.* [98] apresentou o MVAE (*Multidistributed Variational Auto-Encoder*), um modelo focado em criar um espaço latente mais propício para a classificação de tráfego de rede. A metodologia modifica a perda D_{KL} do VAE para incorporar rótulos de classe, forçando uma separação explícita das distribuições no espaço latente. A eficácia do MVAE como extrator de características foi quantificada nos conjuntos de dados NSL-KDD e *UNSW-NB15*. No NSL-KDD, o classificador *Random Forest* obteve um AUC de $83,90\%$ nos dados originais, mas alcançou $94,30\%$ no espaço latente do MVAE. No *UNSW-NB15*, o incremento foi de $88,90\%$ para $96,10\%$.

Chen e Ye (2025) [99] abordaram a extração insuficiente de características na detecção de intrusão. O objetivo central do trabalho foi desenvolver um mecanismo de atenção aprimorado. A metodologia introduz a MSA-CBAM (*Multi-Scale Adaptive Convolutional Block Attention Module*). Esta arquitetura utiliza *pooling* multi-escala e ponderação adaptativa para fundir características de diferentes escalas. O MSA-CBAM foi integrado a um ConVAE (*Convolutional Variational Autoencoder*) e a um classificador CNN. O modelo final utiliza pré-treinamento não supervisionado seguido de ajuste de parâmetros supervisionado, foi avaliado no *CIC-IDS2017* e alcançou um *F1-score* de $93,70\%$.

A análise da literatura demonstra múltiplas abordagens que utilizam Autocodificadores Variacionais e Transformadores para a detecção de intrusão. O papel do VAE nesses trabalhos foca em aprender a distribuição dos dados para modelar o comportamento normal do tráfego. Os trabalhos também demonstram uma tendência em integrar mecanismos de atenção, com as arquiteturas de Transformadores ou suas variantes, com o objetivo de aprimorar a captura de dependências contextuais e temporais nos dados de tráfego de rede.

6 ESTUDO DE CASO

Este capítulo apresenta o estudo de caso conduzido para validar a arquitetura implementada para os dados testados. Foi implementado um Sistema de Detecção de Intrusão não supervisionado, que utiliza um codificador baseado na arquitetura Transformador e um decodificador implementado por um Perceptron de Múltiplas Camadas (MLP, do inglês *Multilayer Perceptron*). A avaliação deste modelo foi realizada sobre o conjunto de dados Orion [100], um conjunto de dados de tráfego de rede baseado em fluxo IP, proveniente do grupo de pesquisa Orion da Universidade Estadual de Londrina.

6.1 Conjunto de Dados

A eficácia dos modelos de Aprendizado Profundo está diretamente ligada à qualidade e representatividade dos dados utilizados em seu treinamento e validação. A obtenção de conjuntos de dados realistas e representativos é fundamental para o desenvolvimento de sistemas robustos. No entanto, o domínio de Redes Definidas por *Software* enfrenta uma escassez de conjuntos de dados públicos disponíveis [34], um desafio que dificulta a validação e a comparação de arquiteturas de Aprendizado Profundo.

O conjunto de dados Orion [100], utilizado neste trabalho, foi desenvolvido pelo grupo de pesquisa Orion da Universidade Estadual de Londrina. Para a sua criação, foi utilizada a ferramenta Mininet para emular uma topologia de rede contendo 6 *switches* e 140 *hosts*. O controlador SDN utilizado foi o Floodlight, e o tráfego de rede foi gerado utilizando a biblioteca Scapy. Os ataques simulados neste conjunto de dados foram o *DDoS* e o *Port Scan*.

O conjunto de dados é estruturado em três dias de captura de tráfego:

- **Dia 1:** Contém exclusivamente tráfego normal da rede, sem a presença de qualquer ataque, ilustrado pela Figura 9.
- **Dia 2:** Apresenta um ataque de *DDoS*, das 09:15:00 às 10:30:00, e um ataque de *Port Scan*, das 13:45:00 às 14:50:00, mostrado pela Figura 10.
- **Dia 3:** Contém um ataque de *Port Scan*, das 11:20:00 às 12:40:00, e um ataque de *DDoS*, das 16:35:00 às 17:50:00, exibido pela Figura 11.

O processamento dos dados brutos em características numéricas foi realizado em duas etapas. Primeiramente, o tráfego foi agregado em janelas temporais de um segundo, totalizando 86.400 registros (amostras) por dia. Para cada uma dessas janelas de um segundo, as características total de bits e total de pacotes foram calculadas pela soma direta

dos respectivos valores observados naquele intervalo. Adicionalmente, foram computadas as métricas de média de bits por segundo, média de pacotes por segundo e a média de duração dos fluxos.

Para as características categóricas (endereços IP e portas), foi aplicada a métrica da Entropia de Shannon [101]. Em termos formais, para uma variável categórica X que pode assumir valores $x_1, x_2, \ldots, x_n$ com probabilidades respectivas $p_1, p_2, \ldots, p_n$, a entropia é dada por:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{6.1}$$

onde:

- $H(X)$ é a entropia de Shannon da variável X;
- p_i é a probabilidade de ocorrência da categoria x_i;
- A base do logaritmo é 2 para que a entropia seja expressa em bits.

A entropia é uma medida fundamental neste problema, pois viabiliza a utilização de informações de IP e porta. Essas características, sendo categóricas e de alta cardinalidade (valores que podem ser únicos para sessões ou dias específicos), não poderiam ser utilizadas diretamente em sua forma qualitativa, pois prejudicariam a generalização do modelo. A entropia soluciona este problema ao converter o conjunto de valores qualitativos em um valor quantitativo de menor cardinalidade. A relevância desta medida é acentuada pelas particularidades do conjunto de dados Orion. Isso se deve ao fato que as características quantitativas, como Total de Bits e Total de Pacotes, mostraram-se pouco significativas para a aprendizagem nesse conjunto de dados, conforme apontado por Ruffo *et al.* (2025) [102].

O cálculo da entropia é realizado para quantificar a diversidade das ocorrências de IPs e portas dentro da mesma janela de um segundo, sendo as probabilidades das ocorrências de IPs e portas calculadas por aparição nesse segundo. Um valor de entropia elevado, por exemplo, indica que um grande número de IPs ou portas distintas foi observado naquele segundo, enquanto um valor baixo sugere concentração em poucos elementos, o que caracteriza uma baixa diversidade [54]. Nesse cálculo, a probabilidade p_i de cada item (IP ou porta) corresponde à sua frequência relativa dentro da janela de um segundo.

O conjunto de dados final resultante é composto por 9 características:

1. Entropia de IP de Destino;
2. Entropia de Porta de Destino;

3. Entropia de IP de Origem;
4. Entropia de Porta de Origem;
5. Total de Bits;
6. Total de Pacotes;
7. Média de Bits Por Segundo;
8. Média de Pacotes Por Segundo;
9. Média de Duração dos Fluxos.

6.1.1 Análise Exploratória de Dados

A Análise Exploratória de Dados, fundamentada nos gráficos das Figuras 9, 10 e 11, mostra um padrão para o ataque *Port Scan*. Uma inspeção visual demonstra que a Entropia de Porta de Destino é a única característica que não apresenta sobreposição substancial entre o tráfego normal (azul) e o anômalo (vermelho) em ambos os dias de ataque para o *Port Scan*. Esta separabilidade é esperada, pois o ataque *Port Scan* não altera significativamente o volume de tráfego, mas, por definição, testa múltiplas portas. Esta varredura de portas eleva a diversidade de portas de destino por segundo, o que resulta no aumento do valor da entropia.

A Entropia de Porta de Destino também é eficaz para a detecção do *DDoS*. Neste conjunto de dados, é especificado que todos os atacantes atacam o mesmo IP na mesma porta de destino. Essa ação de ataque concentrado em uma mesma porta reduz a diversidade de portas observadas por segundo, resultando em um valor de entropia anormalmente baixo durante o ataque.

Esses dois fatores motivaram a escolha de utilizar apenas a Entropia de Porta de Destino como característica para o modelo de Aprendizado Profundo implementado.

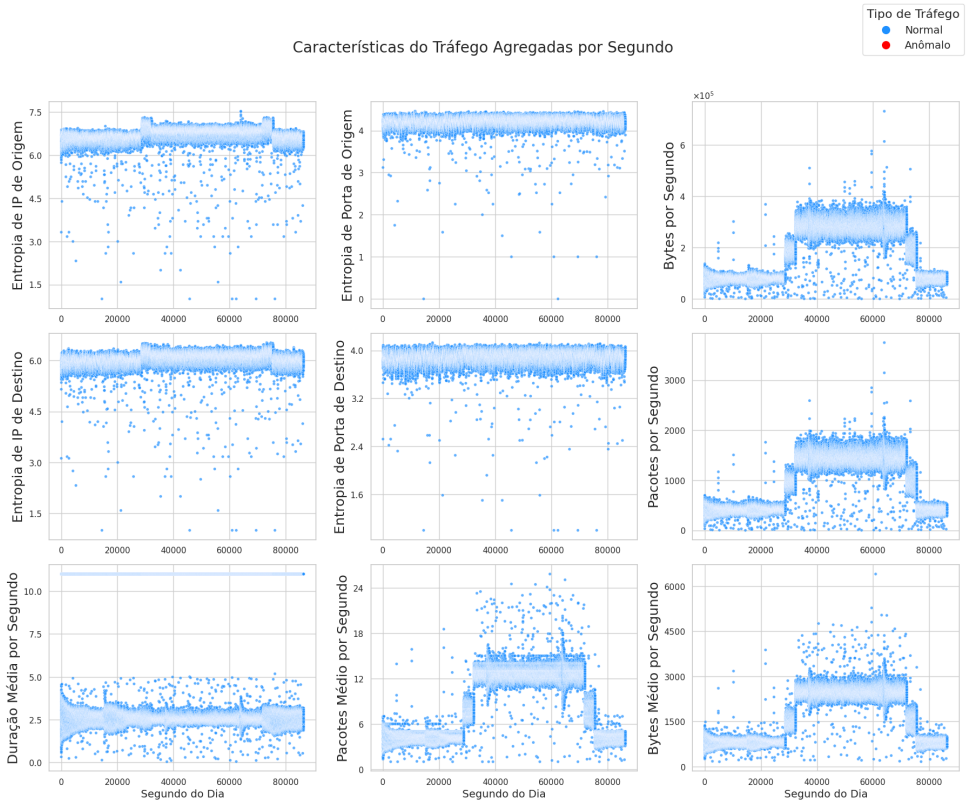


Figura 9 – Características do primeiro dia. Fonte: próprio autor.

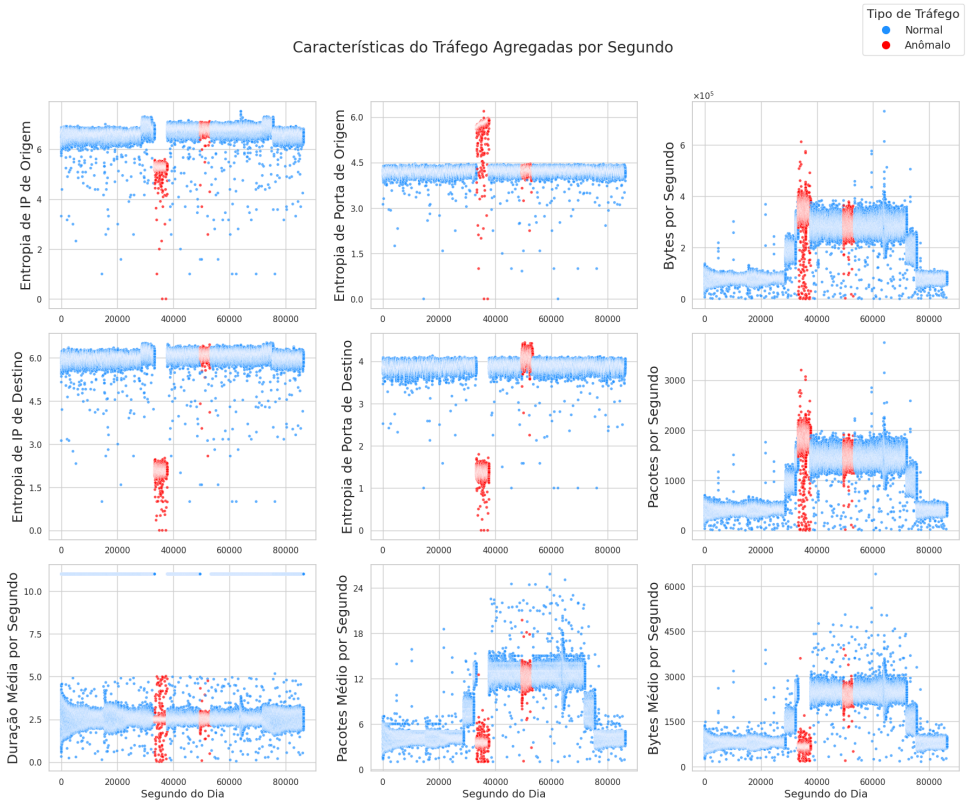


Figura 10 – Características do segundo dia. Fonte: próprio autor.

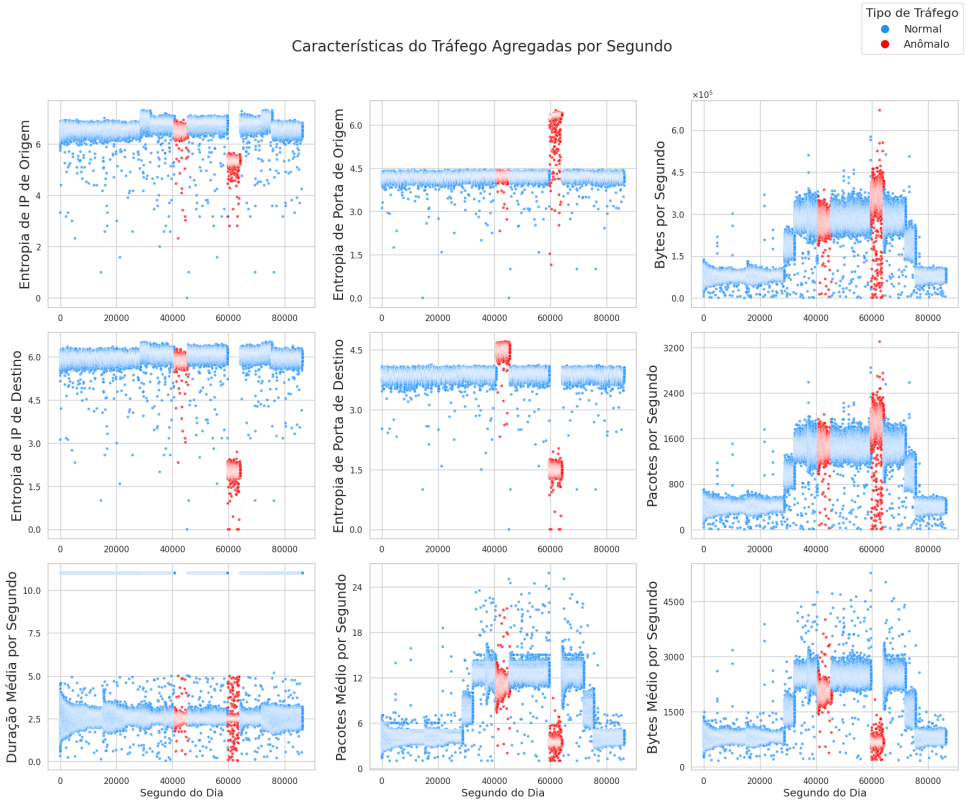


Figura 11 – Características do terceiro dia. Fonte: próprio autor.

6.2 Sistema de Detecção e Mitigação de Intrusão de Redes

A arquitetura do sistema de detecção e mitigação, ilustrada na Figura 12, baseia-se na separação dos planos do modelo SDN (Aplicação, Controle e Dados). O Sistema de Segurança, o componente central desta aplicação, opera no Plano de Aplicação. O processo é iniciado pelo Coletor de tráfego, que requisita os dados de rede ao Controlador SDN por meio da Interface *Northbound*. Após a coleta, os dados passam pelo Pré-processador de dados.

A etapa seguinte é o Detector de anomalias, responsável por analisar os dados processados. Caso uma Anomalia seja detectada, o Mitigador de anomalias é acionado. Este componente implementa a Política de mitigação e a envia de volta ao Controlador SDN que utiliza a Interface *Southbound* para aplicar as ações de mitigação nos Dispositivos de encaminhamento no Plano de Dados. Em seguida, esse sistema será explicado em mais detalhes.

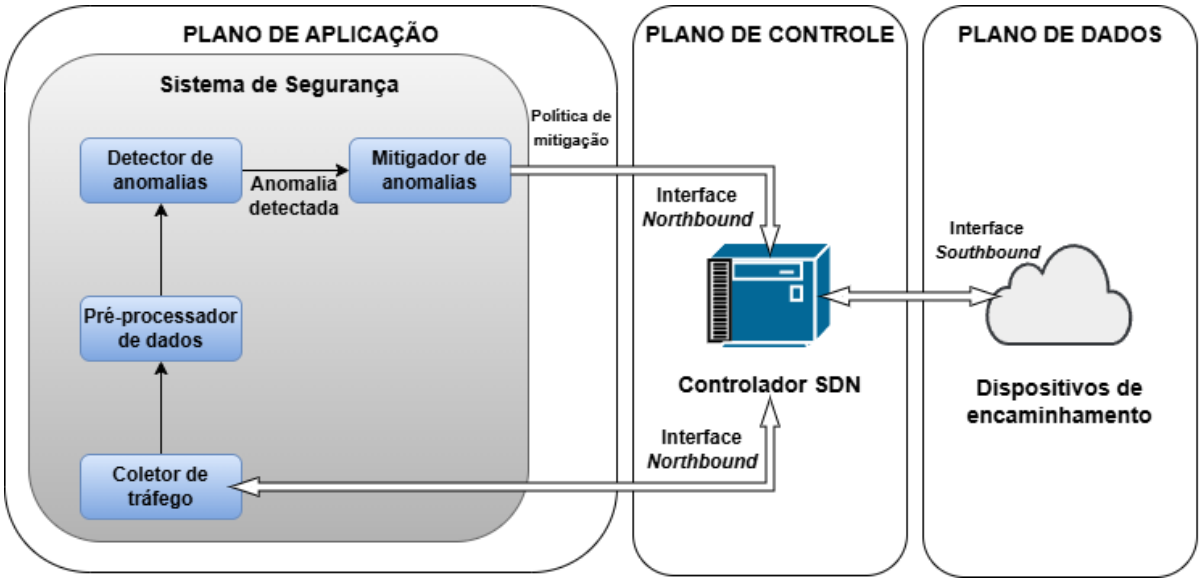


Figura 12 – Arquitetura do Sistema Proposto. Fonte: próprio autor.

6.2.1 Coletor de Tráfego e Pré-processador de Dados

O Coletor de Tráfego é o componente responsável pela aquisição dos dados. Ele opera interagindo com a Interface *Northbound* do Controlador SDN, requisitando os fluxos IP da rede e obtendo esses dados em tempo real. Após a coleta, os dados brutos são encaminhados ao Pré-processador de dados. Este módulo executa as etapas de transformação necessárias para adequar os dados ao formato esperado pelo Detector de Anomalias. O processamento é feito em duas fases:

1. **Engenharia de características:** Esta etapa aplica a Entropia de Shannon nas

portas de destino dos dados agregados na janela de 1 segundo, resultando na única característica utilizada: a Entropia de Porta de Destino.

2. **Normalização de dados:** A Entropia de Porta de Destino resultante é submetida a uma normalização. Utiliza-se a técnica Padronizador Padrão (*Standard Scaler*) [103], que centraliza os dados, removendo a média e escalonando-os pelo desvio padrão. Este passo é realizado para proporcionar estabilidade no treinamento do modelo [104]. A fórmula aplicada a cada amostra x é:

$$z = \frac{x - \mu}{\sigma} \tag{6.2}$$

Onde x é o valor da característica, μ é a média e σ é o desvio padrão da característica, cujos valores são calculados a partir do conjunto de dados de treinamento, e z é o valor normalizado resultante.

A saída deste módulo, está pronta para ser analisada pelo Detector de Anomalias.

6.2.2 Detector de Anomalias

O Detector de anomalias é o módulo central do sistema de segurança, implementado como um modelo de Aprendizado Profundo não supervisionado. A sua finalidade é aprender a reconstruir os dados do tráfego normal. A detecção ocorre ao identificar amostras que divergem deste padrão aprendido pelo modelo.

A arquitetura, ilustrada na Figura 13, é um Autocodificador Variacional que utiliza um Codificador do Transformador e um Perceptron de Múltiplas Camadas (MLP, do inglês *Multilayer Perceptron*) como decodificador. Esta abordagem híbrida captura o contexto temporal por meio do Transformador e, simultaneamente, modela a distribuição de probabilidade dos dados do dia normal por meio do VAE. Por este modelo ter o objetivo de detectar anomalias, utilizar o Codificador do Transformador e o VAE, ele será denominado AnomT-VAE.

6.2.2.1 AnomT-VAE

O modelo AnomT-VAE processa os dados da seguinte forma:

1. **Entrada e projeção:** A entrada é uma sequência de valores de entropia $\{e_i, e_{i+1}, \ldots, e_{i+127}\}$. Como o Transformador opera sobre vetores de dimensão d_{model}, a sequência de entrada passa por uma Projeção Linear, uma MLP. Esta operação desempenha a função de Camada de Incorporação, projetando cada elemento de entrada e_t em um vetor de dimensão d_{model}.

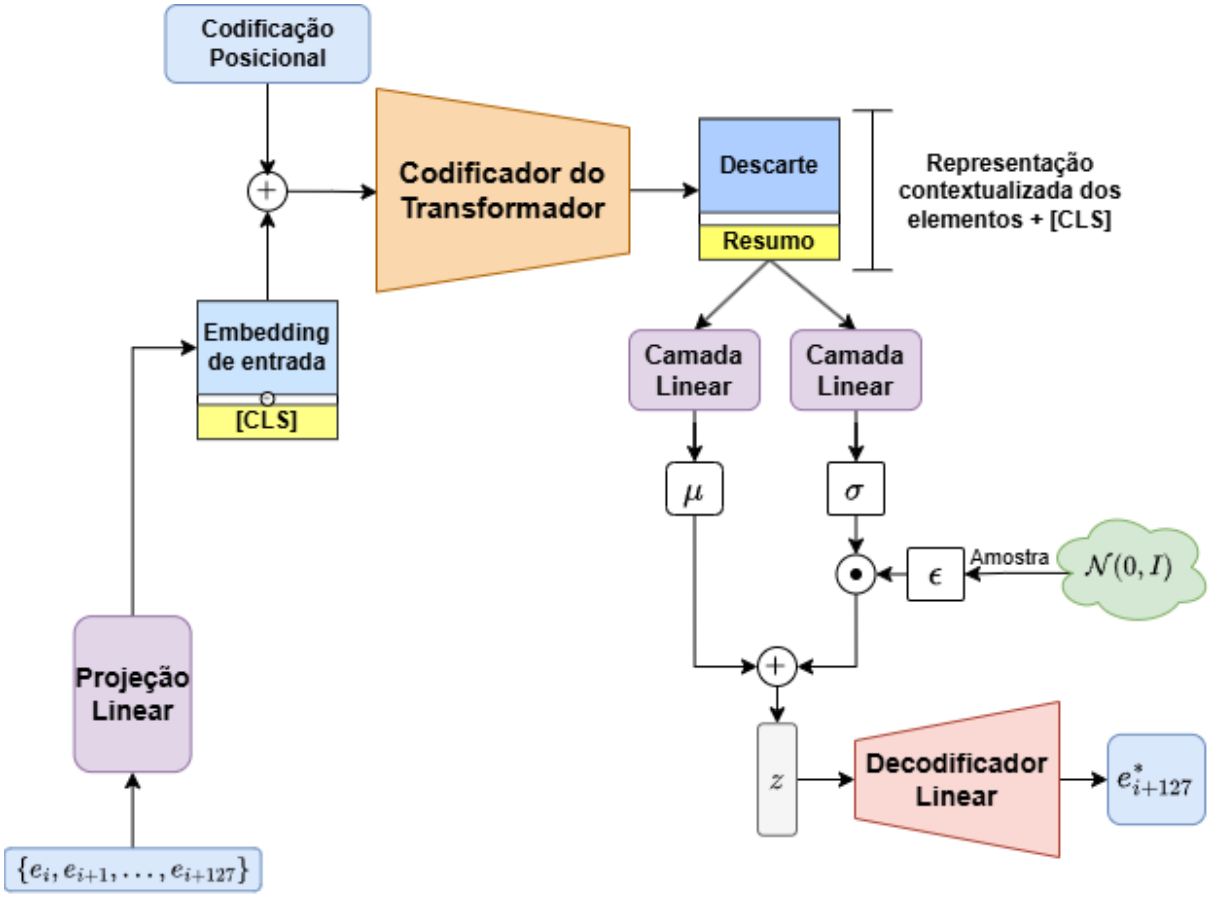


Figura 13 – AnomT-VAE. Fonte: próprio autor.

2. **Agregação de contexto:** Seguindo a técnica popularizada pelo modelo BERT [85] e utilizada em [91, 92], um elemento especial de classificação, representado por `[CLS]` (*Classification token*), é concatenado ao *Embedding* de entrada, resultando na sequência completa de *embeddings*. Originalmente, o `[CLS]` foi utilizado para tarefas de classificação [85], porém, nesse contexto, ele atua como um agregador de informações da sequência de entrada. Após a sequência completa de *embeddings* ser processada pelo Codificador do Transformador, a representação vetorial correspondente ao `[CLS]` tem por finalidade atuar como uma representação compacta e contextualizada de toda a sequência de entrada.

3. **Processamento pelo Transformador:** O Codificador Posicional injeta a informação da ordem temporal na sequência completa de *embeddings*. O Codificador do Transformador processa essa sequência com seu mecanismo de autoatenção. Isso permite que o modelo atribua um grau de importância aos elementos da sequência, capturando as dependências temporais da sequência de entrada e gerando uma matriz que representa a sequência completa de forma contextualizada.

4. **Codificação latente (VAE):** A partir dessa matriz, é extraída apenas a representação vetorial correspondente ao elemento `[CLS]`, identificado na figura como o

vetor de resumo, de cor amarela. Este vetor atua como a representação de toda a sequência de entrada e alimenta duas camadas lineares distintas, implementadas como MLPs. Estas camadas geram os parâmetros da distribuição latente: a média (μ) e a variância (σ^2).

5. **Amostragem e decodificação:** O vetor latente z é amostrado dessa distribuição utilizando o Truque da Reparametrização. O Decodificador Linear recebe o vetor z e tenta reconstruir o último elemento da sequência de entrada e^*_{i+127}.

6.2.2.2 Pontuação e Limiar

A detecção de anomalias é baseada em uma pontuação de anomalia composta pelos dois termos da função de perda do VAE:

1. **Erro de reconstrução:** É utilizado o Erro Quadrático Médio para comparar o último valor da sequência original (e_{i+127}) com o valor reconstruído pelo decodificador (e^*_{i+127}). Um erro que ultrapassa um limiar pré-definido indica que o modelo, treinado em dados normais, falhou em prever aquele valor, mostrando um comportamento anômalo.

2. **Divergência de Kullback-Leibler:** Este termo mede a distância entre a distribuição latente codificada e a distribuição a priori utilizada pelo VAE. Uma DKL que excede um limiar pré-definido sugere que a sequência de entrada desvia do padrão normal, o que faz com que o codificador mapeie para uma região do espaço latente não prevista durante o treinamento com dados normais.

Para classificar uma amostra, um limiar τ é definido de forma não supervisionada. Este processo é executado após o treinamento completo do modelo AnomT-VAE. Utiliza-se um conjunto de dados de validação, composto exclusivamente por tráfego normal, que não foi utilizado para o treinamento do modelo. O modelo treinado é executado sobre todas as amostras deste conjunto de validação normal. Para cada amostra, a pontuação de anomalia, MSE ou DKL, é calculada e armazenada.

Ao final desta execução, obtém-se uma distribuição que representa estatisticamente as pontuações de anomalia do tráfego normal. O limiar τ é estabelecido aplicando uma das duas abordagens estatísticas sobre essa distribuição:

- **Abordagem por percentil:** O limiar é definido como o valor no percentil q da distribuição das pontuações de anomalias dos dados normais. Por exemplo, a escolha de $q = 99,90$ é uma decisão que estabelece a sensibilidade do detector: ela define τ como o valor que $99,90\%$ dos dados normais não ultrapassam.

- **Abordagem paramétrica:** Esta abordagem assume que a distribuição das pontuações de anomalia dos dados normais pode ser aproximada por uma distribuição Gaussiana. Ela calcula a média (μ) e o desvio padrão (σ) dessa distribuição de pontuações. O limiar é definido pela fórmula $\tau = \mu + (k \times \sigma)$, onde k é um multiplicador.

O valor do limiar τ é determinado por meio de um dos métodos de definição disponíveis, sendo eles a abordagem por percentil e a abordagem paramétrica. Este processo possibilita cenários de avaliação distintos, nos quais o MSE e a DKL possuem seus limiares calculados de forma independente. Durante a fase de inferência, a pontuação da amostra é comparada unicamente com o limiar correspondente à métrica, MSE ou DKL, e ao método selecionado para aquele cenário específico, percentil ou paramétrico. Caso a pontuação exceda o valor τ estabelecido, a amostra de entrada é classificado como anômalo e essa informação é passada para o Mitigador de anomalias.

6.2.3 Mitigador de Anomalias

Quando o Detector de Anomalias reporta uma pontuação ($Pontuação > \tau$) para um segundo específico, t, o Mitigador de Anomalias é ativado. Inspirado em Ruffo *et al.* [15], este módulo utiliza regras para identificar as vítimas e os atacantes. A análise de vítimas opera sobre uma janela de 6 segundos ($[t - 5, t]$). O mitigador avalia todos os fluxos dentro dessa janela e tenta identificar as vítimas dos ataques utilizando duas condições:

- Primeiramente, o sistema identifica o endereço IP que recebeu o maior volume de fluxos na janela. Se esta contagem de fluxos ultrapassar um limiar pré-definido, este IP é classificado como vítima de *DDoS*.
- De forma similar, o sistema identifica o endereço IP associado ao maior número de portas de destino distintas. Se esta contagem de portas únicas ultrapassar um limiar, o IP é classificado como vítima de *Port Scan*.

Caso o sistema identifique pelo menos uma vítima, seja de *DDoS* ou *Port Scan*, ele inicia a identificação dos atacantes. Esta segunda análise verifica apenas os fluxos do segundo t, o segundo anômalo. Todos os endereços IPs de origem que estabeleceram comunicação com uma vítima identificada nessa janela são classificados como anômalos.

A lista resultante de IPs de origem anômalos forma a Política de mitigação. Conforme ilustrado na Figura 12, esta política é enviada pelo Mitigador ao Controlador SDN O Controlador SDN, situado no Plano de Controle, recebe esta instrução e cria as regras de fluxo. O controlador, então, utiliza a Interface *Southbound* para programar os Dispositivos de encaminhamento, localizados no Plano de Dados, instruindo-os a bloquear o tráfego dos IPs atacantes identificados.

6.3 Experimentos e Resultados

Os experimentos foram conduzidos integralmente no ambiente Google Colab, GPU (*Graphics Processing Unit*) NVIDIA T4. As principais bibliotecas utilizadas na implementação e análise estão sumarizadas na Tabela 1. Nesta seção serão detalhados os experimentos e os resultados obtidos.

Tabela 1 – Principais Bibliotecas. Fonte: próprio autor.

Categoria	**Bibliotecas**
Processamento de Dados	NumPy, Pandas
Aprendizado Profundo	PyTorch
Métricas de Avaliação	Scikit-learn
Visualização	Matplotlib, Seaborn
Utilitários	SciPy, tqdm, Joblib

6.3.1 Detecção de Anomalias

O modelo AnomT-VAE foi treinado com os hiperparâmetros mostrados na Tabela 2. A configuração do modelo (dimensões do Transformador, VAE) e os parâmetros de treinamento otimizados por meio de ajuste experimental. Os parâmetros de mitigação foram calibrados com base no perfil do tráfego normal. Isso foi feito analisando a contagem de IPs de origem e portas distintas para o mesmo IP de destino na janela de 6 segundos de apenas dados benignos.

Tabela 2 – Hiperparâmetros do Modelo e Treinamento. Fonte: próprio autor.

Parâmetro	**Valor**
Parâmetros de Treinamento	
Semente aleatória	42
Tamanho do lote	256
Número de épocas	10
Taxa de aprendizado	1×10^{-3}
Decaimento de peso	1×10^{-5}
Arquitetura do Modelo	
Comprimento da sequência	128
Dimensão de *embedding* (d_{model})	16
Camadas do Codificador do Transformador	2
Cabeças de atenção do Codificador	2
Dimensão da camada FFN	128
Dimensão latente (VAE)	16
Parâmetros de Mitigação	
Janela de mitigação	6 segundos
Limiar de *DDoS*	10
Limiar de *Port Scan*	10

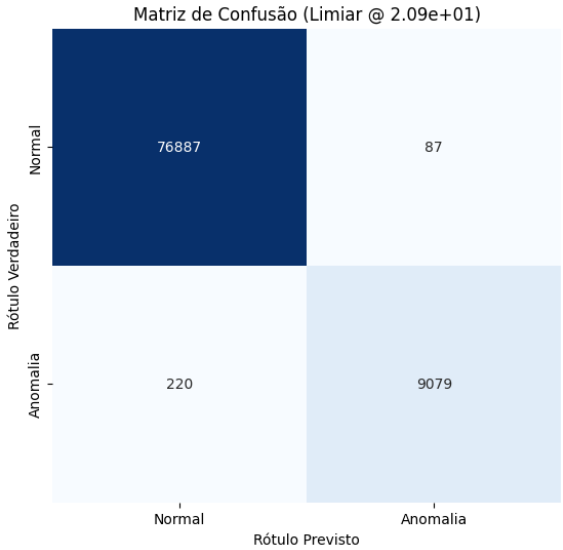
O treinamento do modelo seguiu a abordagem não supervisionada, utilizando exclusivamente os dados do dia 1, representados na Figura 9 (tráfego normal). O conjunto de dados de validação, utilizado para definir o limiar τ, foi composto apenas pelos períodos de tráfego normal do dia 3, mostrados na Figura 11. Por este motivo, o conjunto de dados do dia 3 não foi utilizado como teste.

O teste foi realizado sobre o conjunto de dados completo referente ao dia 2, conforme ilustrado na Figura 10. A Tabela 3 apresenta as métricas de desempenho da detecção, derivadas a partir das matrizes de confusão exibidas na Figura 14 e pelas curvas ROC, ilustradas pela Figura 15. Os resultados estão organizados de acordo com a abordagem de limiar e o método de pontuação de anomalia. Os limiares foram definidos com a seguinte configuração: para a abordagem por percentil, foi utilizado o valor $q = 99,90$ e para a abordagem paramétrica, foi adotado o valor $k = 3$.

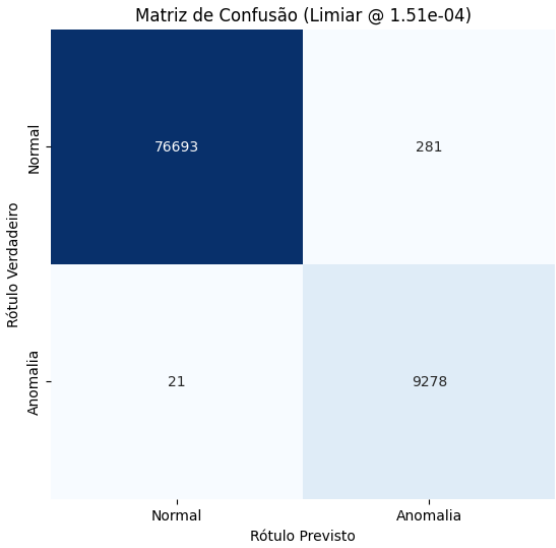
Ao analisar os dados, observa-se que, na configuração utilizando a abordagem de percentil, ambos os métodos de pontuação demonstraram valores superiores a $0,97$ em todas as métricas avaliadas. Para a abordagem paramétrica, o desempenho foi inferior nas métricas em geral. O maior MCC foi obtido pela combinação do percentil com KLD, atingindo o valor de $0,9821$. Por conta desse resultado, essa foi a configuração selecionada para prosseguir na etapa do mitigador.

Tabela 3 – Resultados das métricas de desempenho separadas por abordagem. Fonte: próprio autor.

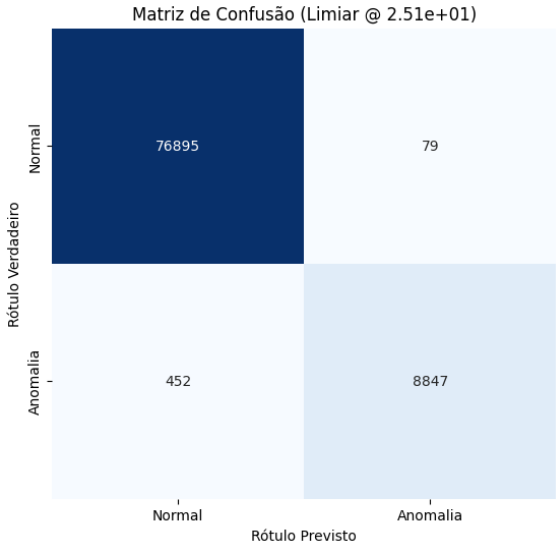
Abordagem	**Método**	**Precisão**	**Revocação**	***F1-Score***	**MCC**	**AUROC**
Percentil	MSE	0,9905	0,9763	0,9834	0,9814	**0,9994**
	KLD	0,9706	0,9977	**0,9840**	**0,9821**	0,9989
Paramétrica	MSE	**0,9911**	0,9514	0,9709	0,9677	**0,9994**
	KLD	0,8710	**0,9981**	0,9302	0,9238	0,9989



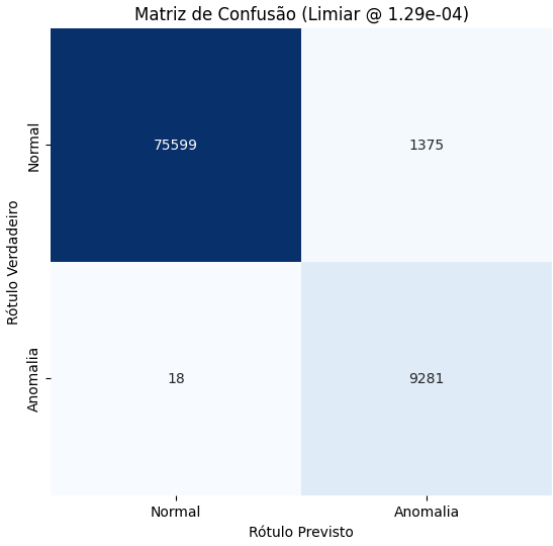
Percentil - KLD



Percentil - KLD

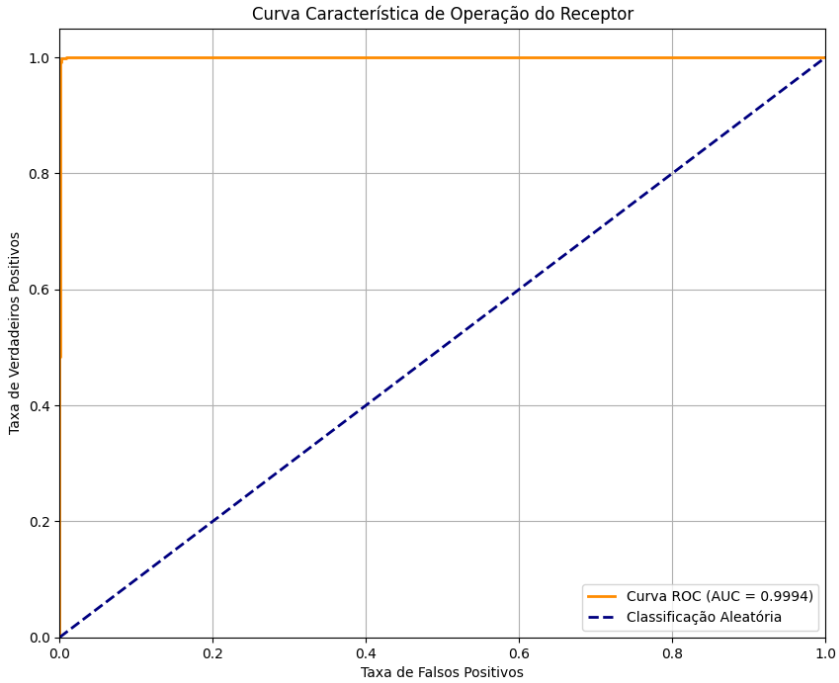


Paramétrica - MSE

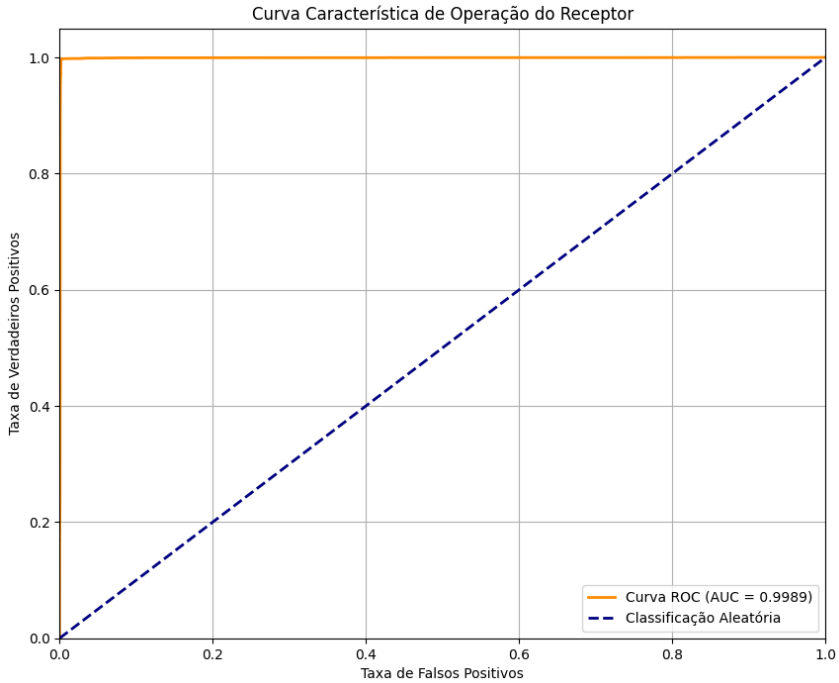


Paramétrica - KLD

Figura 14 – Matrizes de confusão para as quatro configurações avaliadas. Fonte: próprio autor.



MSE (AUC = 0,9994)



KLD (AUC = 0,9989)

Figura 15 – Curvas ROC para as funções de perda MSE e KLD. Fonte: próprio autor.

6.3.2 Mitigação de Anomalias

A avaliação do Mitigador de Anomalias baseou-se na detecçào configurada pela abordagem de percentil à pontuação DKL. Após o Detector de Anomalias identificar o intervalo de um segundo como anômalo, o Mitigador aplica as regras para classificar os IPs de origem como anômalos.

Os resultados desta classificação em nível de fluxo estão sumarizados na Tabela 4. As métricas de precisão, revocação, *F1-score* e MCC avaliam o desempenho final do sistema na identificação de fluxos normais e anômalos:

Tabela 4 – Resultados a Nível de Fluxo do Sistema (Detecção + Mitigação). Fonte: próprio autor.

Métricas	**Valor**
Precisão	0,9965
Revocação	0,9962
F1-score	0,9964
MCC (Matthews)	0,9955

A Tabela 5 mostra os fluxos encaminhados e bloqueados pelo sistema implementado, calculando as porcentagens relativas a cada classe verdadeira por linha.

Tabela 5 – Resultados da Mitigação. Fonte: próprio autor.

Classe do Fluxo	**Encaminhado**	**Bloqueado**
Normal (Benigno)	9.414.587 (99,91%)	8.399 (0,09%)
Anomalia (Maligno)	9.158 (0,37%)	2.446.449 (99,63%)

6.4 Discussão

A análise dos resultados experimentais, exibidos pela Tabela 3, demonstra a robustez da arquitetura proposta, AnomT-VAE. O modelo obteve valores superiores a $0,97$ em todas as métricas testadas, alcançando um desempenho similar ao de trabalhos de referência no conjunto de dados Orion [15, 105, 106]. Adicionalmente, após feita a detecção, o Mitigador de Anomalias obteve valores superiores a $0,99$ em todas as métricas analisadas, mostrado pela Tabela 4.

A principal desvantagem desta abordagem reside no custo computacional do modelo, podendo ser necessária a utilização de GPUs ou TPUs (*Tensor Processing Unit*) para a sua implementação. A utilização do Codificador do Transformador, embora seja eficaz para capturar dependências contextuais e temporais, é reconhecida na literatura como uma abordagem computacionalmente custosa [17]. Este custo pode representar um desafio para a implementação em tempo real em cenários com restrições de *hardware*, sendo um ponto de atenção para futuros trabalhos.

Outra limitação da abordagem utilizada refere-se à granularidade temporal do sistema, que opera com agregações em intervalos de segundo a segundo. Embora essa abordagem seja eficaz para identificar comportamentos anômalos, ela restringe a capacidade de identificar, de forma imediata e granular, qual fluxo específico desencadeou o alerta dentro da janela temporal. Utilizar modelos de detecção diretamente sobre os fluxos é outro ponto de aprimoramento do trabalho.

7 CONCLUSÃO

Este trabalho apresentou um estudo sobre a aplicação de Aprendizado Profundo em Sistemas de Detecção de Intrusão em Redes. Inicialmente, foi realizada uma fundamentação teórica sobre os mecanismos do Aprendizado Profundo e das arquiteturas Autocodificador Variacional e do Transformador.

Foram implementadas partes de um sistema de segurança, incluindo Pré-processador de dados, Detector de Anomalias e Mitigador de Anomalias. A validação experimental foi conduzida utilizando o conjunto de dados Orion, que emula tráfego de rede em fluxos IP, contendo um dia de tráfego normal, utilizado para treinamento, e dois dias com ataques de *Port Scan* e *DDoS*, utilizados para validação e teste. O pré-processamento converteu os dados brutos em uma série temporal de característica única, a Entropia de Porta de Destino.

Os experimentos conduzidos sob a configuração de limiar definido pela abordagem por percentil confirmaram a robustez do Detector de Anomalias AnomT-VAE na classificação em nível de segundo. Ao adotar o Erro de Reconstrução como pontuação de anomalia, o sistema atingiu Precisão de $0,9905$, Revocação de $0,9763$, *F1-score* de $0,9834$, MCC de $0,9814$ e AUROC de $0,9994$. O desempenho mostrou-se ligeiramente superior nas métricas em geral ao utilizar a Divergência de Kullback-Leibler, cenário no qual o modelo obteve o melhor *F1-score* $(0,9840)$ e MCC $(0,9821)$, com Precisão de $0,9706$, Revocação de $0,9977$ e AUROC de $0,9989$.

O Mitigador de Anomalias, responsável pela classificação dos fluxos individuais, também apresentou desempenho robusto. O sistema alcançou Precisão de $0,9965$, Revocação de $0,9962$, *F1-score* de $0,9964$ e MCC de $0,9955$. A análise da classificação demonstra que o sistema encaminhou $99,91\%$ dos fluxos benignos e bloqueou $99,63\%$ dos fluxos malignos corretamente.

A principal limitação identificada nesta abordagem é o custo computacional da arquitetura. O Codificador do Transformador é reconhecido na literatura por sua alta demanda de processamento, podendo exigir o uso de GPUs ou TPUs para sua utilização. Conclui-se que, embora a arquitetura AnomT-VAE seja robusta para a tarefa de detecção nos dados Orion, sua implementação em cenários reais exigiria *hardware* de alto desempenho ou a adoção de arquiteturas de Transformador otimizadas.

Como trabalhos futuros, sugere-se a investigação de duas frentes: a aplicação de modelos de detecção diretamente sobre os fluxos IP, evitando a agregação por segundo, e a exploração de arquiteturas de Transformador otimizadas, tais como variantes esparsas ou destiladas, para reduzir a carga computacional da inferência.

REFERÊNCIAS

[1] PRINCE, S. J. *Understanding Deep Learning*. The MIT Press, 2023. Disponível em: <http://udlbook.com>.

[2] VASWANI, A. et al. Attention is all you need. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 6000–6010. ISBN 9781510860964. Disponível em: <https://dl.acm.org/doi/10.5555/3295222.3295349>.

[3] TANENBAUM, A. S.; WETHERALL, D. J. *Computer Networks*. 5th. ed. [S.l.]: Prentice Hall, 2011. ISBN 0132126958.

[4] CHESHER, C. How computer networks became social. In: _____. *Second International Handbook of Internet Research*. Dordrecht: Springer Netherlands, 2020. p. 105–125. ISBN 978-94-024-1555-1. Disponível em: <https://doi.org/10.1007/978-94-024-1555-1_4>.

[5] FERNANDES, G. et al. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, v. 70, n. 3, p. 447–489, Mar 2019. ISSN 1572-9451. Disponível em: <https://doi.org/10.1007/s11235-018-0475-8>.

[6] CREMER, F. et al. Cyber risk and cybersecurity: a systematic review of data availability. *The Geneva Papers on Risk and Insurance - Issues and Practice*, v. 47, n. 3, p. 698–736, Jul 2022. Disponível em: <https://doi.org/10.1057/s41288-022-00266-6>.

[7] POURRAHMANI, H. et al. A review of the security vulnerabilities and countermeasures in the internet of things solutions: A bright future for the blockchain. *Internet of Things*, v. 23, p. 100888, 2023. ISSN 2542-6605. Disponível em: <https://doi.org/10.1016/j.iot.2023.100888>.

[8] TARIQ, U. et al. A critical cybersecurity analysis and future research directions for the internet of things: A comprehensive review. *Sensors*, v. 23, n. 8, 2023. ISSN 1424-8220. Disponível em: <https://doi.org/10.3390/s23084117>.

[9] GUO, Y. A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer Communications*, v. 198, p. 175–185, 2023. ISSN 0140-3664. Disponível em: <https://doi.org/10.1016/j.comcom.2022.11.001>.

[10] CISCO. Cisco annual internet report (2018–2023) white paper. *Cisco: San Jose, CA, USA*, v. 10, n. 1, p. 1–35, 2020. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.

[11] da Silva Ruffo, V. G. et al. Anomaly and intrusion detection using deep learning for software-defined networks: A survey. *Expert Systems with Applications*, v. 256, p. 124982, 2024. ISSN 0957-4174. Disponível em: <https://doi.org/10.1016/j.eswa.2024.124982>.

[12] ALI, M. L. et al. Deep learning vs. machine learning for intrusion detection in computer networks: A comparative study. *Applied Sciences*, v. 15, n. 4, 2025. ISSN 2076-3417. Disponível em: <https://doi.org/10.3390/app15041903>.

[13] DONG, B.; WANG, X. Comparison deep learning method to traditional methods using for network intrusion detection. In: *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*. [s.n.], 2016. p. 581–585. Disponível em: <https://doi.org/10.1109/ICCSN.2016.7586590>.

[14] LENT, D. M. B. et al. A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks. *IEEE Access*, v. 10, p. 73229–73242, 2022. Disponível em: <https://doi.org/10.1109/ACCESS.2022.3190008>.

[15] RUFFO, V. G. d. S. et al. f-anogan for unsupervised attack detection in sdn environment. *IEEE Transactions on Network Science and Engineering*, p. 1–17, 2025. Disponível em: <https://doi.org/10.1109/TNSE.2025.3558936>.

[16] CHINNASAMY, R. et al. Deep learning-driven methods for network-based intrusion detection systems: A systematic review. *ICT Express*, v. 11, n. 1, p. 181–215, 2025. ISSN 2405-9595. Disponível em: <https://doi.org/10.1016/j.icte.2025.01.005>.

[17] ALSHOMRANI, M. et al. Survey of transformer-based malicious software detection systems. *Electronics*, v. 13, n. 23, 2024. ISSN 2079-9292. Disponível em: <https://doi.org/10.3390/electronics13234677>.

[18] WU, Z. et al. Rtids: A robust transformer-based approach for intrusion detection system. *IEEE Access*, v. 10, p. 64375–64387, 2022. Disponível em: <https://doi.org/10.1109/ACCESS.2022.3182333>.

[19] REZA, S. et al. A multi-head attention-based transformer model for traffic flow forecasting with a comparative analysis to recurrent neural networks. *Expert Systems with Applications*, v. 202, p. 117275, 2022. ISSN 0957-4174. Disponível em: <https://doi.org/10.1016/j.eswa.2022.117275>.

[20] KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Banff, AB, Canada: [s.n.], 2014. Disponível em: <https://openreview.net/forum?id=33X9fd2-9FyZd>.

[21] YAO, R. et al. Unsupervised anomaly detection using variational auto-encoder based feature extraction. In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. [s.n.], 2019. p. 1–7. Disponível em: <https://doi.org/10.1109/ICPHM.2019.8819434>.

[22] MIRSKY, Y. et al. *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*. 2018. Disponível em: <https://doi.org/10.48550/arXiv.1802.09089>.

[23] AKKEPALLI, S.; K, S. Copula entropy regularization transformer with c2 variational autoencoder and fine-tuned hybrid dl model for network intrusion detection. *Telematics and Informatics Reports*, v. 17, p. 100182, 2025. ISSN 2772-5030. Disponível em: <https://doi.org/10.1016/j.teler.2024.100182>.

[24] SINGH, S.; JHA, R. K. A survey on software defined networking: Architecture for next generation network. *Journal of Network and Systems Management*, v. 25, n. 2, p. 321–374, Apr 2017. ISSN 1573-7705. Disponível em: <https://doi.org/10.1007/s10922-016-9393-9>.

[25] YUNGAICELA-NAULA, N. M. et al. Towards security automation in software defined networks. *Computer Communications*, Elsevier, v. 183, p. 64–82, 2022. Disponível em: <https://doi.org/10.1016/j.comcom.2021.11.014>.

[26] JANABI, A. H.; KANAKIS, T.; JOHNSON, M. Survey: Intrusion detection system in software-defined networking. *IEEE Access*, v. 12, p. 164097–164120, 2024. Disponível em: <https://doi.org/10.1109/ACCESS.2024.3493384>.

[27] CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 41, n. 3, jul. 2009. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/1541880.1541882>.

[28] IGLESIAS, F.; ZSEBY, T. Analysis of network traffic features for anomaly detection. *Machine Learning*, v. 101, n. 1, p. 59–84, Oct 2015. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/s10994-014-5473-9>.

[29] KHAN, A. Q. et al. Knowledge-based anomaly detection: Survey, challenges, and future directions. *Engineering Applications of Artificial Intelligence*, v. 136, p. 108996, 2024. ISSN 0952-1976. Disponível em: <https://doi.org/10.1016/j.engappai.2024.108996>.

[30] YANG, Z. et al. A systematic literature review of methods and datasets for anomaly-based network intrusion detection. *Computers Security*, v. 116, p. 102675, 2022. ISSN 0167-4048. Disponível em: <https://doi.org/10.1016/j.cose.2022.102675>.

[31] PROENÇA, M. L. et al. Baseline to help with network management. In: _____. *e-Business and Telecommunication Networks*. Dordrecht: Springer Netherlands, 2006. p. 158–166. ISBN 978-1-4020-4761-9. Disponível em: <https://doi.org/10.1007/1-4020-4761-4_12>.

[32] RAHMAN, M. M.; SHAKIL, S. A.; MUSTAKIM, M. R. A survey on intrusion detection system in iot networks. *Cyber Security and Applications*, v. 3, p. 100082, 2025. ISSN 2772-9184. Disponível em: <https://doi.org/10.1016/j.csa.2024.100082>.

[33] ARNOB, A. K. B. et al. A comprehensive systematic review of intrusion detection systems: emerging techniques, challenges, and future research directions. *Journal of Edge Computing*, v. 4, n. 1, p. 73–104, May 2025. Disponível em: <https://doi.org/10.55056/jec.885>.

[34] GOLDSCHMIDT, P.; CHUDá, D. Network intrusion datasets: A survey, limitations, and recommendations. *Computers Security*, v. 156, p. 104510, 2025. ISSN 0167-4048. Disponível em: <https://doi.org/10.1016/j.cose.2025.104510>.

[35] ENNAJI, S. et al. Adversarial challenges in network intrusion detection systems: Research insights and future prospects. *IEEE Access*, v. 13, p. 148613–148645, 2025. Disponível em: <https://doi.org/10.1109/ACCESS.2025.3600984>.

[36] ELTANBOULY, S. et al. Network packet transformation approaches for intrusion detection systems: A survey. *IEEE Access*, v. 13, p. 107293–107312, 2025. Disponível em: <https://doi.org/10.1109/ACCESS.2025.3581354>.

[37] ASSIS, M. V. et al. A gru deep learning system against attacks in software defined networks. *Journal of Network and Computer Applications*, Elsevier, v. 177, p. 102942, 2021. Disponível em: <https://doi.org/10.1016/j.jnca.2020.102942>.

[38] NOVAES, M. P. et al. Adversarial deep learning approach detection and defense against ddos attacks in sdn environments. *Future Generation Computer Systems*, Elsevier, v. 125, p. 156–167, 2021. Disponível em: <https://doi.org/10.1016/j.future.2021.06.047>.

[39] NOVAES, M. P. et al. Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment. *IEEE Access*, IEEE, v. 8, p. 83765–83781, 2020. Disponível em: <https://doi.org/10.1109/ACCESS.2020.2992044>.

[40] SCARANTI, G. F. et al. Artificial immune systems and fuzzy logic to detect flooding attacks in software-defined networks. *IEEE Access*, IEEE, v. 8, p. 100172–100184, 2020. Disponível em: <https://doi.org/10.1109/ACCESS.2020.2997939>.

[41] ASSIS, M. V. de et al. Near real-time security system applied to sdn environments in iot networks using convolutional neural network. *Computers & Electrical Engineering*, Elsevier, v. 86, p. 106738, 2020. Disponível em: <https://doi.org/10.1016/j.compeleceng.2020.106738>.

[42] HAMAMOTO, A. H. et al. Network anomaly detection system using genetic algorithm and fuzzy logic. *Expert Systems with Applications*, Elsevier, v. 92, p. 390–402, 2018. Disponível em: <https://doi.org/10.1016/j.eswa.2017.09.013>.

[43] CARVALHO, L. F. et al. An ecosystem for anomaly detection and mitigation in software-defined networking. *Expert Systems with Applications*, Elsevier, v. 104, p. 121–133, 2018. Disponível em: <https://doi.org/10.1016/j.eswa.2018.03.027>.

[44] ASSIS, M. V. D. et al. Fast defense system against attacks in software defined networks. *IEEE Access*, IEEE, v. 6, p. 69620–69639, 2018. Disponível em: <https://doi.org/10.1109/ACCESS.2018.2878576>.

[45] PENA, E. H. et al. Anomaly detection using the correlational paraconsistent machine with digital signatures of network segment. *Information Sciences*, Elsevier, v. 420, p. 313–328, 2017. Disponível em: <https://doi.org/10.1016/j.ins.2017.08.074>.

[46] LENT, D. M. B. et al. An unsupervised generative adversarial network system to detect ddos attacks in sdn. *IEEE Access*, v. 12, p. 70690–70706, 2024. Disponível em: <https://doi.org/10.1109/ACCESS.2024.3402069>.

[47] CARVALHO, L. F. et al. A novel anomaly detection system to assist network management in sdn environment. In: *2017 IEEE International Conference on Communications (ICC)*. [s.n.], 2017. p. 1–6. Disponível em: <https://doi.org/10.1109/ICC.2017.7997214>.

[48] ADANIYA, M. H. A. C. et al. Anomaly detection using dsns and firefly harmonic clustering algorithm. In: *2012 IEEE International Conference on Communications (ICC)*. [s.n.], 2012. p. 1183–1187. Disponível em: <https://doi.org/10.1109/ICC.2012.6364088>.

[49] PROENÇA, M. L. et al. The hurst parameter for digital signature of network segment. In: SOUZA, J. N. de; DINI, P.; LORENZ, P. (Ed.). *Telecommunications and Networking - ICT 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 772–781. ISBN 978-3-540-27824-5. Disponível em: <https://doi.org/10.1007/978-3-540-27824-5_103>.

[50] SARKER, I. H. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, v. 2, n. 6, p. 420, Aug 2021. ISSN 2661-8907. Disponível em: <https://doi.org/10.1007/s42979-021-00815-1>.

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

[52] NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [s.n.], 2010. p. 807–814. Disponível em: <https://dl.acm.org/doi/10.5555/3104322.3104425>.

[53] LU, Z. et al. The expressive power of neural networks: a view from the width. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 6232–6240. ISBN 9781510860964. Disponível em: <https://dl.acm.org/doi/10.5555/3295222.3295371>.

[54] BISHOP, C. M. *Deep Learning: Foundations and Concepts*. Springer, 2023. ISBN 978-3031263435. Disponível em: <https://www.bishopbook.com/>.

[55] SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. Disponível em: <https://dl.acm.org/doi/book/10.5555/2621980>.

[56] HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd. ed. Springer, 2009. Disponível em: <https://hastie.su.domains/ElemStatLearn/>.

[57] MURPHY, K. P. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. Disponível em: <http://probml.github.io/book2>.

[58] KULLBACK, S.; LEIBLER, R. A. On information and sufficiency. *The annals of mathematical statistics*, JSTOR, v. 22, n. 1, p. 79–86, 1951. Disponível em: <https://doi.org/10.1214/aoms/1177729694>.

[59] RUDER, S. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. Disponível em: <https://doi.org/10.48550/arXiv.1609.04747>.

[60] KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2017. Disponível em: <https://doi.org/10.48550/arXiv.1412.6980>.

[61] LOSHCHILOV, I.; HUTTER, F. *Decoupled Weight Decay Regularization*. 2019. Disponível em: <https://doi.org/10.48550/arXiv.1711.05101>.

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

[63] LI, P.; PEI, Y.; LI, J. A comprehensive survey on design and application of autoencoder in deep learning. *Applied Soft Computing*, v. 138, p. 110176, 2023. ISSN 1568-4946. Disponível em: <https://doi.org/10.1016/j.asoc.2023.110176>.

[64] CASELLA, G.; BERGER, R. L. *Statistical Inference*. 2nd. ed. [S.l.]: Duxbury Press, 2002. ISBN 978-0534243128.

[65] KEOGH, E.; MUEEN, A. Curse of dimensionality. In: _____. *Encyclopedia of Machine Learning and Data Mining*. Boston, MA: Springer US, 2017. p. 314–315. ISBN 978-1-4899-7687-1. Disponível em: <https://doi.org/10.1007/978-1-4899-7687-1_192>.

[66] KINGMA, D. P.; WELLING, M. An introduction to variational autoencoders. *Found. Trends Mach. Learn.*, Now Publishers Inc., Hanover, MA, USA, v. 12, n. 4, p. 307–392, nov. 2019. ISSN 1935-8237. Disponível em: <https://doi.org/10.1561/2200000056>.

[67] HIGGINS, I. et al. beta-VAE: Learning basic visual concepts with a constrained variational framework. In: *International Conference on Learning Representations*. [s.n.], 2017. Disponível em: <https://openreview.net/forum?id=Sy2fzU9gl>.

[68] ROMBACH, R. et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. Disponível em: <https://doi.org/10.48550/arXiv.2112.10752>.

[69] AN, J.; CHO, S. Variational autoencoder based anomaly detection using reconstruction probability. In: . [s.n.], 2015. Disponível em: <https://api.semanticscholar.org/CorpusID:36663713>.

[70] WANG, M. et al. Anomaly detection in multidimensional time series for water injection pump operations based on lstma-ae and mechanism constraints. *Scientific Reports*, v. 15, n. 1, p. 2020, Jan 2025. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-025-85436-x>.

[71] LI, Z.; HUANG, C.; QIU, W. An intrusion detection method combining variational auto-encoder and generative adversarial networks. *Computer Networks*, v. 253, p. 110724, 2024. ISSN 1389-1286. Disponível em: <https://doi.org/10.1016/j.comnet.2024.110724>.

[72] SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA: MIT Press, 2014. (NIPS'14), p. 3104–3112. Disponível em: <https://doi.org/10.48550/arXiv.1409.3215>.

[73] ISLAM, S. et al. A comprehensive survey on applications of transformers for deep learning tasks. *Expert Systems with Applications*, v. 241, p. 122666, 2024. ISSN 0957-4174. Disponível em: <https://doi.org/10.1016/j.eswa.2023.122666>.

[74] MIENYE, I. D.; SWART, T. G.; OBAIDO, G. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, v. 15, n. 9, 2024. ISSN 2078-2489. Disponível em: <https://doi.org/10.3390/info15090517>.

[75] SHERSTINSKY, A. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, v. 404, p. 132306, 2020. ISSN 0167-2789. Disponível em: <https://doi.org/10.1016/j.physd.2019.132306>.

[76] JOHNSTON, L. et al. Revisiting the problem of learning long-term dependencies in recurrent neural networks. *Neural Networks*, v. 183, p. 106887, 2025. ISSN 0893-6080. Disponível em: <https://doi.org/10.1016/j.neunet.2024.106887>.

[77] MIKOLOV, T. et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. Disponível em: <https://doi.org/10.48550/arXiv.1301.3781>.

[78] ZERVEAS, G. et al. A transformer-based framework for multivariate time series representation learning. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: Association for Computing Machinery, 2021. (KDD '21), p. 2114–2124. ISBN 9781450383325. Disponível em: <https://doi.org/10.1145/3447548.3467401>.

[79] HE, K. et al. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [s.n.], 2016. p. 770–778. Disponível em: <https://doi.org/10.1109/CVPR.2016.90>.

[80] BA, J. L.; KIROS, J. R.; HINTON, G. E. *Layer Normalization*. 2016. Disponível em: <https://doi.org/10.48550/arXiv.1607.06450>.

[81] LIN, T. et al. A survey of transformers. *AI Open*, v. 3, p. 111–132, 2022. ISSN 2666-6510. Disponível em: <https://doi.org/10.1016/j.aiopen.2022.10.001>.

[82] WEN, Q. et al. Transformers in time series: a survey. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. [s.n.], 2023. (IJCAI '23). ISBN 978-1-956792-03-4. Disponível em: <https://doi.org/10.24963/ijcai.2023/759>.

[83] RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, JMLR.org, v. 21, n. 1, jan. 2020. ISSN 1532-4435. Disponível em: <https://doi.org/10.5555/3455716.3455856>.

[84] LEWIS, M. et al. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: JURAFSKY, D. et al. (Ed.). *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020. p. 7871–7880. Disponível em: <https://doi.org/10.18653/v1/2020.acl-main.703>.

[85] DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In: BURSTEIN, J.; DORAN, C.; SOLORIO, T. (Ed.). *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: <https://doi.org/10.18653/v1/N19-1423>.

[86] BROWN, T. B. et al. Language models are few-shot learners. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2020. (NIPS '20). ISBN 9781713829546. Disponível em: <https://doi.org/10.5555/3495724.3495883>.

[87] DOSOVITSKIY, A. et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. Disponível em: <https://doi.org/10.48550/arXiv.2010.11929>.

[88] DARBAN, Z. Z. et al. Deep learning for time series anomaly detection: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 57, n. 1, out. 2024. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3691338>.

[89] HUANG, H. et al. Deep learning advancements in anomaly detection: A comprehensive survey. *IEEE Internet of Things Journal*, v. 12, n. 21, p. 44318–44342, 2025. Disponível em: <https://doi.org/10.1109/JIOT.2025.3585884>.

[90] LI, C. et al. T-vae: Transformer-based variational autoencoder for perceiving anomalies in multivariate time series data. *Expert Systems*, v. 42, n. 7, p. e70078, 2025. E70078 EXSY-Jul-24-2883.R2. Disponível em: <https://doi.org/10.1111/exsy.70078>.

[91] MANOCCHIO, L. D. et al. Flowtransformer: A transformer framework for flow-based network intrusion detection systems. *Expert Systems with Applications*, v. 241, p. 122564, 2024. ISSN 0957-4174. Disponível em: <https://doi.org/10.1016/j.eswa.2023.122564>.

[92] MANOCCHIO, L. D.; LAYEGHY, S.; PORTMANN, M. Flowtransformer: A flexible python framework for flow-based network data analysis. *Software Impacts*, v. 22, p. 100702, 2024. ISSN 2665-9638. Disponível em: <https://doi.org/10.1016/j.simpa.2024.100702>.

[93] WANG, Y. et al. Mtrc: A self-supervised network intrusion detection framework based on multiple transformers enabled data reconstruction with contrastive learning. *Journal of Network and Computer Applications*, v. 243, p. 104300, 2025. ISSN 1084-8045. Disponível em: <https://doi.org/10.1016/j.jnca.2025.104300>.

[94] AKKEPALLI, S.; K, S. Copula entropy regularization transformer with c2 variational autoencoder and fine-tuned hybrid dl model for network intrusion detection. *Telematics and Informatics Reports*, v. 17, p. 100182, 2025. ISSN 2772-5030. Disponível em: <https://doi.org/10.1016/j.teler.2024.100182>.

[95] WANG, W. et al. Robust unsupervised network intrusion detection with self-supervised masked context reconstruction. *Computers Security*, v. 128, p. 103131,

2023. ISSN 0167-4048. Disponível em: <https://doi.org/10.1016/j.cose.2023.103131>.

[96] KUMMEROW, A. et al. Unsupervised anomaly detection and explanation in network traffic with transformers. *Electronics*, v. 13, n. 22, 2024. ISSN 2079-9292. Disponível em: <https://doi.org/10.3390/electronics13224570>.

[97] GHADERMAZI, J. et al. Gtae-ids: Graph transformer-based autoencoder framework for real-time network intrusion detection. *IEEE Transactions on Information Forensics and Security*, v. 20, p. 4026–4041, 2025. Disponível em: <https://doi.org/10.1109/TIFS.2025.3557741>.

[98] VU, L. et al. Learning latent distribution for distinguishing network traffic in intrusion detection system. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. [s.n.], 2019. p. 1–6. Disponível em: <https://doi.org/10.1109/ICC.2019.8762015>.

[99] CHEN, Z.; YE, C. Network traffic intrusion detection by convolutional variational self-encoder incorporating improved convolutional attention. In: *Proceedings of the 2025 4th International Conference on Big Data, Information and Computer Network*. New York, NY, USA: Association for Computing Machinery, 2025. (BDICN '25), p. 710–715. ISBN 9798400712425. Disponível em: <https://doi.org/10.1145/3727353.3727467>.

[100] Orion Research Group. *Datasets used in Publications*. <https://www.uel.br/grupos/orion/datasets.html>. Acessado em 28-10-2025.

[101] SHANNON, C. E. Prediction and entropy of printed english. *Bell System Technical Journal*, v. 30, n. 1, p. 50–64, January 1951. Disponível em: <https://doi.org/10.1002/j.1538-7305.1951.tb01366.x>.

[102] da Silva Ruffo, V. G. et al. Generative adversarial networks to detect intrusion and anomaly in ip flow-based networks. *Future Generation Computer Systems*, v. 163, p. 107531, 2025. ISSN 0167-739X. Disponível em: <https://doi.org/10.1016/j.future.2024.107531>.

[103] PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, JMLR.org, v. 12, n. null, p. 2825–2830, nov. 2011. ISSN 1532-4435. Disponível em: <https://doi.org/10.5555/1953048.2078195>.

[104] LIMA, F. T.; SOUZA, V. M. A large comparison of normalization methods on time series. *Big Data Research*, v. 34, p. 100407, 2023. ISSN 2214-5796. Disponível em: <https://doi.org/10.1016/j.bdr.2023.100407>.

[105] da Silva Ruffo, V. G. et al. Generative adversarial networks to detect intrusion and anomaly in ip flow-based networks. *Future Generation Computer Systems*, v. 163, p. 107531, 2025. ISSN 0167-739X. Disponível em: <https://doi.org/10.1016/j.future.2024.107531>.

[106] ZACARON, A. M. et al. Generative adversarial network models for anomaly detection in Software-Defined networks. *Journal of Network and Systems Management*, v. 32, n. 4, p. 93, set. 2024. Disponível em: <https://doi.org/10.1007/s10922-024-09867-z>.