



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Lamego

Deteção de Intrusões e Ataques em Redes Informáticas

Paulo Francisco Teixeira Castro

Janeiro, 2026



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Lamego

Deteção de Intrusões e Ataques em Redes Informáticas

Paulo Francisco Teixeira Castro

Trabalho de Projeto

Mestrado em Tecnologias de Informação e Automação

Trabalho efetuado sob a orientação de

Professor Doutor Fernando Miguel Soares Mamede dos Santos

Professor Doutor Pedro Filipe Antunes Lopes

Janeiro, 2026



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Lamego

Deteção de Intrusões e Ataques em Redes Informáticas

Paulo Francisco Teixeira Castro

Trabalho de Projeto

Mestrado em Tecnologias de Informação e Automação

Trabalho efetuado sob orientação de:

Professor Doutor Fernando Miguel Soares Mamede dos Santos

Professor Doutor Pedro Filipe Antunes Lopes

Lamego, 2026

“Security is not a product, but a process.”

Bruce Schneier

AGRADECIMENTOS

À semelhança de uma casa, que não se sustenta apenas pelas paredes visíveis, mas sobretudo pelos pilares que a suportam em silêncio, firmes e constantes, resistindo ao peso do tempo e das dificuldades. São eles que garantem estabilidade quando o solo treme e que permitem que a construção se mantenha de pé mesmo nas fases mais frágeis. Desta forma, a realização deste trabalho, foi-se construindo com os pilares que me sustentaram nas fases complicadas, mas que nunca me deixaram desabar.

Aos meus pais, Fernando Castro e Luísa Castro, os meus pilares desde o dia do meu nascimento, o meu apoio nos momentos de fragilidade, a força quando o cansaço apertou e a segurança quando o caminho pareceu incerto. A vocês devo muito mais do que consigo expressar, mas deixo-vos aqui o meu eterno agradecimento por terem feito por mim, todos os possíveis e impossíveis.

Aos dois pilares fundamentais na construção deste trabalho, o meu orientador Prof. Dr. Fernando dos Santos e ao meu coorientador Prof. Dr. Pedro Lopes, agradeço a orientação, a disponibilidade e a confiança ao longo de todo o processo. O vosso acompanhamento foi essencial para a qualidade científica deste trabalho.

Agradeço igualmente ao Instituto Politécnico de Viseu, cuja colaboração e disponibilização dos dados necessários constituíram um pilar institucional indispensável para a concretização deste trabalho.

Os meus amigos foram igualmente pilares importantes, sustentando-me nos momentos de maior desgaste, guiando-me quando me senti perdido e oferecendo palavras de incentivo quando mais precisei. Um agradecimento especial a Francisco Gouveia e Rita Cunha, que nunca deixaram faltar apoio.

Não posso deixar de agradecer a três pilares muito especiais, os meus anjos da guarda que, embora já não estejam fisicamente presentes, sei que continuam a olhar por mim. A vocês, Odília Cró, Maria José e Fernanda Vasconcelos, este trabalho também é vosso, e é merecido que os vossos nomes aqui constem.

Por fim, agradeço a todos aqueles que, direta ou indiretamente, contribuíram para esta construção. Cada gesto, cada palavra e cada apoio ajudaram a tornar este percurso mais leve. A todos, deixo o meu profundo reconhecimento e gratidão.

RESUMO

A detecção eficaz de intrusões em redes informáticas constitui um dos desafios mais persistente e crítico da sociedade digital contemporânea. Dada a complexidade das redes modernas, torna-se um desafio constante garantir que o tráfego e os dados armazenados fiquem protegidos contra a intercetação ilícita ou o acesso indevido por parte de terceiros. Apesar dos avanços na criptografia, a proteção da privacidade e da integridade dos dados continua a depender da capacidade de monitorização dos prestadores de serviços e das organizações.

Atualmente, as soluções de segurança mais difundidas baseiam-se em assinaturas de ataques conhecidos ou regras estáticas, exigindo uma supervisão constante por parte de administradores de sistemas. Estas técnicas apresentam a desvantagem inerente de gerarem um elevado volume de falsos positivos e de serem ineficazes perante ameaças novas ou variantes comportamentais, o que limita severamente a sua eficiência operacional. Surge, por isso, a necessidade premente de aprofundar o estudo em sistemas autodidatas que possam realizar uma análise inteligente dos dados, distinguindo com maior precisão o tráfego legítimo de ocorrências anormais.

O presente trabalho propõe o desenvolvimento e a avaliação de um motor de detecção inteligente, utilizando como base de estudo registos reais de tráfego de uma instituição de ensino superior. O foco incide na aplicação de algoritmos avançados de *Machine Learning* e *Deep Learning* de modo a mitigar o impacto de falsos alertas que sobrecarregam os analistas de segurança. Através desta abordagem, pretende-se não apenas identificar anomalias, mas também desenhar uma arquitetura conceptual que permita a integração destas deteções em ecossistemas de monitorização, promovendo uma defesa mais resiliente e autónoma.

Palavras-chave: Deteção de Intrusões; Machine Learning; Deep Learning; Segurança de Redes; Anomalias.

ABSTRACT

An efficient intrusion detection in computer networks remains one of the most persistent and critical challenges of today's digital society. Given the complexity of modern networks, it is a constant challenge to ensure that traffic and stored data are protected against illegal interception or unauthorized access. Despite advances in encryption, protecting privacy and data integrity still depends on the monitoring capacity of service providers and organizations.

Currently, the most common security solutions are based on signatures of known attacks or static rules, requiring constant supervision by system administrators. These techniques have the inherent disadvantage of generating a high volume of false positives and being ineffective against new threats or behavioral variations, which severely limits their operational efficiency. Therefore, there is an urgent need to study self-learning systems that can perform an intelligent analysis of data, accurately separating legitimate traffic from abnormal events.

This research aims to develop and evaluate an intelligent detection engine, based on real traffic logs from a higher education institution. The focus is on applying advanced Machine Learning and Deep Learning algorithms to reduce the impact of false alerts that overwhelm security analysts. This approach aims not only to identify anomalies but also to design a conceptual architecture to integrate these detections into monitoring ecosystems, promoting a more resilient and autonomous defence.

Keywords: Intrusion Detection; Machine Learning; Deep Learning; Network Security; Anomalies.

ÍNDICE GERAL

1	Revisão da literatura.....	5
1.1	Redes Informáticas	5
1.2	Enquadramento Histórico	5
1.3	Segurança em Redes Informáticas.....	6
1.3.1	Pilares da Segurança.....	6
1.4	Vulnerabilidades, Ataques, Riscos e Defesas	7
1.5	Ameaças e Ataques.....	7
1.5.1	Ameaças Externas e Internas	7
1.6	Sistemas de Detecção de Intrusões.....	9
1.6.1	Arquitetura dos IDS.....	10
1.6.2	Classificação dos IDS	11
1.7	Sistemas de Monitorização de Segurança de Rede	16
1.7.1	Snort.....	16
1.7.2	Suricata	17
1.7.3	OSSEC.....	18
1.8	Logs	19
1.8.1	Ferramentas de Logs.....	21
1.9	SIEM.....	23
1.10	Modelos de Inteligência Artificial	26
1.10.1	Algoritmos de Machine Learning Supervisionado	26
1.10.2	Algoritmos de Machine Learning Não Supervisionado	33
1.10.3	Algoritmos de Deep Learning Supervisionado	39
1.10.4	Algoritmos de Deep Learning Não Supervisionado.....	42
1.11	Análise dos Modelos de Inteligência Artificial	47
1.11.1	Análise Comparativa dos modelos Machine Learning Supervisionado ..	49
1.11.2	Análise Comparativa dos modelos Machine Learning Não Supervisionado	50
1.11.3	Análise Comparativa dos modelos Deep Learning Supervisionado	51
1.11.4	Análise Comparativa dos modelos Deep Learning Não Supervisionado	52
1.12	Métricas de Avaliação de Desempenho	52
1.13	Trabalhos Relacionados.....	55
1.13.1	Fundamentação e Desafios	55

1.13.2	Modelos de Ensemble.....	55
1.13.3	Modelos de Deep Learning Sequencial	56
1.13.4	Síntese e Contributos do Estudo	56
2	Metodologia E resultados.....	57
2.1	Introdução	57
2.2	Ambiente Computacional	57
2.3	Pré-processamento dos Dados	58
2.4	Avaliação dos Modelos.....	61
2.4.1	Machine Learning Supervisionado	61
2.4.2	Machine Learning Não Supervisionado	71
2.4.3	Deep Learning Supervisionado	77
2.4.4	Deep Learning Não Supervisionado	87
2.4.5	Síntese da Avaliação de Desempenho e Implicações Metodológicas.....	92
2.5	Simulação de Cenários de Ataque e Análise de Desempenho.....	93
2.5.1	Simulação de DDoS.....	93
2.5.2	Simulação de Exfiltração de Dados.....	95
2.5.3	Simulação de SQL Injection	96
2.5.4	Análise da Simulação	97
2.6	Operacionalização e Integração em sistemas de Segurança	98
2.6.1	Enquadramento do Motor de IA na Arquitetura IDS.....	99
2.6.2	Integração com Ecossistemas SIEM.....	99
2.6.3	Viabilidade e Desafios de Implementação.....	100

ÍNDICE DE TABELAS

Tabela 1.1: Tabela Comparativa - HIDS, NIDS, Híbrido	14
Tabela 1.2: Comparação dos modelos Machine Learning Supervisionado.....	49
Tabela 1.3: Comparação dos modelos Machine Learning Não Supervisionado	50
Tabela 1.4: Comparação dos modelos Deep Learning Supervisionado	51
Tabela 1.5: Comparação de modelos Deep Learning Não Supervisionado	52
Tabela 2.1: Comparação dos resultados dos modelos Machine Learning Supervisionado	68
Tabela 2.2: Desempenho comparativo entre o modelo XGBoost base e otimizado.....	70
Tabela 2.3: Resultados comparativos do desempenho do modelo Isolation Forest com diferentes níveis de ‘contamination’	72
Tabela 2.4: Resultados comparativos de desempenho do modelo K-means com números de clusters distintos.....	74
Tabela 2.5: Resultados comparativos de desempenho do modelo DBSCAN em configurações ‘min_samples’	76
Tabela 2.6: Comparação dos resultados dos modelos Deep Learning Supervisionados	84
Tabela 2.7: Desempenho comparativo entre o modelo GRU base e otimizado	86
Tabela 2.8: Simulação de DDoS em três intensidades de injeção distintas.....	94
Tabela 2.9: Simulação de Exfiltração de Dados em três intensidades de injeção distintas	95
Tabela 2.10: Simulação de SQL Injection em três intensidades de injeção distintas	97

ÍNDICE DE FIGURAS

Figura 1.1: Arquitetura funcional de um IDS.....	11
Figura 1.2: Arquitetura de IDS baseado em Host.....	12
Figura 1.3: Arquitetura de IDS baseado em Network	13
Figura 1.4: Arquitetura Snort	17
Figura 1.5: Arquitetura Suricata	18
Figura 1.6: Arquitetura OSSEC.....	19
Figura 1.7: Fontes de Eventos	21
Figura 1.8: Arquitetura básica do SIEM.....	24
Figura 1.9: Matriz de Confusão.....	53
Figura 2.1: Random Forest – Relatório de Classificação.....	62
Figura 2.2: Random Forest – Matriz de Confusão	63
Figura 2.3: Random Forest – Curva Precision_Recall.....	63
Figura 2.4: XGBoost – Relatório de Classificação	64
Figura 2.5: XGBoost – Matriz de Confusão	65
Figura 2.6: XGBoost - Curva Precision-Recall.....	65
Figura 2.7: SGD - Relatório de Classificação	67
Figura 2.8: SGD – Matriz de Confusão.....	67
Figura 2.9: SGD - Curva Precision-Recall.....	67
Figura 2.10: Visualização do agrupamento K-means com PCA	74
Figura 2.11: LSTM – Relatório de Classificação	79
Figura 2.12: LSTM – Matriz de Confusão	79
Figura 2.13: LSTM – Curva Precision-Recall	79
Figura 2.14: GRU – Relatório de Classificação	81
Figura 2.15: GRU – Matriz de Confusão	82
Figura 2.16: GRU – Curva Precision-Recall.....	82
Figura 2.17: VAE – Relatório de Classificação.....	88
Figura 2.18: VAE – Matriz de Confusão	88
Figura 2.19: VAE – Curva Precision-Recall	88
Figura 2.20: DeepLog – Relatório de Classificação	90
Figura 2.21: Deeplog - Matriz de Confusão.....	91

Figura 2.22: DeepLog - Curva Precision-Recall.....	91
---	-----------

LISTA DE SIGLAS

IDS	- Intrusion Detection System
IPS	- Intrusion Prevention System
DDoS	- Distributed Denial-of-Service
IA	- Inteligência Artificial
WI-Fi	- Wireless Fidelity
IoT	- Internet of Things
ARPA	- Advanced Research Projects Agency
HIDS	- Host Intrusion Detection Systems
NIDS	- Network Intrusion Detection Systems
SO	- Sistema Operativo
HIPS	- Host Intrusion Prevention System
NIPS	- Network Intrusion Prevention System
IDPS	- Intrusion Detection and Prevention System
VPN	- Virtual Private Network
HTTP	- HyperText Transfer Protocol
DNS	-Domain Name System
FTP	- File Transfer Protocol
SIEM	- Security Information and Event Management
OSSEC	- Open Source Security Information and Event Management
TI	- Tecnologias de Informação
API	- Application Programming Interface
UEBA	- User and Entity Behavior Analytics

RGPD	- Regulamento Geral de Proteção de Dados
XGBoost	- Extreme Gradient Boosting
SVM	- Support Vector Machine
DBSCAN	- Density-Based Spatial Clustering of Applications with Noise
LSTM	- Long Short-Term Memory
RNN	- Recurrent Neural Network
GRU	- Gated Recurrent Unit
BERT	- Bidirectional Encoder Representations from Transformers
PLN	- Processamento de Linguagem Natural
MLM	- Masked Language Model
NSP	- Next Sentence Prediction
LogBERT	- Log Bidirectional Encoder Representations from Transformers
MLKP	- Masked Log Key Prediction
VHM	- Volume of Hypersphere Minimization
VAE	- Variational Autoencoder
ELBO	- Evidence Lower Bound
MSE	- Mean Squared Error
VP	- Verdadeiro Positivo
FN	- Falso Negativo
FP	- Falso Positivo
VN	- Verdadeiro Negativo
TFN	- Taxa de Falsos Negativos
TFP	- Taxa de Falsos Positivos
CSV	- Comma Separated Values

NaN	-Not a Number
SMO	- Sequential Minimal Optimization
SVC	- Support Vector Classifier
SGD	- Stochastic Gradient Descent
PCA	- Principal Component Analysis
CIDF	- Common Intrusion Detection Framework
JSON	- JavaScript Object Notation
XAI	- Explainable Artificial Intelligence

INTRODUÇÃO

Preâmbulo

As redes informáticas revolucionaram a forma como comunicamos e partilhamos informações trazendo benefícios inegáveis, como a comunicação instantânea e a partilha de dados em tempo real. No entanto, com o aumento exponencial do número de redes, tem-se verificado um crescimento considerável de ataques, expondo os sistemas a uma variedade crescente de ameaças e vulnerabilidades. Qualquer rede informática pode ser vulnerável a ataques, desde as de menor dimensão, como as redes domésticas, até às mais tecnologicamente avançadas e desenvolvidas [1].

Neste cenário, torna-se imperativo adotar estratégias eficazes de segurança, como o uso de *firewalls*, criptografia, segmentação de rede e, sobretudo, mecanismos especializados na deteção e prevenção de intrusões. Entre os métodos comumente utilizados, destacam-se *Intrusion Detection System* (IDS) e *Intrusion Prevention System* (IPS). Os métodos referidos anteriormente, desempenham um papel fundamental na monitorização contínua do tráfego da rede, permitindo identificar e responder a comportamentos anómalos ou maliciosos, funcionando assim como uma camada adicional de defesa contra ciberataques.

As ameaças podem assumir diversas formas, desde a ação de *malwares*, *Distributed Denial-of-Service* (DDoS), *phishing*, que visam comprometer dados sensíveis ou até paralisar infraestruturas críticas [2].

Neste contexto, o presente trabalho propõe-se abordar a deteção de intrusões e ataques em redes informáticas com base na análise de *logs*. Com o uso de técnicas avançadas de Inteligência Artificial (IA), abre-se caminho para soluções mais inteligentes e adaptativas, capazes de identificar padrões complexos e antecipar comportamentos maliciosos que, de outra forma, poderiam passar despercebidos.

Esta abordagem pretende, não apenas dar uma resposta reativa a incidentes, mas também uma mitigação proativa das ameaças, contribuindo para redes mais resilientes e seguras.

Motivação

Atualmente, as redes informáticas encontram-se em diversos setores, desde a saúde à educação. Esta ubiquidade e interconectividade, embora tragam vários benefícios, introduzem também novos riscos e vulnerabilidades que colocam desafios crescentes à cibersegurança. Deste modo, organizações de vários setores enfrentam a necessidade de adaptação dos seus sistemas face à complexidade e frequência dos ataques.

Os ataques às redes informáticas são motivados por uma variedade de fatores. Entre os objetivos mais comuns está o acesso indevido a dados sensíveis, nomeadamente, informações pessoais, credenciais bancárias ou registos clínicos, com o intuito de obter lucros com a sua venda ou exploração. Adicionalmente, ataques como os de DDoS resultam na indisponibilidade de sistemas, causando prejuízos significativos à continuidade operacional de empresas e entidades públicas.

Face a este panorama, torna-se evidente que soluções tradicionais de segurança, como as *firewalls* acabam por não ser suficientes para responder à complexidade e dinamismo das ameaças atuais, o que leva à necessidade de utilizar sistemas IDS/IPS para preencher esta lacuna.

Objetivos

Este trabalho insere-se no contexto académico e pretende contribuir para a investigação e desenvolvimento de soluções baseadas em IA aplicadas à cibersegurança, com especial enfoque na deteção de intrusões e ataques em redes informáticas através da análise de *logs*. Parte-se do princípio de que os registos de eventos constituem uma fonte valiosa de informação para identificar comportamentos suspeitos ou maliciosos, representando um pilar essencial nas abordagens modernas de defesas suportadas por Inteligência Artificial.

Neste sentido, o principal objetivo desta investigação concerne na avaliação comparativa dos diversos modelos de IA, nomeadamente algoritmos de *Machine Learning* e *Deep Learning*, aplicados à análise de dados de *logs* de rede. Pretende-se determinar quais os modelos que demonstram melhor desempenho na identificação de padrões anómalos e compreender de que forma as decisões metodológicas influenciam os resultados obtidos.

Entre os objetivos específicos destacam-se:

- Estruturar e preparar os dados provenientes de *logs* de rede, assegurando a sua integridade, anonimização e qualidade para análise;
- Aplicar técnicas de pré-processamento, incluindo normalização, codificação e engenharia de características, de modo a garantir a compatibilidade com os diferentes algoritmos;
- Implementar e avaliar diversos modelos de *Machine Learning* e *Deep Learning*, supervisionados e não supervisionados;
- Comparar o desempenho dos modelos a partir de métricas apropriadas, com o foco na minimização dos falsos negativos e na fiabilidade da deteção;
- Simular cenários de ataque para avaliar a capacidade de generalização e resposta dos modelos;
- Analisar as limitações observadas nos resultados e discutir possíveis melhorias metodológicas a considerar em trabalhos futuros;
- Discutir a viabilidade técnica e conceptual de integração das soluções desenvolvidas com sistemas de segurança existentes, nomeadamente IDS e plataformas SIEM, promovendo uma monitorização automatizada e inteligente de eventos.

Com a realização destes objetivos, pretende-se fundamentar o potencial das abordagens baseadas em IA na deteção de anomalias e demonstrar a sua relevância prática na defesa de infraestruturas computacionais críticas.

Estrutura

O presente trabalho encontra-se organizada em dois capítulos principais, para além das considerações finais, estruturados de forma a proporcionar uma progressão lógica entre o enquadramento teórico, a metodologia adotada e a análise dos resultados obtidos.

O primeiro capítulo é dedicado à revisão da literatura, tendo como objetivo estabelecer o enquadramento teórico necessário à compreensão do tema em estudo. Neste capítulo são introduzidos e discutidos os conceitos fundamentais relacionados com as redes informáticas, a sua evolução histórica e os princípios da segurança em redes. São ainda abordadas as principais ameaças, vulnerabilidades e tipos de ataques, bem como os

sistemas de detecção e prevenção de intrusões. Adicionalmente, é apresentada uma análise detalhada dos *logs* enquanto fonte essencial de informação para a detecção de intrusões, bem como das principais ferramentas de monitorização e gestão de eventos de segurança. O capítulo culmina com a apresentação e análise dos modelos de Inteligência Artificial, incluindo algoritmos de *Machine Learning* e *Deep Learning*, supervisionados e não supervisionados, que constituem a base conceptual para o desenvolvimento experimental do trabalho.

O segundo capítulo centra-se na metodologia e nos resultados, descrevendo de forma detalhada o processo experimental desenvolvido ao longo do trabalho. Inicialmente, é apresentado o ambiente computacional utilizado, seguido das etapas de pré-processamento dos dados de *logs*. Posteriormente, são descritos e avaliados os diferentes modelos de *Machine Learning* e *Deep Learning* aplicados, recorrendo a métricas de desempenho adequadas à detecção de intrusões. Este capítulo inclui ainda a simulação de cenários reais de ataque, como ataques de DDoS, exfiltração de dados e *SQL Injection*, permitindo avaliar a capacidade de generalização e robustez dos modelos implementados. A análise crítica dos resultados obtidos e a respetiva discussão encerram este capítulo.

Por fim, este trabalho termina com as conclusões, onde são sintetizadas as principais contribuições do trabalho, discutidas as limitações identificadas e apresentadas propostas de trabalho futuro. Esta secção visa refletir sobre a aplicabilidade prática das soluções desenvolvidas, bem como sobre o potencial de integração das abordagens baseadas em IA em sistemas de segurança existentes.

1 REVISÃO DA LITERATURA

1.1 Redes Informáticas

As Redes Informáticas são uma estrutura de comunicação que interligam computadores, servidores, dispositivos móveis, entre outros, permitindo a troca de informações e recursos entre si, de maneira rápida, segura e eficiente. É importante salientar que as interconexões podem ser realizadas através do meio físico (cabos de rede) ou através de *Wireless Fidelity* (Wi-Fi).

Na atualidade, as utilizações de redes desempenham um papel crucial na transformação digital e no funcionamento de diversos setores, uma vez que permite aumentar a produtividade e a diminuição de custos. No contexto educativo, as redes informáticas revolucionaram o sistema de ensino, como por exemplo o Moodle, que possibilita a partilha de recursos entre professores e alunos.

Assim sendo, as redes informáticas estão responsáveis por otimizar processos, facilitar a comunicação em tempo real, automatizar tarefas e integrar tecnologias avançadas, como *Internet of Things* (IoT) e a Inteligência Artificial.

Em síntese, as redes informáticas são a base que sustenta a conectividade e impulsiona a inovação no mundo contemporâneo.

1.2 Enquadramento Histórico

Os meios de comunicação, com o passar dos séculos foram sofrendo diversas modificações. Por volta do ano de 1960, a rede telefónica era o que dominava o mundo naquela época. Estamos cientes de que vivemos num mundo que está em constante mutação e inovação tecnológica, que parte da realização constante de diversos estudos e pesquisas.

Durante a guerra fria, surgiu pelo Departamento de Defesa dos Estados Unidos através da *Advanced Research Projects Agency* (ARPA) a necessidade de criar uma rede informática que fosse resistente a falhas e ataques, garantindo a comunicação entre diferentes centros de pesquisa e militares num cenário de guerra. Assim sendo, levou a que vários investigadores fossem incentivados a realizar os primeiros estudos sobre a comutação de pacotes e idealizar a construção de uma

grande rede que mantivesse as comunicações seguras, levando aos primórdios das redes informáticas com o surgimento da ARPANET por volta do ano 1970 [3].

Com o fim da guerra, a ARPANET deixou de ser relevante para os militares e, portanto, esta tecnologia foi distribuída para universidades e outros investigadores que estenderam a comunicação para outros países, o que consequentemente levou ao que conhecemos hoje como *internet* [4].

1.3 Segurança em Redes Informáticas

A Segurança em Redes Informáticas é constituída por um conjunto de práticas, políticas, ferramentas e mecanismos de segurança implementados para defender sistemas e dados numa rede.

A determinação do nível de segurança a implementar numa determinada rede deve considerar os riscos associados a possíveis falhas de segurança, o envolvimento de custos na adoção de mecanismos de proteção e os benefícios resultantes da minimização destes riscos [5].

1.3.1 Pilares da Segurança

Manter a segurança da rede é essencial para o bom funcionamento da mesma. Deste modo, entre os principais pilares da segurança estão a autenticação, confidencialidade, integridade e disponibilidade.

Estes pilares são complementares entre si, tendo em conta que para garantir a confidencialidade é necessário primeiramente garantir a autenticidade.

A autenticação, como o próprio nome indica, valida a autenticação de um utilizador, dispositivo ou processo. A transmissão e a manipulação de dados sensíveis só estará disponível após a autenticação previamente efetuada, o que garante total confidência.

A integridade, por sua vez, garante que os dados enviados, recebidos ou armazenados não sejam corrompidos ou manipulados. Já a disponibilidade assegura que na eventualidade de ocorrer eventos perturbadores, como cortes de energia, calamidades naturais, acidentes ou ataques, os dados e serviços permanecem operacionais [5] [6].

Assim sendo, o investimento em boas práticas e soluções de segurança é essencial para manter a estabilidade e resiliência da infraestrutura da rede.

1.4 Vulnerabilidades, Ataques, Riscos e Defesas

No contexto da segurança em redes informáticas, entender os conceitos de vulnerabilidades, ameaças, riscos e defesas é indispensável para proteger as redes e sistemas contra possíveis incidentes.

A vulnerabilidade refere-se a um sistema que apresenta fragilidades que podem ser exploradas, o que conseqüentemente poderá levar a um ataque, resultando em danos ou acessos indevidos. Um risco, é o dano que provém da execução de um ataque bem-sucedido.

A defesa consiste no conjunto de políticas e mecanismos desenhados, com o intuito de diminuir as vulnerabilidades de um sistema, detetar, neutralizar e minimizar os riscos resultantes de ataques bem-sucedidos. Uma estratégia de defesa eficaz deve, obrigatoriamente, de satisfazer as três vertentes fundamentais anteriormente enunciadas e não apenas uma delas. Independentemente das medidas de proteção implementadas, é imprescindível manter uma monitorização contínua do estado e do funcionamento dos sistemas, de modo a permitir a deteção precoce de anomalias e comportamentos suspeitos.

Por fim, partindo do princípio de que um ataque foi realizado com sucesso, é fundamental efetuar planos de contingência, que inclua formas de recuperação de bens e serviços para minimizar os danos provenientes do ataque [7].

1.5 Ameaças e Ataques

Os riscos podem ser classificados de diversas formas, dependendo da origem da ameaça e do tipo de ataque realizado. Entre as demais categorias, destacam-se as ameaças internas e externas, que dizem respeito à origem do perigo, e os ataques passivos e ativos, que diferem no modo como os sistemas e a privacidade dos dados podem ser afetados.

1.5.1 Ameaças Externas e Internas

As ameaças internas são riscos que nascem dentro da própria organização, frequentemente causados por funcionários, prestadores de serviços ou antigos colaboradores. Assim sendo, os riscos internos apresentam uma maior dificuldade em ser detetados perante as ameaças externas, pois os operadores possuem acesso aos recursos de uma organização e estão cientes das respetivas medidas de segurança.

Uma ameaça interna não necessita de ser propriamente intencional, porém pode ser fatal para a segurança da rede. Por exemplo, colaboradores podem cometer o erro de responder a uma campanha de *phishing* ou instalar inadvertidamente *malware* [8].

Espionagem, sabotagem, fraude, roubo, são exemplos entre as demais ameaças internas que podem ocorrer em contexto malicioso. Independentemente da designação que possa ser atribuída a estas ações intencionais todas partem do mesmo princípio, roubo de informações confidenciais. Muitas destas ameaças revelam-se partir de operadores com falta de reconhecimento por parte da organização ou por demissão [8] [9].

Relativamente às ameaças externas, estas referem-se a riscos provenientes de entidades que não façam parte de uma organização, como *hackers*, grupos cibercriminosos ou qualquer indivíduo mal-intencionado que pretenda comprometer a segurança de uma rede informática. Ataques baseados em *malwares*, que incluem vírus, *worms*, *trojans*, *ransomware* e *spyware*, é uma das formas mais comuns de ameaças. Estes tipos de ataques são programas introduzidos nos sistemas através de e-mails, páginas *web* comprometidas, entre outros [10].

1.5.1.1 Ataques Ativos e Passivos

Um ataque passivo leva a que um invasor monitorize, intercete ou recolha informações sem causar perturbações diretamente no sistema ou na comunicação. Por outras palavras, o objetivo central passa por espionar a rede na procura de informações relevantes por meio da análise de tráfego. Isto, por sua vez, leva à divulgação de informações críticas em relação à rede ou nós, como a sua topologia, a localização ou até mesmo a identidade de nós importantes [11] [12].

No que diz respeito a um ataque ativo, os invasores intervêm no fluxo normal da informação, procedendo à modificação, injeção ou danificação da mesma. Estes ataques por norma, são facilmente detetados pois envolvem interações diretas com o sistema alvo. Porém, as perturbações que os invasores causam podem ser graves ao ponto de interromper ou degradar significativamente o desempenho da rede, como por exemplo um ataque DDoS [11].

1.5.1.2 Tipos de Ataques

Eavesdropping – Um ataque de espionagem consiste na tentativa de intercetar e aceder a dados transmitidos na rede. O invasor monitoriza o tráfego sem ser detetado, recolhendo informações confidenciais, como dados financeiros ou outros tipos de dados sensíveis [13].

Distributed Denial of Service – Ataques DDoS sobrecarregam a rede com uma série de *bots* deixando o tráfego irregular, o que consequentemente leva à má funcionalidade ou interrupção da rede por um longo período [14].

Malware – É um termo geral a qualquer *software* malicioso projetado a invadir ou danificar sistemas computacionais, redes, etc.

- *Ransomware* – É um tipo *malware* que mantém os dados roubados criptografados. Posto isto, o invasor exige um valor monetário à vítima para fornecer a chave de criptografia necessária para descriptografar os dados [15].
- *Trojan* – Também conhecido como cavalo de Troia, é um *malware* desenvolvido na intenção de se fazer passar por um programa legítimo, ocultando funcionalidades maliciosas [16].
- *Spyware* – É um tipo de *malware* que é instalado num dispositivo sem o conhecimento ou permissão do utilizador, com o objetivo de roubar informações sobre o mesmo [17].

Phishing – É um tipo de ataque que tenta enganar as vítimas para obter informações confidenciais, tais como, senhas, dados bancários ou outras informações pessoais. Este tipo de ataque geralmente ocorre por meio de e-mails, mensagens ou sites enganosos que se fazem passar por uma entidade legítima [18].

Man-in-the-Middle – É um termo geral para quando um invasor intercepta a comunicação entre duas partes sem que estes se apercebam. O objetivo passa por espionar, roubar informações sensíveis ou modificar dados transmitidos [19].

SQL injection – É um ataque que parte da vulnerabilidade de segurança da *web*, permitindo que um invasor consiga aceder à base de dados, procedendo à modificação ou exclusão de informações que se encontravam dentro da mesma [20].

1.6 Sistemas de Detecção de Intrusões

Segundo [7], os sistemas computacionais apresentam-se cada vez mais poderosos, e consequentemente mais complexos de configurar e explorar. Tais fatores contribuem para o aumento de vulnerabilidades nos sistemas.

Deste modo, os IDS consistem em detetar e contrariar intrusões. São sistemas automáticos que analisam o tráfego da rede em tempo real e detetam tentativas não autorizadas de acesso à infraestrutura lógica [21].

O objetivo primordial dos Sistemas de Detecção de Intrusões passa pela recolha de dados, e através dos mesmos, identificar padrões suspeitos ou atividades anómalas que possam indicar tentativas de acesso não autorizado, ataques cibernéticos ou violações de segurança dentro de uma rede ou sistema.

Apesar de este mecanismo de defesa ter sido rapidamente adotado, a sua utilização real torna-se complexa, levando a ocorrência de erros de deteção. Diz-se que ocorre um falso positivo quando um IDS conclui de forma incorreta que existe uma intrusão. Contrariamente, diz-se que ocorre um falso negativo quando um IDS não deteta uma intrusão. As taxas de erro dos IDS geralmente são difíceis de controlar e são críticas para a sua utilidade. Estes sistemas perdem a credibilidade com excesso de falsos positivos, e torna-se inúteis com falsos negativos em demasia [7].

1.6.1 Arquitetura dos IDS

A arquitetura funcional de um IDS, apresentada na Figura 1.1, é constituída por quatro componentes, sendo eles: Evento (*E boxes*), Análise (*A boxes*), Armazenamento (*D boxes*), Contramedida (*C boxes*).

As *E boxes* intercetam as interações do sistema com o objetivo de capturar eventos de baixo nível ou elementares, como por exemplo, datagramas, sessões realizadas por utilizadores, alterações de ficheiros, entre outros. Estes eventos são enviados para *A boxes*, que irão fundir, correlacionar e interpretar as informações recolhidas pelas *E boxes*, na procura de ações maliciosas.

Na eventualidade de se verificar uma ação maliciosa, as *C boxes* estão responsáveis por reagir ao ataque, seja ao criar um alarme, tomar medidas de proteção como a reconfiguração de *firewalls*, ações de sobrevivências, como isolar o sistema atacado e ativar sistemas alternativos. Todos os eventos recolhidos por *E boxes* ou produzidos por *A boxes* são armazenados em *D boxes*.

A recolha de eventos pelas *E boxes* podem ser realizadas sob dois modos, denominados de *push* e *pull*. Relativamente ao modelo *push*, a *E box* envia eventos quando ocorrem, permitindo que as *A boxes* operem em tempo real. No que diz respeito ao modelo *pull* é efetuado um pedido explícito às *E boxes*, para que estas analisem

determinados dados e produzem eventos, tornando-se o mais adequado para uma análise precedente a intrusões [7].

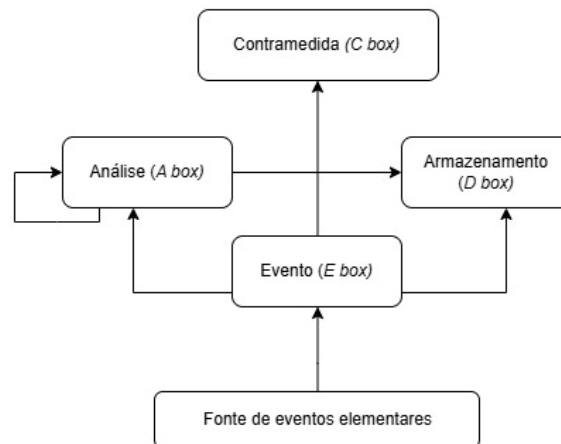


Figura 1.1: Arquitetura funcional de um IDS

Nota: Adaptado de [7]

1.6.2 Classificação dos IDS

Atualmente existe diversos tipos de Sistemas de Detecção de Intrusões, que podem ser classificados pelas suas características funcionais, tais como o seu Método de Detecção, Fonte de Eventos, Instante de Detecção, Reatividade e o seu Tipo de Análise.

1.6.2.1 Métodos de Detecção

No que diz respeito aos métodos de deteção, existem apenas duas variantes. O método baseado em conhecimento, também designado por método em assinaturas, que analisa a atividade do sistema na procura de padrões de ataque ou intrusão conhecidas.

Este método geralmente é eficiente e permite que o IDS tenha uma taxa reduzida de falsos positivos, contudo, é indispensável que o sistema esteja bem configurado, evitando o uso excessivo de assinaturas. No entanto, a sua principal limitação é que apenas deteta problemas previamente conhecidos, baseado nas assinaturas que possui. Por este facto, pode originar uma taxa elevada de falsos negativos.

O método baseado em comportamento, também conhecido por método de deteção de anomalias, deteta desvios à normalidade no comportamento do sistema computacional. Por ser um método que utiliza técnicas de análise estatística e de inteligência artificial, é necessário que o IDS seja treinado a compreender o comportamento “normal” do sistema, e assim, procedendo à deteção de novas formas de ataque e de penetração.

Este método também tem as suas desvantagens, nomeadamente, a dificuldade de definir o comportamento “normal” do sistema, pois pode variar com o tempo ou com a

utilização de diferentes utilizadores. Por fim, o grau de sensibilidade torna-se desvantajoso em relação a desvios da normalidade, no sentido de que, se o IDS for muito sensível poderá produzir falsos positivos em demasia e consequentemente leva à inércia dos seus operadores. No entanto, se o Sistema de Detecção de Intrusões for pouco sensível poderá não cumprir as suas funções. Deste modo, o ideal é encontrar um meio termo, apesar de ser uma questão crítica e complexa de gerir [7].

1.6.2.2 Fontes de Eventos

Os IDS usam geralmente duas fontes de eventos: máquinas ou redes.

Os *Host Intrusion Detection Systems* (HIDS), ilustrado na Figura 1.2, fazem a recolha de evento em máquinas, ou seja, monitorizam o estado dos vários componentes de cada máquina, nomeadamente, *hardware*, núcleo do sistema operativo, ficheiros de registos de operação (*logs*) do sistema ou de servidores, entre outros.

Na prática o Sistema de Detecção de Intrusões baseado em *Host* identifica as atividades anormais do intruso ou comportamento nocivo, emitindo alertas para o administrador da rede. Deste modo, a principal vantagem desta fonte de evento é pelo facto de estar mais próximo do local onde é realizada a intrusão, e por sua vez, facilita na recolha de indícios da sua ocorrência. Contudo, na existência de diversas máquinas com diferentes tipos de sistemas operativos ou configurações, a recolha destes eventos tem de ser configurada individualmente tornando-se numa tarefa complexa. Por conseguinte, não é possível verificar ataques que sejam realizados noutras máquinas ou HIDS de outras máquinas da mesma organização [7] [22].

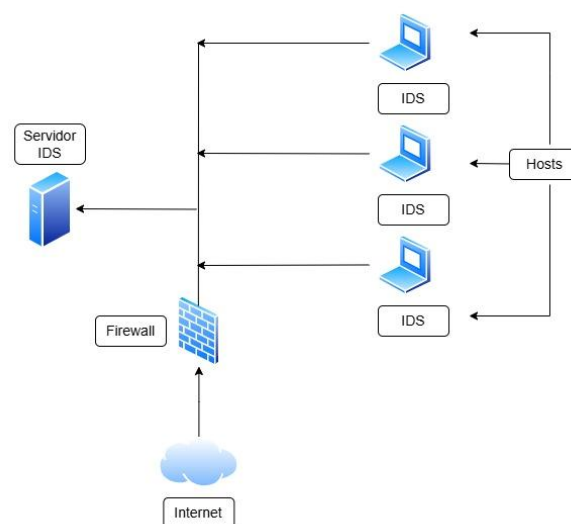


Figura 1.2: Arquitetura de IDS baseado em *Host*

Nota: Adaptado de [7]

Os *Network Intrusion Detection Systems* (NIDS), destacado na Figura 1.3 [23], é projetado para supervisionar e analisar as interações entre dispositivos em tempo real, oferecendo uma visão bem mais ampla do tráfego da rede, se compararmos com os HIDS.

Estes Sistemas de Detecção de Intrusão em Redes são compostos por diversos componentes, tais como, sensores de rede, que podem ser dispositivos normais ou equipamentos de interligação (*gateways* ou *routers*) que fazem a recolha e a filtragem de pacotes, mecanismos de alerta, interface de utilizador, sistema de registo (*logs*) que armazena os eventos detetados para auditoria, análise e ajustes na política de segurança.

Em comparação com os HIDS, esta fonte de evento apresenta muito mais vantagens, e, portanto, teve uma rápida adoção. A sua simplicidade de exploração, deve-se a um número reduzido de sensores que opera na rede e não em dispositivos individuais, podendo assim utilizar máquinas e Sistemas Operativos (SO) heterogéneos. A possibilidade de ter máquinas dedicadas a sensores, diminui o impacto negativo no desempenho da rede e não sobrecarrega máquinas destinadas a outros propósitos.

Por fim, os NIDS apresentam algumas desvantagens como, a ineficiência contra ameaças internas e na incapacidade de analisar pacotes que estejam criptografados [7] [24].

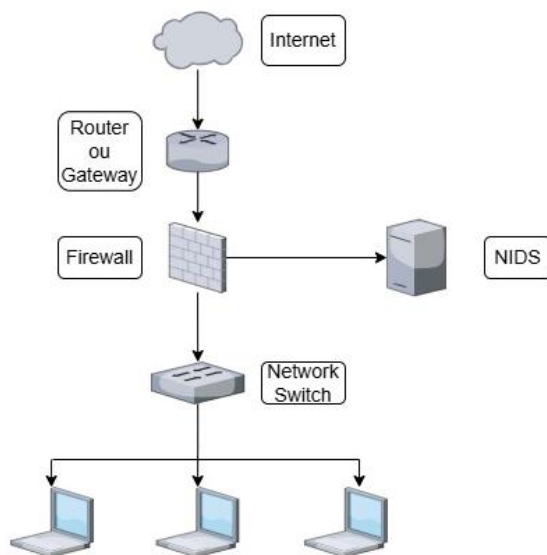


Figura 1.3: Arquitetura de IDS baseado em *Network*

Nota: Adaptado de [23]

Na busca de uma arquitetura mais segura, conforme evidenciado na Tabela 1.1, o ideal será a implementação da combinação entre as duas fontes de eventos referidas anteriormente, denominada Híbrido.

De acordo com [25], cada uma das tecnologias possui características que, quando integradas, suprem algumas deficiências presentes nas implementações individuais, oferecendo uma proteção mais abrangente contra ameaças cibernéticas.

Tabela 1.1: Tabela Comparativa - HIDS, NIDS, Híbrido

Características	HIDS	NIDS	HÍBRIDO
Monitorização	Atividades no <i>host</i>	Tráfego na rede	Redes e <i>Hosts</i>
Deteção de ataques internos	Sim	Não	Sim
Deteção de ataques externos	Limitada	Sim	Sim
Fontes dos <i>logs</i>	Processos; Acessos; Alterações em Ficheiros	Pacotes de rede; IP's; Portas; Protocolos	Combinação dos <i>logs</i> de NIDS e HIDS
Visibilidade sobre tráfego criptografado	Sim	Não	Sim
Implementação	Moderada	Fácil	Complexa
Precisão de deteção	Alta	Média	Muito alta
Geração de <i>logs</i>	Detalhada	Resumida	Completa
Capacidade de Auditoria e Forense	Alta	Média	Muito alta

Nota: Elaboração própria

As fontes de eventos orientadas à prevenção são nominadas de *Host Intrusion Prevention System* (HIPS) e *Network Intrusion Prevention System* (NIPS). Os princípios são semelhantes ao HIDS e ao NIDS, adicionando a capacidade de bloquear ameaças.

Assim sendo, através da fusão de HIDS, NIDS, HIPS e NIPS, surge um sistema híbrido completo nominado de *Intrusion Detection and Prevention System* (IDPS), que abrange tanto a deteção quanto a prevenção.

Segundo [25], o IPS é um complemento ao IDS, pois utiliza a capacidade de deteção juntamente com a habilidade de bloqueio de uma *firewall*. Qualquer tipo de ação suspeita é notificado e bloqueado de forma eficiente em vários pontos da arquitetura de uma rede, tornando-se assim uma das ferramentas de segurança de maior abrangência.

1.6.2.3 Instante de Detecção

O instante de detecção refere-se ao período que o sistema demora na identificação de uma intrusão ou atividade suspeita. Este instante é fundamental na efetividade, visto que, define a competência de responder ou mitigar uma ameaça. Posto isto, pode-se distinguir três tipos de instantes de detecção: tempo real, virtual e *a posteriori*.

A detecção em tempo real, como o próprio nome indica, deteta o momento exato em que um ataque ocorre e evita que tenha sucesso. Entre os demais tipos, este é o instante ideal, apesar de ser bastante complexo de atingir porque os ataques são formados por vários passos, envolvendo manobras de diversão.

A detecção em tempo real virtual é a resposta à detecção de ataques que foram bem-sucedidos na primeira tentativa, impedindo que estes ataques tenham sucesso novamente ao eliminar as vulnerabilidades que a tornaram possível. Com este tipo de detecção existe a possibilidade de isolar o sistema penetrado até que se compreenda o ocorrido e evite-se a sua repetição futura.

A detecção *a posteriori* permite identificar ataques ou atividades suspeitas sucedidas. A identificação é auxiliada através de ficheiros de registos de operações que apresentam anormalidades. Este instante revela-se vantajoso numa análise detalhada, que possibilita uma compreensão aprofundada dos ataques, sendo essencial para aumentar a segurança. No entanto, por não ser uma detecção em tempo real, a resposta e a mitigação são bem mais lentos, o que consequentemente pode levar a danos gravíssimos [7].

1.6.2.4 Reatividade

Relativamente à reatividade, os IDS podem variar entre um nível passivo e um nível ativo. Os IDS passivos apenas alertam os analistas de segurança ao entregar um relatório detalhado para que estes possam responder à ameaça com uma maior rapidez.

Os IDS ativos são conhecidos pela sua resposta automatizada, que apesar de não possuírem métodos defensivos, conseguem interagir com sistemas de defesa. A reatividade pode tomar medidas defensivas que incluem a modificação de regras da *firewall*, isolamento das componentes comprometidas, reparação ou até a substituição por um sistema alternativo [26]. Um contra-ataque não é de todo uma medida responsável, que para além de ser tão ilegal quanto um ataque, não será efetivo visto que grande maioria dos atacantes utilizam métodos de anonimato, como *Virtual Private Networks* (VPNs), *proxies*, servidores de aluguer ou máquinas comprometidas [7].

1.6.2.5 Tipo de Análise

Em termos de tipo de análise, os IDS podem efetuar de maneira isolada ou cooperativa.

A simplicidade e rapidez, são características da análise isolada, visto que, os eventos ou pacotes de dados são analisados de forma independente. No entanto, apresenta dificuldades em identificar ataques sofisticados que ocorrem em diversas etapas.

Na análise cooperativa, é realizado uma análise complexa, que correlaciona informações de vários eventos ou sistemas, que pode implicar o recurso a normas, caso seja utilizado IDS de fabricantes distintos. Assim sendo, a detecção de ataques é mais precisa, porém necessita de mais poder computacional [7].

1.7 Sistemas de Monitorização de Segurança de Rede

A monitorização de redes é uma prática fundamental para garantir a estabilidade, a segurança e o desempenho de ambientes computacionais. Trata-se de um processo contínuo que procura identificar e corrigir possíveis falhas ou lentidão em componentes da rede interna, promovendo respostas mais rápidas a incidentes e contribuindo para a disponibilidade dos serviços. Independentemente da solução de monitorização escolhida, deverá oferecer uma interface descomplicada, sem comprometer a profundidade analítica, visto que, os dados recolhidos serão direcionados para diferentes públicos, como administradores de redes e sistemas, bem como gestores.

Em resumo, a monitorização de redes assume um papel estratégico na proteção de informações sensíveis, controlando os pontos de acesso e prevenindo incidentes relacionados a fontes externas e internas [27].

1.7.1 Snort

O *Snort* é uma ferramenta *open-source* focada na detecção de intrusões em redes informáticas. Trata-se de um sistema baseado em assinaturas, capaz de operar como *sniffer*, *packet logger* ou como NIDS. No modo *sniffer*, o *Snort* apenas lê e apresenta os cabeçalhos dos pacotes em tempo real. No modo *packet logger*, este *software* armazena pacotes em diretórios para análise posterior. O modo mais relevante, no contexto da segurança, é o NIDS, onde o *Snort* analisa o tráfego em tempo real, identifica padrões suspeitos e gera alertas com base em regras predefinidas [28].

A logística por detrás deste sistema centra-se em regras, que são ficheiros de texto com uma estrutura específica. Cada regra é composta por um cabeçalho, onde é definido

o protocolo, endereços de IP e portas relevantes, e por uma seção de opções, onde é especificado os critérios detalhados de detecção.

Um dos aspetos mais relevantes da sua arquitetura é a utilização de pré-processadores, sendo estes responsáveis por processar os pacotes antes da aplicação das regras. Estes componentes permitem funções adicionais, como descodificação de protocolos, normalização de dados e deteção de anomalias no tráfego de rede [27].

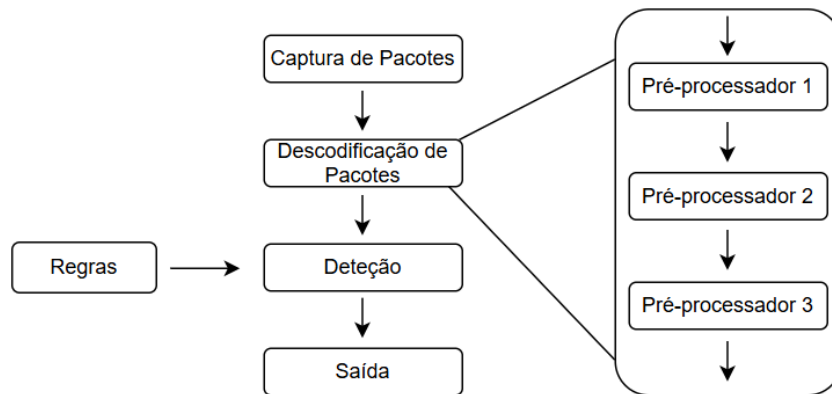


Figura 1.4: Arquitetura *Snort*

Nota: Adaptado de [27]

Em síntese, o *Snort* revela-se uma solução eficiente e extensível para a monitorização e proteção de redes, sendo especialmente útil em contextos que envolvem análise detalhada de *logs* de tráfego para a deteção de comportamentos anómalos e tentativas de intrusão.

1.7.2 Suricata

O Suricata é uma ferramenta sofisticada de monitorização de rede, ao desempenhar um papel central na deteção de intrusões e destacando-se também pela sua capacidade robusta de geração e análise de *logs*. Este *software* opera como um IDS que efetua uma *Deep Packet Inspection*, ao gerar registos detalhados sobre o tráfego, as ameaças detetadas e os comportamentos anómalos observados.

A sua compatibilidade com o formato de regras do Snort facilita a integração com ambientes de segurança já estabelecidos, permitindo o reaproveitamento de regras e políticas previamente definidas. Além disso, ao oferecer suporte integrado a diversos protocolos da camada de aplicação, como *HyperText Transfer Protocol* (HTTP), *Domain Name System* (DNS) e *File Transfer Protocol* (FTP), o Suricata possibilita a criação de *logs* mais detalhados e contextualizados.

A arquitetura do Suricata assemelha-se à do *Snort*, diferenciando-se no pré-processador que se divide em descodificação e deteção. No processo de descodificação, os pacotes são processados camada por camada, com informações relevantes sendo extraídas e armazenadas numa representação interna. Estes dados alimentam a fase de deteção, onde as regras são aplicadas e eventos de segurança são identificados. O resultado desta análise é armazenado num ficheiro *log* detalhado, podendo incluir registos de alerta, transações HTTP, fluxos de rede, entre outros.

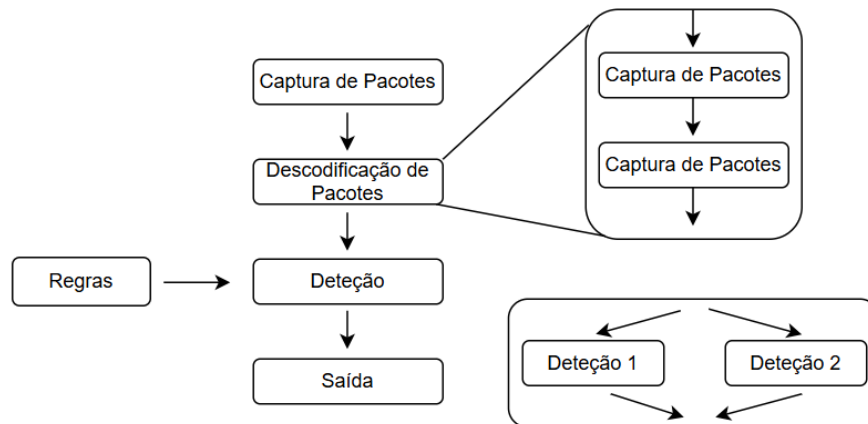


Figura 1.5: Arquitetura Suricata

Nota: Adaptado de [27]

Uma das grandes vantagens do Suricata como ferramenta de *logs* é a sua arquitetura *multithread*, que permite o processamento paralelo do tráfego sem a necessidade de distribuição externa de carga. Este sistema torna-se deveras adequado em ambientes com grandes volumes de dados.

Os *logs* gerados pelo Suricata são amplamente compatíveis com *Security Information and Event Management* (SIEM), permitindo a visualização, correlação e análise centralizada dos eventos de rede [27]. Além disso, o suporte a formatos como *JSON* facilita a integração com *pipelines* modernos de análise de dados.

1.7.3 OSSEC

O *Open Source Security Information and Event Management* (OSSEC) é *host-based* IDS, amplamente reconhecido na comunidade da cibersegurança pela sua versatilidade na monitorização e análise de *logs*, verificação de integridade de ficheiro e deteção de *rootkits* [29]. Diferencialmente de ferramentas como o *Snort* e o Suricata, que operam predominantemente ao nível da rede analisando o tráfego entre dispositivos, o OSSEC atua diretamente no interior dos sistemas operativos. Esta localização estratégica

permite-lhe obter uma visibilidade profunda sobre eventos internos que seriam invisíveis para um sensor de rede, tais como a modificação de binários do sistema, tentativas de atribuição indevida de privilégios ou o acesso indevido a ficheiros sensíveis [30].

A arquitetura do OSSEC baseia-se num modelo de cliente-servidor, estruturando-se em torno de um gestor central e múltiplos agentes distribuídos pelos anfitriões. O processo inicia-se com a recolha de dados em tempo real pelos agentes, que monitorizam registos de eventos, integridade de ficheiros e atividade do sistema operativo.

Estes dados são enviados através de um canal encriptado para o gesto, onde ocorre a fase de descodificação e análise. Nesta etapa, os descodificadores processam os *logs* brutos para extrair campos relevantes e normalizar a informação, permitindo que o motor de análise aplique regras predefinidas para identificar comportamentos anómalos ou assinatura de ataques.

Uma vez detetada uma ameaça, o sistema gera alertas detalhados e pode desencadear respostas ativas, como o bloqueio automático de endereços IP ou a execução de *scripts* de mitigação, armazenando todos os eventos num registo centralizado para auditoria posterior [29], [30].

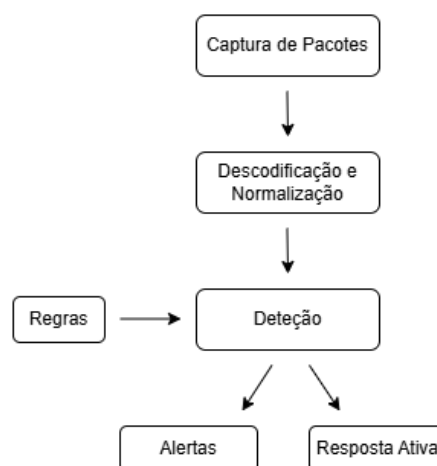


Figura 1.6: Arquitetura OSSEC

1.8 Logs

No domínio da computação e da cibersegurança, os *logs*, ou registos de eventos, constituem uma fonte de dados primordial para a compreensão do comportamento de sistemas e aplicações. Fundamentalmente, os *logs* são registos cronológicos de eventos, gerados automaticamente por uma variedade de fontes digitais, incluindo sistemas operativos, aplicações de *software*, dispositivos de rede e infraestruturas digitais em geral.

Estes registos podem ser compostos por uma única linha de texto ou por múltiplas linhas, capturando detalhes de interações e estados ao longo do tempo [31].

A prática de documentar ocorrências, de forma detalhada, precede significativamente no surgimento do primeiro computador, encontrando-se presente ao longo da história da humanidade. Na navegação marítima, os capitães e navegadores mantinham registos detalhados sobre rotas, condições meteorológicas, entre outros acontecimentos a bordo. No setor da saúde, hospitais e médicos efetuavam registos sobre pacientes, sintomas e tratamentos.

Com o início da era dos computadores, por volta das décadas de 1950 e 1960, originaram-se paralelamente os *logs* digitais, que inicialmente eram projetados para manutenção de *hardware*. Em sistemas de computação modernos, estes registos digitais tornaram-se uma fonte rica e indispensável para diversas análises.

As fontes de eventos, como podemos verificar na Figura 1 [32], normalmente armazenam as atividades de forma sequencial, ordenadas por tempo [33]. Assim sendo, estes eventos são registos de uma interação realizada durante a execução de uma operação do sistema. Os registos podem fornecer dados tais como:

- Data e Hora – O instante preciso em que o evento ocorreu;
- Origem – A entidade (sistema, aplicação, serviço) que gerou o registo;
- Nível de Severidade – Classificação da criticidade do evento (“Informação”, “Aviso”, “Erro”, “Fatal”);
- Endereço de IP – Identificadores de rede ou dispositivos envolvidos em comunicações;
- Mensagens de Texto – Descrições pormenorizadas do evento, que podem variar de notificações concisas a detalhados rastreios de erros;
- Outros parâmetros que se mostrem relevantes, podem ser adicionados ao registo.

Para garantir o processamento consistente destes registos, os eventos são manipulados através de uma expressão regular (*regex*) que contém uma estrutura padrão, utilizada na identificação e extração de informações específicas. Este formato garante que o sistema consiga processar *logs* de maneira consistente. A mensagem em destaque inclui todo o conteúdo que possa estar inserido num registo [34].

Type	Date	Time	Source	Category	Event	User	Computer
Information	1/21/2010	10:34:15 ...	ESENT	General	100	N/A	VIDYAVASU
Error	1/18/2010	10:34:20 ...	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:20 ...	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:23 ...	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:25 ...	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:28 ...	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:30 ...	crypt32	None	8	N/A	VIDYAVASU
Information	1/18/2010	10:34:56 ...	EAPOL	None	2002	N/A	VIDYAVASU
Information	1/18/2010	10:34:56 ...	EAPOL	None	2003	N/A	VIDYAVASU
Information	1/18/2010	10:35:59 ...	iPod Service	None	0	N/A	VIDYAVASU
Error	1/18/2010	10:36:24 ...	crypt32	None	8	N/A	VIDYAVASU
Warning	1/15/2010	10:36:30 ...	crypt32	None	6	N/A	VIDYAVASU
Error	1/15/2010	10:36:30 ...	crypt32	None	8	N/A	VIDYAVASU
Information	1/18/2010	10:36:39 ...	DesktopCentral	None	103	N/A	VIDYAVASU
Error	1/18/2010	10:36:39 ...	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:37:31 ...	crypt32	None	8	N/A	VIDYAVASU
Information	12/24/2009	10:38:15 ...	ESENT	General	101	N/A	VIDYAVASU

Figura 1.7: Fontes de Eventos

Nota: Retirado de [32]

Fontes de eventos que fornecem descrições estruturadas, possuem entradas distintas e uniformes que descrevem cada evento para um determinado conjunto de características. No entanto, os *logs* não estruturados, não necessitam de entradas de eventos uniformes [33].

A cibersegurança é um assunto que ganhou uma enorme relevância ao longo dos anos. Com o aumento exponencial de invasões e ataques, considera-se os *logs* como uma ferramenta fundamental na detecção e análise de possíveis ameaças. Cada vez mais, hackers exploram as falhas de segurança, comprometendo assim a autenticidade, a integridade, a confidencialidade e a disponibilidade de um sistema [35]. Assim sendo, a recolha de eventos de *logs* de entrada da rede e das demais aplicações determinam a existência de violações [36].

Devido à natureza automatizada na produção de *logs* nos sistemas de computação, tornou-se impraticável efetuar uma inspeção manualmente. Para além de ser extremamente trabalhoso, revela-se inoportuno para responder a incidentes em tempo real [37]. Deste modo, cria-se a necessidade de implementar técnicas de detecção de anomalias capazes de aprender automaticamente os modelos que representam o comportamento normal do sistema. Posteriormente, estas técnicas identificam e alertam atividades possivelmente adversas, que requerem a atenção de operadores humanos [31] [38].

1.8.1 Ferramentas de Logs

A crescente complexidade de ambiente de Tecnologias de Informação (TI) e o aumento exponencial do volume de dados gerados por sistemas, aplicações e dispositivos de rede tornaram essencial o uso de ferramentas especializadas para a gestão e análise de *logs*. Estas ferramentas desempenham um papel indispensável na detecção de anomalias, na resposta a incidentes, na auditoria e na monitorização contínua da infraestrutura.

1.8.1.1 Elastic Stack

A *Elastic Stack*, é uma solução de larga adoção para a centralização, análise e visualização de *logs* provenientes de diversas fontes. Esta plataforma *open source* destaca-se pela sua escalabilidade, flexibilidade e habilidade em lidar com conjuntos de dados volumosos em tempo real.

A sua arquitetura é constituída por quatro componentes principais:

- *Elasticsearch* – Este componente é o mecanismo central de pesquisa e indexação. É responsável por armazenar os *logs* num formato estruturado e permite consultas eficientes e escaláveis. A sua estrutura orientada a documentos facilita a análise de grandes quantidades de dados em praticamente tempo real.
- *Logstash* – Atua como uma *pipeline* de dados, responsável pela ingestão, transformação e encaminhamento de *logs* para o *Elasticsearch*. Suporta um vasto conjunto de entradas, filtros e saídas, permitindo normalizar e enriquecer os dados durante o processo de recolha.
- *Kibana* – É a interface de visualização dos dados armazenados, possibilitando a criação de painéis interativos, relatórios personalizados e alertas.
- *Beats* – São agentes leves, instalados diretamente nos sistemas a serem monitorizados, que recolhem e enviam dados para o *Logstash* ou diretamente para o *Elasticsearch*. Existem diversos tipos de *Beats*, nomeadamente, o *Filebeat* para ficheiros de *log*, *Packetbeat* para tráfego de rede e *Metricbeat* para métricas do sistema.

Estes componentes, em conjunto formam uma solução de *log management* eficiente e idealizada para ambientes distribuídos e dinâmicos. A *Elastic Stack* permite a correlação de eventos, deteção de anomalias e resposta a incidentes, sendo estas funcionalidades indispensáveis em contextos de segurança de informação. Além disso, a sua integração com ferramentas de segurança como o Suricata, OSSEC e soluções SIEM, amplia a sua aplicabilidade como parte de um ecossistema mais abrangente de monitorização e defesa cibernética [39] [40].

1.8.1.2 Splunk

O *Splunk* é uma plataforma comercial para análise de *machine data*, com ênfase na gestão, indexação e visualização de *logs*. É geralmente utilizada em ambientes empresariais devido à sua escalabilidade, capacidades avançadas de correlação de eventos e facilidade de integração com outras ferramentas de segurança [41].

Entre as principais componentes destacam-se:

- Indexador – É responsável por receber os dados, provenientes dos *forwarders* ou de outras fontes. Estes dados são organizados e estruturados e de seguida armazenados para posterior consulta.
- Cabeçalho de Pesquisa – Trata-se do componente responsável para interface com o utilizador.
- Encaminhadores – Também conhecidos como *forwarders*, este componente recolhe e encaminha as informações para o indexador ou para outro encaminhador [42].

1.8.1.3 Graylog

O *Graylog* é uma plataforma *open-source* orientada à centralização, indexação e análise de dados, sendo principalmente utilizados para fins operacionais, auditoria e segurança da informação. Projetado para lidar com grandes volumes de dados em tempo real, o *Graylog* destaca-se pela sua escalabilidade, modularidade e integração com múltiplas fontes de dados, como servidores, dispositivos de rede, aplicações e sistemas de segurança [43].

A sua arquitetura é constituída por três componentes principais, nomeadamente o *Graylog Server*, que executa o processamento central de mensagens e fornece *Application Programming Interface* (APIs) de consulta, o *Elasticsearch*, que fica responsável pela indexação e pesquisa dos dados e o *MongoDB*, que armazena metadados e configurações do sistema [44].

O objetivo central do *Graylog* é reunir *logs* provenientes de várias fontes, como servidores, aplicações, dispositivos de rede, sistemas operativos e ferramentas de segurança, para permitir sua análise em tempo real e gerar *insights* relevantes para operações, auditoria e segurança.

1.9 SIEM

O SIEM tem vindo a se destacar como uma ferramenta fundamental na prevenção, deteção e na reação a ataques cibernéticos. Estes sistemas evoluíram significativamente, consolidando-se como plataformas abrangentes que oferecem uma visibilidade acrescida. Esta visibilidade melhorada auxilia na identificação de áreas críticas com maior exposição a riscos, possibilitando a formulação de estratégias proativas de mitigação.

Consequentemente, estas abordagens contribuem substancialmente para a diminuição dos custos operativos e na redução do tempo de resposta a incidentes de segurança [45] [46].

A arquitetura típica de um SIEM é composta por vários módulos, nomeadamente, dispositivos de origem, recolha de *logs*, normalização de análise sintática, mecanismos de regras, armazenamento de *logs* e monitorização de eventos [47], [48] .

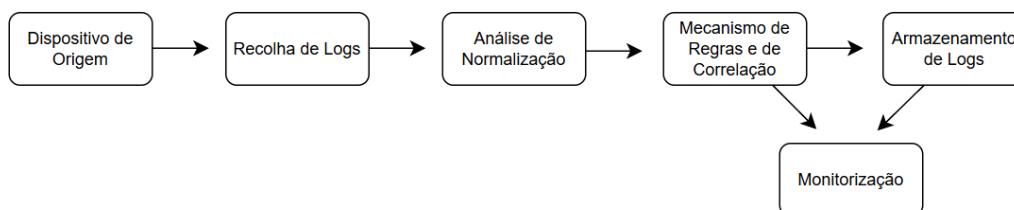


Figura 1.8: Arquitetura básica do SIEM

Nota: Adaptado de [45]

Estes componentes trabalham entre si para fornecer uma análise em tempo real dos eventos de segurança, ao permitir a deteção de padrões anómalos e a geração de alertas que promovem respostas imediatas [45], [47].

Para além destas capacidades estruturais, os SIEMs distinguem-se por vários atributos funcionais. A sua eficiência na deteção de eventos encontra-se diretamente relacionado à robustez das regras de correlação, sendo que apenas algumas soluções oferecem suporte a linguagens avançadas para buscas mais complexas [46].

As soluções modernas englobam também recursos de processamento em tempo real, tendo a capacidade de lidar com milhões de eventos simultaneamente, garantindo agilidade na resposta. No entanto, a gestão de dados volumosos continua a representar um desafio técnico e económico, sobretudo no que diz respeito ao armazenamento e à eficiência de indexação [46], [47]. Outro fator determinante é a capacidade de visualização, visto que é indispensável uma interface intuitiva ou flexível para uma melhor compreensão e análise dos eventos ocorridos [45], [46].

O desempenho geral de um SIEM é avaliado com base com vários requisitos, como a sua capacidade de processamento e armazenamento, destacando-se, entre os mais relevantes, os seguintes:

- Escalabilidade – Determina a capacidade de distribuir as tarefas de ingestão, correlação e armazenamento de eventos entre múltiplos nós ou instâncias, garantindo desempenho consistente mesmo com o aumento do volume de dados e do número de dispositivos monitorizados;

- Resiliência – Refere-se à capacidade do SIEM de manter a continuidade das suas operações mesmo perante falhas ou incidentes, assegurando que os mecanismos de correlação funcionem de forma redundante, que existam estratégias eficazes de recuperação e que o armazenamento dos eventos disponha de replicação dos dados para evitar perdas de informação.

Adicionalmente, destaca-se a importância de outros recursos complementares, como a capacidade de reação automatizada a incidentes, a geração de relatórios abrangentes, a integração com ferramentas de análise comportamental, nomeadamente, o *User and Entity Behavior Analytics* (UEBA) e a adoção de mecanismos de segurança como criptografia e a automação de processos. Estes aspetos, em conjunto, definem o grau de maturidade e a eficácia de uma solução SIEM frente aos desafios crescentes da cibersegurança [45], [46].

Apesar dos avanços significativos nos Sistemas de Gestão de Informações e Eventos de Segurança, estes ainda apresentam uma série de limitações. Um dos principais desafios enfrentados é a limitação na cobertura dos dados capturados. Apesar de integrarem grandes volumes de *logs* provenientes de *firewalls*, controladores de domínio e sistemas de antivírus, muitos não incluem fontes críticas como tráfego de DNS, *logs* de *endpoints* ou *e-mail*. A ausência destes dados impede uma visão abrangente da rede, comprometendo a eficiência dos mecanismos de correlação [46], [47]. Além disso, a fragmentação da identidade dos utilizadores e as restrições impostas por normas como o Regulamento Geral de Proteção de Dados (RGPD) dificultam a análise precisa de atividades suspeitas, aumentando o valor de falsos positivos e negativos [45], [46].

As regras de correlação adotadas na maioria dos SIEMs tendem a se restringir a encadeamentos simples de eventos, voltados para padrões específicos e previsíveis. Todavia, este modelo mostra-se inadequado perante à complexidade e à variedade dos vetores de ataque contemporâneos, que frequentemente recorrem a técnicas furtivas e estratégias de *zero-day*. A ausência de mecanismos de correlação histórica ou baseados em anomalias comportamentais, reduz significativamente a capacidade de deteção proativa [47], [48].

A limitação na capacidade de armazenar dados de eventos por longos períodos continua sendo um desafio, especialmente em casos de ataques persistentes que são apenas identificados meses após a sua ocorrência. Muitos SIEMs armazenam apenas eventos recentes e não reutilizam dados históricos para análises posteriores, o que

consequentemente, restringe significativamente a eficiência da investigação e da resposta a incidentes mais complexos [45], [47].

A identificação do tipo de ameaça, a seleção da resposta apropriada e a implementação de contramedidas continuam a exigir conhecimento especializado e intervenção manual, o que retarda a resposta e aumenta a probabilidade de erro [45], [46].

Embora alguns SIEMs ofereçam a possibilidade de executar *scripts* ou ações automáticas em resposta a determinados alertas, estas funcionalidades costumam ser limitadas e exigem configuração manual. Ademais, há uma escassez de ferramentas integradas para gerar relatórios de incidentes que atendam às exigências de diferentes normativos e estruturas regulatórias, comprometendo o cumprimento de obrigações legais e de auditoria [46], [47].

A visualização dos dados recolhidos é, em muitos casos, pouco eficaz, genérica e não adaptada às necessidades dos analistas de segurança. A ausência de representações visuais interativas e contextuais dificulta a interpretação rápida dos eventos e a identificação de padrões anómalos. Além disso, a maioria dos sistemas raramente incorporam visualizações baseadas em dados de fontes externas ou comportamentos de utilizadores, restringindo a perceção situacional da rede [45], [46].

1.10 Modelos de Inteligência Artificial

No contexto da cibersegurança, os modelos de Inteligência Artificial possibilitam a aprendizagem de padrões representativos, em relação ao comportamento normal do sistema, que subsequentemente permitem identificar desvios que possam traduzir-se em falhas, comportamentos anormais ou potenciais ameaças à integridade dos sistemas. A implementação de técnicas de IA, nomeadamente, *Machine Learning* e *Deep Learning*, contribuem para uma resposta rápida, precisa e informada a incidentes [49] [38].

1.10.1 Algoritmos de Machine Learning Supervisionado

Os algoritmos de *Machine Learning* são métodos computacionais que permitem que os sistemas aprendam padrões, a partir de dados e façam previsões ou tomadas de decisão, sem serem explicitamente programados para isso [50]. O *Machine Learning Supervisionado* é uma técnica que se divide em dois tipos de tarefas, sendo elas, classificação e regressão. A regressão associa-se à previsão de um valor contínuo, ou seja, o modelo tenta prever uma saída numérica através de dados de entrada ainda não observados. Em contraste, a tarefa de classificação envolve a previsão de valores

categoricos, atribuindo a cada entrada uma classe específica entre um conjunto de categorias possíveis [51].

1.10.1.1 Random Forest

O *Random Forest* é um método de *ensemble* que desempenha previsões a partir da agregação dos resultados de múltiplas árvores de decisão [52]. Este método é considerado particularmente adequado para situações com um número elevado de características (*features*), sendo capaz de lidar com grandes volumes de dados e manter uma previsão precisa, mesmo na presença de ruído e dados ausentes. A eficácia deste algoritmo reflete-se na sua estrutura constituída por diversos classificadores fracos, cuja combinação resulta num classificador forte. Este tipo de abordagem permite atenuar o problema de *overfitting*, algo comum em árvores de decisão individuais, aumentando a capacidade de generalização do modelo.

No entanto, num caso hipotético em que, num conjunto de características contenha apenas uma pequena porção de informação relevante, o desempenho do *Random Forest* pode ser afetado negativamente. A introdução de atributos irrelevantes tende a influenciar a construção das árvores, levando-as a ajustar-se ao ruído presente nos dados, o que pode comprometer todo o desempenho [53] [54][35]. Ainda assim, o algoritmo minimiza este risco por meio da técnica *bagging*, que será discutida adiante.

A construção de uma única árvore, dentro da floresta, inicia-se com a seleção de uma característica x e um limite t que dividem um conjunto de treino X , composto por N amostras pertencentes a duas classes e descritas por G características, em dois subconjuntos X_L e X_R . Esta divisão visa maximizar um critério de separação, normalmente medido pelo índice de *Gini* ou pela *Entropia* [53].

$$\text{Gini}(S) = 1 - \sum_{i=1}^C p_i^2$$

$$\text{Entropia}(S) = - \sum_{i=1}^C p_i \log_2(p_i)$$

Onde p_i representa a proporção de elementos da classe i no conjunto S . C é o número total de classes. Este procedimento é iterativamente repetido para cada subconjunto, utilizando outras combinações (x, t) , até que não seja possível efetuar mais nenhuma divisão.

Ao contrário da *Decision Tree*, o *Random Forest* utiliza a técnica de *bagging*, na qual cada árvore é treinada com uma amostra aleatória, com reposição de N instâncias. Estas amostras constituem o conjunto denominado *in-bag*, enquanto as instâncias não selecionadas durante o processo são encaminhadas para um conjunto *out-of-bag*, que eventualmente poderá ser utilizado na estimativa de erro do modelo sem a necessidade de uma validação cruzada.

$$\text{Err}_{\text{OOB}} = \frac{1}{N} \sum_{i=1}^N I(\hat{y}_{\text{OOB}}^{(i)} \neq y^{(i)})$$

Onde $y^{(i)}$ representa o rótulo verdadeiro da amostra i , $\hat{y}_{\text{OOB}}^{(i)}$ é a previsão para a amostra i , feita apenas pelas árvores que não utilizaram essa amostra durante o treino e $I(\cdot)$ é a função indicadora, que retorna 1 se a previsão for incorreta e 0 se estiver correta.

Adicionalmente, na tentativa de reduzir a correlação entre árvores e aumentar a diversidade *ensemble*, considera-se, em cada nó, apenas um subconjunto aleatório de G características. $G = \sqrt{G}$ é considerado para determinar a melhor divisão. Esta aleatoriedade adicional contribui para a consistência do algoritmo, ao melhorar a precisão em problemas complexos e de alta dimensionalidade.

A predição final para uma nova instância é obtida por votação majoritária entre as árvores, de acordo com a seguinte expressão [53]:

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), \dots, h_R(x)\}$$

Onde $h_1(x)$ é a previsão de i -ésima árvore para a instância x , e R é o número total de árvores da floresta.

Constata-se que o *Random Forest* se destaca por diversas vantagens, nomeadamente a sua robustez, a sua eficiência na gestão de conjunto de dados com elevada dimensionalidade, entre outras. Contudo, importa salientar que o algoritmo apresenta limitações. A aleatoriedade introduzida na seleção de atributos, embora indispensável na redução de correlação entre árvores, pode em certos casos resultar numa dependência excessiva das regras de construção, aumentando o risco de *overfitting* [55]. Além disso, o desempenho do algoritmo tende a ser afetado quando aplicado a conjuntos de dados desequilibrados. Por fim, tanto o treino quanto a fase de inferência podem tornar-se computacionalmente exigentes, especialmente quando se utiliza um número alto de

árvores, o que representa um desafio em ambientes com limitações de tempo e recursos computacionais.

1.10.1.2 XGBoost

O *Extreme Gradient Boosting* (XGBoost) é um algoritmo baseado em Árvores de Decisão, sendo amplamente utilizado em tarefas de classificação e regressão [56]. O seu desenvolvimento foi focado no desempenho computacional, escalabilidade e capacidade de generalização [57]. Trata-se de uma técnica de *ensemble learning* baseada no algoritmo de *gradient boosting*, ou seja, permite envolver diversos modelos de aprendizagem que, em conjunto, melhoram o desempenho de cada um individualmente [57] [58]. No decorrer deste processo, n modelos são treinados consecutivamente, sendo que cada novo modelo considera os erros cometidos pelo anterior. Deste modo, o processo consiste em minimizar a função de perda $L(y, F(x))$, onde y caracteriza o valor real e $F(x)$ a previsão do modelo [57].

Inicialmente, o modelo base realiza uma predição constante, e de seguida são calculados os erros de predição, indicando a diferença entre valores reais e os valores preditos. Através da otimização da função de perda é efetuado o processo de ajuste baseado no *gradient*, ao utilizar o algoritmo *gradient descent*. Uma nova árvore é construída em cada etapa, com o objetivo de prever os erros do modelo anterior, consoante a seguinte expressão:

$$F_m(x) = F_{m-1}(x) + \alpha_m \cdot h_m(x, r_{m-1})$$

Onde $F_m(x)$ representa o modelo atualizado na interação m , h_m é o modelo base treinado sobre os erros r_{m-1} , e α_m e r_m são os coeficientes de regularização e de erro que minimizam a função de perda $L(y, F(x))$. O coeficiente α_m é determinado através da seguinte equação:

$$\arg \alpha \min \sum_{i=1}^m L(y_i, F_{i-1}(x_i) + \alpha h_i(x_i, r_{i-1}))$$

Contrariamente ao *bagging*, que constrói as árvores de maneira independente, o *boosting* reduz o viés do modelo, ou seja, ao aprender com os erros cometidos leva a um menor risco de *underfitting* e consequentemente a uma maior flexibilidade.

A cada divisão binária (nó) de atributos, é calculado o *Information Gain* que tem como função medir a qualidade da divisão.

$$IG = \text{similarity}_{\text{ramo1}} + \text{similarity}_{\text{ramo2}} - \text{similarity}_{\text{raiz}}$$

Com base nos erros obtidos, o *score* de similaridade de cada ramo é dado por:

$$\frac{\sum (y - \hat{y})^2}{P(1 - P) + \lambda}$$

Em que P caracteriza a probabilidade prevista pelo modelo anterior e λ o parâmetro de regularização, utilizado para evitar o *overfitting*.

Para tarefas de classificação, o XGBoost utiliza funções de ativação como por exemplo a função *sigmoide* [57].

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

De uma forma geral, a implementação do *gradient boosting* no XGBoost segue um fluxo lógico. O algoritmo começa por calcular a previsão média em problemas de regressão ou probabilidade constante no caso de tratar de um problema de classificação. De seguida, calcula os erros e constrói uma nova árvore a partir desses mesmos erros, combina as previsões com uma taxa de aprendizagem α e, por fim, repete este processo indefinidamente até atingir um número de árvores satisfatório.

Para além das vantagens mencionadas anteriormente, este algoritmo oferece uma ampla adaptabilidade na configuração de hiperparâmetros, possibilitando ajustar o modelo conforme as características específicas dentro do contexto do problema. Parâmetros como *max_depth*, *min_child_weight*, *gamma*, permitem controlar a complexidade da árvore e induzir a aleatoriedade durante o treino, favorecendo a generalização e a eficiência do modelo. No entanto, o XGBoost apresenta determinadas limitações, como a sua elevada complexidade computacional, tanto no treino quanto na fase de inferência, principalmente quando é utilizado diversas árvores ou quando o algoritmo é aplicado num volumoso conjunto de dados. A necessidade de um ajuste criterioso dos hiperparâmetros também pode representar uma barreira, já que o desempenho do modelo pode eventualmente variar consoante as configurações necessárias [58].

1.10.1.3 SVM

O *Support Vector Machine* (SVM) é utilizado tanto para problemas de classificação como de regressão [59]. É amplamente usado em tarefas de classificação de padrões, que podem ser lineares ou não lineares [60]. Em padrões linearmente separáveis, as amostras são facilmente divididas e, portanto, o algoritmo constrói um limite de decisão, designado por hiperplano. No caso dos padrões não lineares, os dados não são facilmente separáveis, o que leva o SVM a usar funções *kernel* [59] [60]. Estas funções mapeiam o espaço original num novo espaço para que os dados que originalmente não eram separáveis, possam ser separados sem calcular explicitamente todas as novas dimensões [61].

Na seleção do hiperplano, é indispensável resolver o problema da margem máxima. Esta margem define a distância entre o limite de decisão e os vetores de suporte [59]. Quanto maior a margem, o melhor será o processo de classificação. Assim sendo, os cálculos são realizados através da seguinte expressão.

$$aX + bY = C$$

Em padrões não linearmente separáveis, como já foi referido anteriormente, é necessário mapear os dados, para um novo espaço de maiores dimensões, através de uma função *kernel*.

$$K(x, y) = \Phi(x) \cdot \Phi(y)$$

Outro parâmetro de extrema importância é o parâmetro de complexidade C, que se refere à soma das distâncias de todos os pontos que se encontram do lado errado do hiper plano [60]. Este valor não deverá ser muito elevado, uma vez que a sua tolerância ao erro será praticamente nula, podendo levar a *overfitting*. O valor C também não deverá ser muito pequeno, visto que o torna demasiado tolerante a erros, podendo assim levar a *underfitting*. Assim sendo, o valor de C não deverá ser nem muito grande nem muito pequeno, pois poderá influenciar negativamente o desempenho da classificação.

De uma forma geral, o princípio do SVM num conjunto de amostras $\{(x_i, y_i)\}_{i=1}^N$, onde cada vetor $x_i \in R^d$ corresponde a uma amostra com d características, e cada classe $y_i \in \{-1, 1\}$ indica a classe a que a amostra corresponder e encontrar um hiperplano de separação que divida corretamente estas amostras consoante as suas classes. O hiperplano define-se pela seguinte equação:

$$w^T x + b = 0$$

Onde w representa o vetor normal ao hiper plano e b caracteriza o viés. Assim, a otimização do problema fica reduzido a um problema quadrático, em que o seu objetivo é maximizar a margem.

$$\min \Phi(w) = \frac{1}{2} |w|^2 = \frac{1}{2} (w, w)$$

Em que $|w|^2$ é o quadrado da norma do vetor w , que se responsabiliza por medir a largura da margem. O fator $\frac{1}{2}$ é apenas uma convenção matemática, facilitando a derivada no processo de otimização, porém não altera a solução final.

Para garantir que os pontos sejam separados corretamente pelas classes, é indispensável impor uma restrição que assegure que todos os pontos de um lado do hiper plano pertençam à sua respectiva classe. A restrição é expressa pela seguinte equação:

$$y_i(w \cdot x_i + b) \geq 1$$

Onde y_i é a classe da amostra i . Para uma amostra da classe +1 esteja do lado correto do hiper plano, o valor da equação deverá ser igual ou superior a 1. No caso de uma amostra da classe -1, o valor de equação deverá ser igual ou inferior a -1.

O *Support Vector Machine* é capaz de generalizar um problema, sendo o teor da aprendizagem estatística. A teoria da aprendizagem estatística é fornecer um enquadramento para estudar o problema, fazendo previsões e tomadas de decisões através de um conjunto de dados. Deste modo, a aprendizagem supervisionada é formulada da seguinte maneira:

Dado um conjunto de treino $\{(x_1, y_1), \dots, (x_n, y_n)\} \subset R^n \times R$, onde x_1 e y_1 são pares de entrada e de saída, extraídos de uma distribuição de probabilidade desconhecida $P(x, y)$ e uma função de perda $V(y, f(x))$ que mede o erro, o objetivo passa por encontrar uma função f que diminua a probabilidade de erros nos novos dados.

$$\min \int V(y, f(x)) P(x, y) dx dy$$

Uma das maiores vantagens do SVM reside na sua notável capacidade de generalização. Ao procurar maximizar a margem entre as classes, o modelo tende a reduzir significativamente a probabilidade de *overfitting*, sendo particularmente eficaz em conjuntos de dados com alto número de características e poucas amostras [59]. Porém,

esta eficácia tem como contrapartida um elevado custo computacional durante o processo de treino.

Em conjuntos de dados de grandes dimensões, o processo de otimização quadrática pode tornar-se extremamente exigente em termos de recursos computacionais [62]. Além disso, o facto do *Support Vector Machine* ter sido concebido para classificação binária, todo o tipo de problema que envolva mais de duas classes necessita de estratégias adicionais, como *one-vs-all*, onde treina-se um classificador para cada classe, tratando esta classe como positiva e todas as outras como negativas, ou *one-vs-one*, onde é treinado um classificador para cada par de classes possíveis e a predição é efetuada com base no voto maioritário. Estas abordagens, apesar de serem competentes, aumentam a complexidade computacional e o tempo de execução à medida que o número de classes cresce [59].

Por fim, o modelo gerado por um SVM não é facilmente interpretável, sobretudo quando se utiliza um *kernel* não linear. Esta falta de interoperabilidade pode ser um obstáculo quando é exigido uma justificação das decisões do modelo [62].

1.10.2 Algoritmos de Machine Learning Não Supervisionado

O *Machine Learning* Não Supervisionado tem como principal objetivo identificar grupos ou padrões no conjunto sem dispor de classificações de classe previamente definidos durante a fase de treino. Esta abordagem visa descobrir relações ou estruturas ocultas nos dados, de modo que certos padrões emergentes ocorram com maior frequência do que outros. Esta técnica divide-se em duas tarefas fundamentais, sendo elas, *clustering* e redução de dimensionalidade. A tarefa de *clustering* tem como objetivo agrupar os dados baseando-se nas suas semelhanças, organizando-os num número definido de grupos. Por sua vez, a redução de dimensionalidade, busca simplificar os dados, convertendo-os de um espaço com muitas variáveis para um espaço de menores dimensões, preservando, ao máximo, as características mais importantes da distribuição original. Esta transformação é útil para compressão de informação, remover atributos redundantes e melhorar o desempenho computacional [51].

1.10.2.1 DBSCAN

O algoritmo *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN), como o próprio nome indica, utiliza uma abordagem baseada em densidade, que permite identificar e agrupar conjuntos de dados com diferentes densidades em formatos esféricos [63]. No contexto deste algoritmo, o agrupamento é o processo de

formação de *clusters* baseados nas regiões com alta e baixa densidade [64] [65]. Por outras palavras, o agrupamento permite dividir conjuntos de dados em grupos variados com a similaridade *intracluster* maximizada e minimizada [64].

A técnica de agrupamento é um dos principais métodos de *data mining*. É um processo que busca padrões relevantes e anteriormente desconhecidos, porém potencialmente úteis, a partir de conjuntos de dados espaciais de grandes dimensões [66]. Deste modo, o funcionamento geral pode ser dividido em duas etapas. A primeira etapa procura um procedimento que realize uma estimativa de densidade de um ponto e identifique objetos que se encontrem em regiões densas. A segunda etapa é o processo de identificar grupos de observações que são atingíveis através de algumas observações principais. Na impossibilidade de adicionar mais pontos dentro de um *cluster*, estes pontos restantes são designados como *outliers* ou ruído [65].

A construção dos agrupamentos leva a cabo dois parâmetros fundamentais. Os *Eps* (épsilon) caracterizam o raio em torno de um ponto do qual considera-se os pontos vizinhos, determinados a densidade da região [64]. O *MinPts* refere-se à menor contagem de pontos necessária dentro do agrupamento, formando assim um *cluster*.

A vizinhança *Eps* de um ponto p , indicado por $N_{Eps}(p)$ é definida como o conjunto de pontos $q \in D$, tal que a distância entre p e q é menor ou igual a *Eps*. Por outras palavras, trata-se da localização dos pontos a uma distância máxima de *Eps* do ponto p . Com base nesta definição, verifica-se num cluster, os pontos designados por pontos centrais (*core points*) que se situam inseridos dentro do *cluster* e os pontos de fronteira (*border points*) localizam-se nas extremidades do agrupamento.

Um ponto p é diretamente acessível por densidade através de um ponto q , relacionado aos parâmetros *Eps* e *MinPts*, apenas se ambas as condicionantes forem simultaneamente satisfeitas:

- $p \in N_{Eps}(q)$ – p apresenta-se na vizinhança de q ;
- $|N_{Eps}(q)| \geq MinPts$ – q é um ponto central apenas se tiver uma densidade de pontos satisfatória.

Além disso, um ponto p é atingível por densidade através de um ponto q se existir uma cadeia de pontos $p_1, p_2, \dots, p_n$, com $p_1 = q$ e $p_n = p$, tal que cada ponto $p_i + 1$ seja diretamente acessível por densidade a partir de p_i . Esta definição possibilita que pontos

mais distantes, ainda conectados dentro de regiões densas, sejam considerados pertencentes ao mesmo agrupamento.

Dois pontos p e q são considerados conectados por densidade se existir um ponto intermediário o tal que p e q sejam atingíveis por densidade através do ponto o . Esta característica é fundamental para definir agrupamentos coerentes, mesmo quando os pontos p e q não estão diretamente acessíveis entre si.

Apoiado nestas definições, um agrupamento C , em relação a Eps e $MinPts$, é um subconjunto não vazio da base de dados D que satisfaz as seguintes propriedades:

- Maximalidade – Se $p \in C$ e q é acessível por densidade através de p , concluímos que $q \in C$;
- Conectividade – Todos os pares de pontos $p, q \in C$ estão conectados por densidade.

Os *outliers* são descritos por um conjunto de pontos não pertencentes a qualquer agrupamento identificado. De outro modo, todos os pontos $p \in D$ para os quais não existe nenhum agrupamento C_i tal que $p \in C_i$ [24].

Após concluir todo o fluxo lógico integrado no algoritmo DBSCAN para qualquer conjunto de dados, é obtido geralmente 3 tipos de pontos [64]:

- Núcleo – na formação de diversos grupos, o núcleo é o ponto a partir do qual há pelo menos k pontos numa distância r do raio;
- Borda – Qualquer ponto que contenha um ou mais pontos centrais dentro de uma distância r do raio;
- Ruído – Pode ser qualquer ponto exceto um núcleo ou uma borda e que obrigatoriamente tenha pelo menos k número de pontos numa r distância do raio.

Deste modo, verificamos que este algoritmo tem um excelente desempenho quando se trata de ruído, tendo em conta a sua habilidade em ignorar pontos não pertencentes a qualquer região densa [64]. Assim sendo, auxilia na extração de *insights* relevantes em conjuntos de dados complexos. No entanto, o DBSCAN é inadequado quando se trata de agrupar conjuntos volumosos de dados, devido ao elevado tempo de computação. O algoritmo também não possui a capacidade de identificar *clusters* caracterizados por diferentes densidades e limites parcialmente interligados [65].

1.10.2.2 K-means

O *K-means* é um algoritmo, onde k refere-se ao número de *clusters* e *means* é a média dos atributos. É geralmente utilizado para agrupar grandes conjuntos de dados, semelhante ao que ocorre no DBSCAN, principalmente quando se trata de *data mining* e reconhecimento de padrões [67] [68]. Além disso, a sua eficácia computacional e rápida convergência torna-o claramente adequado para aplicações de larga escala. Em contraste, demonstra limitações ao tratar todas as *features* de um conjunto de dados de igual forma, sem considerar os níveis de relevância [69]. A sua dependência do valor de k , leva a que a escolha inadequada deste parâmetro possa comprometer a qualidade dos agrupamentos. Ademais, o *K-Means* revela sensibilidade na presença de *outliers*, que podem distorcer substancialmente a posição dos centroides e, conseqüentemente, afetar a formação de *clusters* [70].

O principal objetivo desta ferramenta é minimizar o erro quadrático, ao procurar k *clusters* que melhor retratam os dados. Deste modo, o algoritmo busca minimizar o índice de desempenho de acordo com a seguinte função:

$$J = \sum_{j=1}^c \sum_{X \in S_j} |X - M_j|^2$$

Onde J caracteriza a soma dos erros quadráticos, S_j retrata os pontos no *cluster* j , e M_j é o vetor médio, também conhecido como centroide. De acordo com a função anterior, o algoritmo pretende minimizar a distância entre os pontos de dados e seus respectivos centroides, resultando em *clusters* bem definidos.

Para cada classe S_j , o vetor médio M_j é calculado através da expressão:

$$M_j = \frac{1}{N_j} \sum_{X \in S_j} X$$

Onde N_j é o número de amostras no *cluster* S_j [68].

Geralmente, quando falamos no processo de agrupamento, há três etapas envolvidas. Inicialmente, são escolhidos k pontos como centroides iniciais, que basicamente são o “centro” de um grupo de pontos de dados. De seguida, atribui-se ao centroide mais próximo cada ponto restante com base na distância euclidiana. Assim sendo, é formada uma classificação inicial de dados, onde a hipótese desta classificação não ser aceitável, os centroides são novamente calculados como a média dos pontos

atribuídos a cada *cluster*. Estas etapas são repetidas até que os centroides permaneçam no mesmo local [71].

1.10.2.3 Isolation Forest

O *Isolation Forest* é um algoritmo baseado em árvores, popularmente conhecido devido à sua eficácia na detecção de anomalias. Considera-se anomalias como padrões que não se identificam a um modelo bem definido de padrões normais. Assim sendo, a palavra *Isolation* refere-se ao processo de isolamento que este método realiza na separação de padrões incomuns dentro de um conjunto de dados [72].

Ao contrário dos métodos referidos anteriormente, este algoritmo não calcula distâncias nem densidade e, portanto, o tempo de processamento e a exigência de memória são consideravelmente reduzidos [72].

No processo de construção, o modelo cria várias árvores designadas por *Isolation Trees* (*iTrees*), ou em português, Árvores Isoladas. A construção de cada *iTree* é efetuada de forma recursiva, ou seja, em cada passo o algoritmo tende a escolher aleatoriamente uma *feature* e um valor de corte. Assim, divide os dados em dois subconjuntos, como ocorre numa árvore binária. O processo de divisão é concluído apenas quando uma amostra é isolada, ou seja, quando se torna o único elemento presente num subconjunto ou quando a árvore atinge a profundidade máxima previamente estabelecida. Desta forma, o modelo compreende que não será possível efetuar mais repartições e, consequentemente não haverá mais ganhos ao proceder em mais ramificações. É importante referir ainda que amostras que sejam isoladas nos primeiros níveis de ramificação, o percentual de serem classificadas como anômalas é elevada [73].

No entanto, o *Isolation Forest* apresenta dificuldades em lidar com anomalias que apenas são detetáveis quando é considerado a combinação de múltiplas variáveis. Esta situação ocorre quando o algoritmo realiza divisões unidimensionais e, portanto, dificulta o processo de isolar pontos anômalos envolvidos por dados normais em diversas dimensões. Outra limitação importante referir, denomina-se de região fantasma. O algoritmo tende a atribuir pontuações artificialmente baixas a áreas onde não existe quaisquer dados reais ou padrões idênticos aos das regiões normais. Devido a uma limitação estrutural, esta abordagem executa apenas cortes paralelos aos eixos dos dados. Portanto, impede interações não lineares entre variáveis e, por conseguinte, há uma maior probabilidade de gerar falsos negativos [74].

Com base nos princípios descritos, o *Isolation Forest* fundamenta-se na premissa de que o isolamento de amostras anómalas é uma estratégia eficaz. De acordo com a árvore binária, cada árvore de isolamento é constituída por nós que desempenham funções distintas no processo de isolamento. Se T for um nó interno, este realiza um teste de divisão e possui dois nós filhos, nomeadamente a subárvore esquerda T_l e a subárvore direita T_r . Se T for um nó externo (folha), retrata o ponto em que o isolamento de uma amostra termina.

Para construir uma *iTree*, primeiramente é selecionado aleatoriamente um atributo A e um valor de divisão p , extraídos do conjunto de dados $D = \{d_1, d_2, \dots, d_n\}$. De seguida, cada amostra d_u é inserida à subárvore T_l no caso de $d_u(A) < p$, ou na subárvore de direita se $d_u(A) > p$. Este processo termina quando uma das condições referidas previamente for satisfeita.

A métrica principal utilizada é o comprimento de caminho $h(d)$, que caracteriza o total de arestas percorridas desde a raiz até à folha onde o ponto d é isolado. O cálculo desta métrica é dado pela seguinte equação:

$$c(n) = 2H(n-1) - ((2(n-1)/n))$$

Onde $c(n)$ é a média de $h(d)$, $H(i)$ é o número harmónico e n retrata o número de folhas. A pontuação de anomalia s de uma instância d é dada pôr:

$$s(d, n) = 2 - (E(h(d)/c(n)))$$

Onde $E(h(d)) \rightarrow$ caracteriza a média do comprimento do caminho para $h(d)$ numa coleção de árvores de isolamento.

Interpretamos $s(d, n)$ da seguinte maneira:

- Quando $E(h(d)) \rightarrow 0$, então $s \rightarrow 1$ – a amostra foi isolada rapidamente e, portanto, tem uma forte probabilidade de ser considerada uma anomalia;
- Quando $E(h(d)) \rightarrow c(n)$, então $s \rightarrow 0,5$ – a amostra segue o padrão de isolamento das restantes, indicando um comportamento normal;
- Quando $E(h(d)) \rightarrow n-1$, então $s \rightarrow 0,5$ – a amostra exigiu um percurso longo para ser isolado, revelando fortes indicações de normalidade [75].

1.10.3 Algoritmos de Deep Learning Supervisionado

O *Deep Learning* é um subcampo do *Machine Learning*, que simula o modo como o cérebro humano processa informações e adquire conhecimentos. Este paradigma baseia-se em modelos compostos por diversas camadas de representação, em que cada camada utiliza os dados processados pela camada antecedente com o objetivo de contruir abstrações cada vez mais complexas. Os modelos *Deep Learning* Supervisionados dependem de dados estruturados e categorizados para produzir resultados precisos [76] [77].

1.10.3.1 LSTM

Long Short-Term Memory (LSTM) é um tipo de *Recurrent Neural Network* (RNN) que pretende superar o problema do *vanishing gradient*, que normalmente ocorre durante o processo de aprendizagem de dependências de longo prazo em sequências temporais [78].

A arquitetura LSTM é composta por um conjunto de sub-redes com ligações recorrentes, conhecidas como *memory blocks* [79]. Cada bloco é constituído por uma ou várias *memory cells* com ligações próprias e três unidades de *gates* multiplicativas, sendo elas, *input gate*, *output gate* e *forget gate* [79] [80]. A *forget gate* inicialmente não estava presente na arquitetura deste algoritmo, tendo sido posteriormente introduzida para permitir que a rede aprendesse a “esquecer” informações irrelevantes armazenadas nas *memory cells* [78]. A *cell* tem a responsabilidade de transmitir valores de estado em intervalos de tempo variáveis e as três *gates* controlam o fluxo de informações através de operações, tais como, gravação, leitura e reinicialização [78][80].

O processo de computação inicia-se pela atualização de *block input*, combinando a entrada atual $x^{(t)}$ e a saída $y^{(t-1)}$ na última iteração, através da seguinte expressão:

$$z^{(t)} = g(W_z x^{(t)} + R_z y^{(t-1)} + b_z)$$

Onde W_z e R_z são os pesos associados a $x^{(t)}$ e $y^{(t-1)}$, respetivamente, enquanto b_z caracteriza o vetor de peso de polarização. Esta mesma estrutura aplica-se às demais *gates*, substituindo os subscritos conforme seja necessário.

De seguida, a *input gate* unifica a entrada atual $x^{(t)}$, a saída $y^{(t-1)}$ e o valor da célula $c^{(t-1)}$ no passo de tempo anterior, mediante a seguinte fórmula:

$$i^{(t)} = \sigma(W_i x^{(t)} + R_i y^{(t-1)} + p_i \odot c^{(t-1)} + b_i)$$

Onde $\odot$ representa a multiplicação pontual de dois vetores.

Nas etapas anteriores, a camada LSTM decide quais informações devem ser mantidas nos estados das células da rede $c^{(t)}$. Isto envolve a seleção dos valores candidatos $z^{(t)}$ que podem ser adicionados aos estados das células, bem como a definição dos valores de ativação $i^{(t)}$ das *inputs gates*.

A *forget gate*, estabelece quais informações devem ser removidas dos estados anteriores de célula $c^{(t-1)}$. Para isso, os valores de ativação $f^{(t)}$ da *forget gate* no passo de tempo t , são calculados com base na entrada atual $x^{(t)}$, nas saídas $y^{(t-1)}$ e no estado $c^{(t-1)}$ das *memory cells* no passo de tempo anterior ($t - 1$), considerando também as conexões de *peephole* e os termos de polarização b_f da *forget gate*. É calculado a partir da seguinte equação:

$$f^{(t)} = \sigma(W_f x^{(t)} + R_f y^{(t-1)} + p_f \odot c^{(t-1)} + b_f)$$

De seguida, calcula-se o valor da *cell*, combinado com o *block input* $z^{(t)}$, a *input gate* $i^{(t)}$ e o *forget gate* $f^{(t)}$, juntamente com o valor da célula do passo tempo anterior. Calculado através da expressão:

$$c^{(t)} = z^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)}$$

Na sequência, o cálculo da *indoor gate*, unifica a entrada atual $x^{(t)}$, a saída $y^{(t-1)}$ e o valor da célula $c^{(t-1)}$ na última instância, conforme a seguinte função:

$$o^{(t)} = \sigma(W_o x^{(t)} + R_o y^{(t-1)} + p_o \odot c^{(t)} + b_o)$$

Por fim, a saída de bloco calcula o *block output* que combina o valor atual da célula $c^{(t)}$ com o valor da *output gate* atual da seguinte forma:

$$y^{(t)} = g(c^{(t)}) \odot o^{(t)}$$

No entanto, a escassa interoperabilidade deve-se à arquitetura complexa baseada em diversas camadas e operações não lineares do LSTM, funcionado como uma “caixa negra”. Portanto, dificulta a compreensão e a explicação das decisões tomadas pelo modelo, comprometendo a transparência de resultados. Além disso, tal como acontece noutros tipos de redes neuronais, exige vários parâmetros de controlo, tais como, a quantidade de neurónios por camadas, o número de passos temporais, entre outros [80].

O *Gated Recurrent Unit* (GRU) segue uma arquitetura muito semelhante à do LSTM. Por sua vez, o GRU simplifica a estrutura ao fundir as *inputs gates* e *forget gates* numa única porta de atualização, e ao unificar o estado da célula com o estado oculto. Comparativamente com o LSTM, este algoritmo exige menos números de parâmetros, tornando-o mais leve computacionalmente e destacando-se pela sua velocidade de treino e simplicidade [81] [82].

Porém, perante o LSTM, o *Gated Recurrent Unit* apresenta uma capacidade de memória mais limitada por exigir menos parâmetros, o que poderá dificultar na captura de dependências de longo prazo em sequências complexas. O facto de o estado da célula estar fundido com o estado oculto, poderá ser desvantajoso no caso de ser necessário reter informações relevantes ao longo de sequências mais extensas [81] [83].

1.10.3.2 BERT

O *Bidirectional Encoder Representations from Transformers* (BERT), é um algoritmo *opensource*, baseado nos *Transformers*, para Processamento de Linguagem Natural (PLN) [84]. O princípio de funcionamento deste modelo consiste na fase de pré-treino e *fine tuning*, nas quais aprende representações de texto a partir de grandes volumes de dados textuais não supervisionados, podendo atingir um excelente desempenho relativamente a tarefas de PLN [85].

O BERT utiliza um *two-way Transformer*, ou seja, um Transformador bidirecional como codificador, permitindo considerar simultaneamente a informação contextual. Desta forma, a relação entre palavras é compreendida facilmente, em relação a outros modelos que apenas utilizam codificadores unidirecionais. Para a aprendizagem de representação textual no pré-treino, este algoritmo utiliza duas tarefas, sendo elas, *Masked Language Model* (MLM) e *Next Sentence Prediction* (NSP). A tarefa MLM tem como objetivo forçar o modelo na captura de informação contextual e, assim, captura representações contextuais mais informativas das palavras. O processo desta tarefa leva a que uma percentagem das palavras permaneça em estado oculto, levando o modelo a ser treinado na previsão das palavras ocultas, maximizando a função de verossimilhança, contribuindo na capacidade de entender os vários significados de diversas palavras em diferentes contextos [85] [86].

Na tarefa NSP, os dados são divididos em duas partes iguais. Numa metade os pares de frases são contextualmente consecutivos e na outra metade não. Portanto, esta tarefa leva a que o algoritmo pré-treinado tenha a capacidade de identificar os pares de

frases consecutivos [86]. Uma vez treinado, o BERT poderá ser utilizado em diversas tarefas de processamento de linguagem natural, como é o caso de análise de *logs* [85].

Na fase *fine-tuning*, ao adicionar apenas uma camada de saída ao modelo treinado previamente e combinar com dados categorizados para a aprendizagem supervisionada, com o intuito de realizar uma tarefa específica.

Dado que este método aprende a linguística comum, o número de parâmetros necessário a serem ajustados é bastante reduzido, refletindo num excelente desempenho, numa maior flexibilidade e versatilidade [85].

1.10.4 Algoritmos de Deep Learning Não Supervisionado

Os modelos de *Deep Learning* Não Supervisionado são capazes de identificar automaticamente padrões, atributos e relações relevantes através de dados não estruturados, sem a necessidade de intervenção humana direta. Além disso, estes sistemas possuem a capacidade de avaliar e melhorar continuamente as suas próprias saídas, promovendo o seu nível de precisão ao longo do tempo [77].

1.10.4.1 LogBERT

O *Log Bidirectional Encoder Representations from Transformers* (LogBERT), é um algoritmo focado na detecção de anomalias em dados de *logs*, inspirado na arquitetura BERT e adaptado ao domínio de sistemas computacionais. Trata os *logs* como sequências de eventos, ao aprender automaticamente padrões sequenciais e dependências contextuais complexas. Através da adaptação do poder de representação semântica do BERT, o LogBERT simula o comportamento normal do sistema com base em grandes volumes de *logs* não categorizados. Durante o período de treino, qualquer desvio dos padrões aprendidos pode ser identificado como anomalia.

A arquitetura de LogBERT é constituída por três estágios, designadamente, o pré-processamento, a codificação contextual com camadas *Transformer* e o pré-treino a partir de duas tarefas principais, sendo elas, *Masked Log Key Prediction* (MLKP) e *Volume of Hypersphere Minimization* (VHM).

Cada evento e_i é convertido numa chave de *log*, mais concretamente designado pelo processo de Tokenização, e associado a um vetor de *embedding* $x_i = \text{Embedding}(e_i)$, formando uma sequência $S = [e_1, e_2, \dots, e_n]$. Estes *embeddings* são processados por camadas *encoder* do *Transformer*, dando resultado em representações contextuais $H = [h_1, h_2, \dots, h_n]$.

O MLKP, inspirado pelo MLM do algoritmo BERT, esta tarefa consiste em mascarar aleatoriamente algumas posições da sequência com um *token* especial (*MASK*) e prever a chave de *log* original. A previsão para cada posição mascarada $i \in M$ é dada por:

$$\hat{e}_i = \arg \max_{e \in V} P(e|S_{\text{mask}})$$

Com a função de perda associada:

$$L_{\text{MLKP}} = - \sum_{i \in M} \log P(e_i|S_{\text{mask}})$$

O VHM atua como regularizador, ao comprimir representações normais dentro de uma hiperesfera de raio R e com centro c . A perda VHM é dada pela seguinte expressão:

$$L_{\text{VHM}} = R^2 + \frac{1}{n} \sum_{i=1}^n \max(0, |z_i - c|^2 - R^2)$$

No período de treino, o modelo é otimizado com a combinação ponderada de ambas as perdas:

$$L_{\text{total}} = \lambda_1 L_{\text{MLKP}} + \lambda_2 L_{\text{VHM}}$$

Na fase de treino, o LogBert pode identificar anomalias com base no critério MLKP, no caso de a chave real e_i não estiver entre as k mais prováveis, a sequência é considerada anômala:

$$Anomalia_{\text{MLKP}}(e_i) = \begin{cases} 1, & \text{se } e_i \notin \text{Top-}k(P(\cdot|S_{\text{mask}})) \\ 0, & \text{caso contrário} \end{cases}$$

Um outro critério a ter em conta, é a distância latente à hiperesfera. Se a distancia entre a representação z da sequência e o centro c for superior ao valor do raio, é então considerada uma anomalia:

$$Anomalia_{\text{VHM}}(S) = \begin{cases} 1, & \text{se } \|z - c\|^2 > R^2 \\ 0, & \text{caso contrário} \end{cases}$$

Assim sendo, o LogBERT apresenta vantagens como a contextualização bidirecional, ao considerar o contexto completo à esquerda e a direita de cada evento. A sua flexibilidade e generalização deve-se à sua capacidade de adaptar tipos de sistemas e formatos de *log*. No entanto, possui limitações como o custo computacional visto que o

pré-treino necessita de grandes volumes de dados. A sua eficácia depende da qualidade da tokenização das chaves de *log* e do tamanho da janela de contexto escolhida. Além disso, é menos eficaz na detecção de anomalias cujo padrão sequencial se mantém, mas o conteúdo semântico é anômalo [87].

1.10.4.2 VAE

O *Variational Autoencoder* (VAE) é um subconjunto dos *Autoencoders*, enquadrando-se no domínio dos modelos generativos baseados em variáveis latentes. Enquanto os *autoencoders* convencionais têm como principal tarefa comprimir os dados de entrada numa representação de dimensões reduzidas e seguidamente reconstruí-los, os VAEs introduzem uma abordagem probabilística que permite não apenas a reconstrução dos dados, como também gerar novas amostras realistas [88] [89].

Diferentemente de um *autoencoder* tradicional, que codifica diretamente os dados de entrada de um vetor latente fixo, como por exemplo, uma imagem de proporções 28x28, o VAE aprende uma representação probabilística contínua do espaço latente. Assim sendo, o codificador deste algoritmo transforma os dados de entradas em dois vetores distintos, nomeadamente, μ que retrata a média e σ que caracteriza o desvio padrão de uma distribuição latente. Estes dois parâmetros definem uma distribuição de probabilidade através do espaço latente de forma contínua e estocástica [90].

Ao considerar x um vetor de variáveis observadas, proveniente de um sistema subjacente fixo, é dado um conjunto de dados $D = \{x^{(1)}, x^{(2)}, \dots, x^{(3)}\}$, que se supõe que as amostras são independentes e identicamente distribuídas, segundo uma distribuição $p^*(x)$. O objetivo é aproximar esta distribuição com um modelo paramétrico $p_\theta(x)$:

$$p_\theta(x) \approx p^*(x)$$

A aprendizagem baseia-se na maximização da verossimilhança dos dados:

$$\log p_\theta(D) = \sum_{x \in D} \log p_\theta(x)$$

Com a introdução de variáveis latente z , o modelo passa a considerar a relação conjunta $p_\theta(x, z)$, sendo a marginal $p_\theta(x)$ obtido por integração:

$$p_\theta(x) = \int p_\theta(x, z) dz = \int p_\theta(z) p_\theta(x|z) dz$$

Neste contexto, $p_\theta(z)$ representa a distribuição *a priori* sobre o espaço latente, enquanto $p_\theta(x|z)$ caracteriza a distribuição condicional usada na reconstrução dos dados através de uma amostra z .

Como a integral marginal $p_\theta(x)$ é, em geral, intratável, recorre-se à inferência variacional para aproximar a distribuição posterior $p_\theta(z|x)$ com um modelo auxiliar $q_\theta(z|x)$, parametrizado por ϕ :

$$q_\theta(z|x) \approx p_\theta(z|x)$$

O objetivo do treino fundamenta-se na maximização do limite inferior da evidencia, designadamente, *Evidence Lower Bound* (ELBO), que fornece uma aproximação inferior ao logaritmo da verossimilhança marginal:

$$\log p_\theta(x) = \mathcal{L}_{\theta, \phi}(x) + D_{KL}(q_\theta(z|x) \parallel p_\theta(z|x))$$

Uma vez que a divergência de *Kullback-Leibler* (KL) é sempre positiva, $\mathcal{L}_{\theta, \phi}(x)$ funciona efetivamente com um limite inferior. O ELBO é expresso por:

$$\mathcal{L}_{\theta, \phi}(x) = E_{q_\theta(z|x)}[\log p_\theta(z|x)] - D_{KL}(q_\theta(z|x) \parallel p_\theta(z))$$

Deste modo, o ELBO representa o equilíbrio entre a qualidade de reconstrução dos dados e a regularização do espaço latente.

No entanto, a retropropagação de *gradients* surge como uma limitação, visto que as variáveis aleatórias tendem a serem instáveis devido à elevada variância. Na tentativa de contornar este problema, o VAE utiliza o chamado truque de reparametrização, no qual o vetor latente z é reescrito como uma transformação determinística de uma variável auxiliar $\epsilon \sim N(0, I)$:

$$z = \mu + \sigma \odot \epsilon$$

Onde μ e σ são as saídas do codificador e correspondem, respetivamente, à média e ao desvio padrão da distribuição aproximada $q_\theta(z|x)$. Este processo auxilia o método a tornar-se derivável, ao permitir a propagação durante o período de treino. Ao assumir que $q_\theta(z|x) \parallel N(\mu, \sigma^2 I)$ e que $p_\theta(z) = N(0, I)$, a divergência KL entre essas duas distribuições gaussiana multivariada poder computada de forma analítica:

$$D_{KL}(q_\theta(x|z) \parallel p_\theta(z)) = \frac{1}{2} \sum_{j=1}^k (\sigma_j^2 + \mu_j^2 - 1 - \log \sigma_j^2)$$

Este resultado é fundamental para a formulação da função de perda final do algoritmo, que combina a reconstrução dos dados com a regularização do espaço latente. Esta estrutura possibilita que o VAE possua a capacidade de aprender representações latentes consistentes, estruturada e apropriada para gerar dados realistas [91].

1.10.4.3 DeepLog

O *DeepLog* surge como uma abordagem assente em *deep learning*, desenvolvido para analisar e monitorizar eventos de *logs*, tirando partido das redes neurais recorrentes. Construído de acordo com a arquitetura LSTM, este modelo foi concebido para operar em tempo real e, portanto, possibilita o processamento contínuo de evento à medida que são produzidos. Por ser inspirado em técnicas PLN, o *DeepLog* interpreta os registos como sequências estruturadas, semelhantes a frases linguísticas, permitindo a aprendizagem de padrões típicos de execução com base no histórico de eventos. Deste modo, é capaz de identificar desvios que possam indicar uma potencial anomalia, mesmo em ambientes amplamente dinâmicos e com exigências de resposta imediata.

A arquitetura do *DeepLog* é composta por três componentes principais, nomeadamente, o modelo de deteção de anomalias baseado nas chaves de *log*, o modelo de deteção de anomalias em valores contínuos e o modelo de fluxo de trabalho, também conhecido como *workflow*.

O modelo de deteção de anomalias nas chaves de *logs*, é utilizada uma rede LSTM para estimar a probabilidade condicional do próximo evento m_{t+1} através numa janela de eventos antecedentes $w = \{m_t - h, \dots, m_t\}$. Assim sendo, a previsão é dada pela seguinte expressão:

$$Pr[m_{t+1} = k_i | w] = \text{softmax}(f_{LSTM}(w))$$

Eventos cujas chaves não se encontrem entre os d valores mais prováveis são considerados anómalos.

O segundo componente trata da deteção de anomalias em valores contínuos, como tempos de resposta ou métricas de desempenho. Estes são tratados como séries temporais multivariadas, ao permitir que o modelo antecipe os valores futuros dos parâmetros. A discrepância entre os valores previstos e os valores observados é calculada através do *Mean Squared Error* (MSE), definido pôr:

$$MSE = \frac{1}{n} \sum_{i=1}^n (v_i^{\text{real}} - v_i^{\text{previsto}})^2$$

No decorrer do treino, o modelo ajusta uma distribuição *Gaussiana* sobre os erros de previsão, o que possibilita definir limiares estatísticos para identificar desvios anómalos.

Por fim, através do modelo de fluxo de trabalho, o algoritmo reconstrói os percursos normais de execução de diferentes tarefas do sistema. Este mecanismo permite identificar ramificações, concorrência e ciclos nos processos, oferecendo uma visão mais aprofundada para diagnosticar comportamento atípicos.

Entre os diversos benefícios do *DeepLog*, destaca-se a sua elevada taxa de precisão, com valores *F-measure* superiores a 96% mesmo com conjuntos de dados de treino reduzidos. Outra característica relevante é a sua capacidade de atualização incremental, ou seja, o modelo adapta-se recorrendo ao *feedback* do utilizador, sem a necessidade de um reprocessamento completo. Além disso, o algoritmo não requer dados categorizados, exigindo apenas um conjunto representativo do comportamento normal. A inferência dos *workflows* adiciona uma camada adicional de interoperabilidade, que facilita na identificação da origem e contexto das anomalias.

Apesar das suas vantagens, o *DeepLog* apresenta algumas limitações. A sua sensibilidade perante a ordem dos eventos pode comprometer o desempenho em sistemas com múltiplas tarefas em execução concorrente, sendo indispensável a utilização de identificadores para segmentar corretamente os fluxos. Adicionalmente, a fiabilidade do modelo depende da precisão na extração de chaves e parâmetros dos *logs*, o que requer ferramentas de *parsing* fiáveis. Por último, a natureza computacionalmente intensiva das redes LSTM pode representar um entrave em ambientes com restrições de recursos [92].

1.11 Análise dos Modelos de Inteligência Artificial

Com o objetivo de definir uma abordagem metodológica eficaz para a análise de *logs* no contexto institucional, foi realizada esta pesquisa abrangente dos diversos modelos de IA existentes. Esta investigação visa identificar, dentro de cada categoria, os modelos mais adequados à deteção de intrusões e comportamentos anómalos, tendo em consideração as especificidades dos dados normalmente presentes em ficheiros de *log*.

Para uma análise comparativa mais estruturada e intuitiva dos modelos de Inteligência Artificial examinados, as suas características e desempenho foram classificados qualitativamente. As categorias principais utilizadas são ‘Muito Alta’, ‘Alta’, ‘Moderada’ e ‘Baixa’, embora alguns critérios possam usar termos relacionados para refletir nuances específicas. Estas classificações não representam valores

quantitativos absolutos, mas sim uma síntese interpretativa do consenso predominante e das tendências observadas na literatura científica revisada para cada critério avaliado.

A interpretação das categorias referidas anteriormente é a seguinte:

- ‘Muito Alta’ – Indica um desempenho ou característica superior, destacando-se significativamente na maioria dos cenários e estudos avaliados. Representa um ponto forte excepcional do modelo para o critério específico;
- ‘Alta’ – Sugere um desempenho ou característica fiável, ao demonstrar uma eficácia elevada na maioria das aplicações. É uma indicação de um ponto forte consistente;
- ‘Moderada’, ‘Média’ – Reflete um desempenho ou característica intermédia ou equilibrada, que pode ser eficaz em certas condições ou apresentar um balanço entre vantagens e desvantagens;
- ‘Baixa’ – Sinaliza um desempenho ou característica limitada, com eficácia reduzida ou dependência de condições muito específicas para ser viável. Representa uma fraqueza significativa do modelo para o critério em questão.

Adicionalmente, alguns critérios utilizam classificações específicas:

- Parametrização (‘Simples’, ‘Complexa’) – Refere-se à dificuldade e ao número de hiperparâmetros que necessitam de ajuste rigoroso para otimizar o desempenho do modelo. ‘Simples’ indica poucos parâmetros ou ajuste intuitivo e ‘Complexa’ indica um número elevado de parâmetros e/ou dificuldade significativa no seu ajuste;
- Suporte a Múltiplas Classes (‘Sim’, ‘Limitado’) – ‘Sim’ significa que o modelo lida nativamente com classificações multiclasse. ‘Limitado’ implica que o modelo foi concebido primariamente para problema binários e requer estratégia adicionais para lidar com múltiplas classes, o que pode aumentar a complexidade ou custo computacional;
- Tipos de Dados (‘Temporal’, ‘Textual’) – Descreve a natureza dos dados para os quais o modelo é mais eficazmente concebido ou aplicado, baseando-se na sua arquitetura fundamental e nos domínios de aplicação típicos.

Para finalizar, critérios como ‘Custo Computacional’ ou ‘Desempenho Computacional’ são avaliados de forma inversa em relação à preferência, ou seja, um ‘Custo Computacional Alto’ ou ‘Desempenho Computacional Baixo’ seria considerado

menos favorável, ao passo que ‘Baixo’ para custo ou ‘Alto’ para desempenho são desejáveis [38].

1.11.1 Análise Comparativa dos modelos Machine Learning Supervisionado

Tabela 1.2: Comparação dos modelos *Machine Learning* Supervisionado

	Random Forest	XGBoost	SVM
Sensibilidade a Padrões Anômalos	ALTA	ALTA	ALTA
Resistência a Ruído	ALTA	ALTA	MODERADA
Custo Computacional	MODERADO	ALTO	ALTO
Velocidade de Treino	MODERADA	BAIXA	BAIXA
Parametrização	SIMPLES	COMPLEXA	MODERADA
Suporte a Múltiplas Classes	SIM	SIM	LIMITADO
Interoperabilidade	MÉDIA	BAIXA	BAIXA
Adaptação a Dados Desequilibrados	MODERADO	ALTA	BAIXA

Nota: Elaboração própria

De acordo com a análise apresentada na Tabela 1.2, verifica-se que o modelo *Random Forest* destaca-se como uma das abordagens mais promissoras para a detecção de ataques e intrusões. A sua resistência a possíveis ruídos, aliada à sua eficiência na generalização e à facilidade de aplicação em ambientes com elevada dimensionalidade, tornam-no particularmente indicado no tratamento de *logs* com estruturas e padrões diversos. O *XGBoost*, apesar de mostrar um desempenho competitivo e um forte poder preditivo, exige um processo de parametrização mais complexo e uma maior capacidade computacional, o que eventualmente poderá limitar a sua aplicação em sistemas em que os recursos sejam mais restritos. Por sua vez, o SVM, apesar da sua eficácia teórica em contextos binários e de elevada dimensionalidade, revela-se menos adequado para conjunto de dados volumosos e com mais de duas categorias, levando assim a necessidade de adicionar estratégia na classificação de várias classes e um maior custo em termos de processamento. Deste modo, os resultados analíticos sustentam a escolha do *Random Forest* como o modelo de referência a ser aplicado dentro do contexto do trabalho [38].

1.11.2 Análise Comparativa dos modelos Machine Learning Não Supervisionado

Tabela 1.3: Comparação dos modelos *Machine Learning* Não Supervisionado

	DBSCAN	K-Means	Isolation Forest
Deteção de Intrusões	BOA	FRACA	MUITO ALTA
Sensibilidade a Comportamentos Ocultos	MODERADA	BAIXA	ELEVADA
Robustez a Ruído	ALTA	BAIXA	ALTA
Necessidade de Conhecimento Prévio	MÉDIA	ALTA	BAIXA
Interpretação de Eventos Suspeitos	MÉDIA	ALTA	MÉDIA
Eficácia em Ambientes de Tráfego Variável	ALTA	FRACA	ALTA
Capacidade de lidar com Grandes Volumes de Dados	MÉDIA	ALTA	MUITO ALTA

Nota: Elaboração própria

Conforme a comparação realizada, constata-se que o algoritmo *Isolation Forest* destaca-se como o mais adequado para a classificação de *logs*. A sua elevada sensibilidade a padrões anómalos, em conjunto com a sua resistência face a eventos atípicos e à baixa necessidade de parametrização, tornam-no particularmente ideal em ambientes em que a alta densidade de dados como é recorrente na diversidade de registos. O DBSCAN, apesar de evidenciar uma excelente capacidade na identificação de comportamentos suspeitos, apresenta uma maior sensibilidade na densidade dos dados e uma menor escalabilidade, podendo comprometer a sua aplicação em contextos operacionais de grande escala. Em contrapartida, o K-means, demonstra limitações substanciais no que diz respeito à deteção de eventos anómalos isolados, revelando-se mais adequado em tarefas de segmentação de padrões conhecidos do que para identificação de ataques desconhecidos ou pouco recorrentes [38].

1.11.3 Análise Comparativa dos modelos Deep Learning Supervisionado

Tabela 1.4: Comparação dos modelos *Deep Learning* Supervisionado

	LSTM	GRU	BERT
Tipos de Dados Mais Indicados	TEMPORAL	TEMPORAL	TEXTUAL
Retenção de Contexto	ALTA	MODERADA	MUITO ALTA
Desempenho Computacional	BAIXO	ALTO	MODERADO
Velocidade de Treino	LENTA	RÁPIDA	MÉDIA
Adequação à Detecção de Intrusões	ALTA	MODERADA	MUITO ALTA
Necessidade de Dados Categorizados	ALTA	ALTA	ALTA
Interpretação do Modelo	BAIXA	BAIXA	MÉDIA

Nota: Elaboração própria

Segundo a análise comparativa efetuada, o modelo BERT destaca-se como o mais eficiente no teor do estudo, em virtude da sua capacidade de compreender o contexto textual dos registos e de identificar padrões linguísticos relacionados a possíveis comportamentos suspeitos. A sua arquitetura bidirecional torna este algoritmo especialmente apropriado para a análise de *logs* com informação detalhada e contextualizada. O LSTM apesar de se mostrar eficiente na deteção de padrões temporais mais extensos, como tentativas de acessos distribuídas ao longo do tempo, apresenta uma maior complexidade e tempo de treino, o que pode limitar a sua aplicação na prática. Por outro lado, o GRU, é um método mais leve e eficaz, destacando-se em cenários que exigem uma deteção rápida, embora seja sacrificada alguma profundidade na compreensão sequencial dos dados. Assim sendo, a escolha do modelo ideal depende dos requisitos específicos da tarefa em concreto.

Quando se trata de registos ricos em linguagem natural, como as mensagens de erro ou descrições de eventos, o Bert torna-se o mais apropriado. O LSTM é vantajoso em cenário onde os ataques manifestam-se de modo sequencial no decorrer do tempo e, por fim, o GRU surge como uma alternativa eficiente na deteção rápida de padrões suspeitos, particularmente útil em ambientes menos exigentes em termos de processamento [38].

1.11.4 Análise Comparativa dos modelos Deep Learning Não Supervisionado

Tabela 1.5: Comparação de modelos *Deep Learning* Não Supervisionado

	LogBert	VAE	DeepLog
Sensibilidade a Anomalias	MUITO ALTA	ALTA	ALTA
Interoperabilidade de Padrões Complexos	ELEVADA	MODERADA	ELEVADA
Eficiência à Variação nos Logs	ALTA	MODERADA	BAIXA
Custo Computacional	ELEVADO	MODERADO	ELEVADO
Interpretação dos Resultados	MODERADA	BAIXA	ALTA
Adaptação a Dados Ruidosos	ALTA	ALTA	MODERADO
Aplicabilidade Geral à Detecção de Intrusões	MUITO ALTA	MODERADA	ALTA

Nota: Elaboração própria

Com base na comparação realizada, verifica-se que o modelo *LogBert* apresenta o melhor desempenho na detecção de intrusões, devido à sua elevada capacidade de compreender relações semânticas complexas e ao facto de não exigir dados categorizados. O *DeepLog* mostra-se eficaz na detecção de alterações em sequências de eventos, sendo útil em sistemas com fluxos de execução bem definidos, embora dependa fortemente da qualidade da segmentação de *logs*. Por sua vez, o VAE, apesar de ser flexível na reconstrução de padrões normais, revela uma menor precisão na identificação de intrusões complexas, devido à baixa interoperabilidade e à sensibilidade ao treino. Entre os modelos analisados, o *LogBert* surge como a solução mais eficiente e versátil para este tipo de tarefa [38].

1.12 Métricas de Avaliação de Desempenho

As métricas de avaliação de desempenho são essenciais para avaliar a eficácia de modelos de *Machine Learning*. Estas métricas possibilitam a comparação de diferentes abordagens, no ajuste de hiperparâmetros e na tomada de decisões. Em contrapartida, no contexto de *Deep Learning*, a dependência destas métricas é menor, visto que muitos modelos têm a capacidade de aprender representações complexas diretamente a partir dos dados.

Todas as métricas de avaliação são fundamentadas nos diferentes atributos utilizados na Matriz de Confusão, sendo esta uma matriz bidimensional que concede informações sobre a classe Real e Prevista [93].

- Verdadeiro Positivo (VP) – Classifica corretamente uma amostra anômala como ataque;
- Falso Negativo (FN) – Classifica incorretamente uma amostra de ataque como instância comum;
- Falso Positivo (FP) – Classifica incorretamente uma amostra comum como instância anômala;
- Verdadeiro Negativo (VN) – Classifica corretamente uma amostra sem ataque como instância comum.

Na Matriz de Confusão, os elementos que se apresentam na diagonal evidenciam as previsões corretas, enquanto todos aqueles que não se apresentam na diagonal referem-se às previsões erradas [93].

	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

Figura 1.9: Matriz de Confusão

Nota: Retirado de [94]

A *Accuracy* mede a proporção de previsões corretamente classificadas em relação ao total de previsões realizadas [95].

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

A Precisão refere-se à proporção de intrusões corretas que um IDS foi capaz de prever em relação ao total de alertas gerados. Um *P* elevado é indicador de um alarme mais baixo, pois significa que a proporção de falsos positivos é reduzida, o que

consequentemente resulta em alertas mais confiáveis [95]. A Precisão é uma medida de desempenho útil apenas quando um conjunto de dados se encontra equilibrado [93].

$$Precision = \frac{VP}{VP + FP}$$

O *Recall* é responsável por medir o quanto de uma intrusão é expectável contra a quantidade de intrusões que realmente ocorreram [95]. Por outras palavras, o *Recall* tem a capacidade de avaliar a cobertura do modelo em detetar corretamente uma intrusão.

Quanto maior o *recall*, melhor será o modelo na captura de ataques. No entanto, um valor elevado de *recall* pode vir acompanhado de um aumento de falsos positivos, o que pode diminuir a precisão. Portanto, é fundamental equilibrar o *recall* e a precisão para obter um modelo eficaz.

$$Recall = \frac{VP}{VP + FN}$$

O *F1-score* é fundamentalmente a média harmónica da Precisão e do *Recall* [93]. De outro modo, o *F1-score* permite encontrar um equilíbrio entre as duas métricas, tanto que se um IDS for bastante preciso (gera poucos falsos positivos) e completo (deteta a maior parte dos ataques), indica que o valor de *F1-score* foi elevado.

$$F1\ score = 2 * \left(\frac{Precision * Recall}{Precision + Recall} \right)$$

A Taxa de Falsos Negativos (TFN) é representada pela proporção de falsos negativos em relação ao total de amostras que deveriam ter sido classificadas como positivas [96], ou seja, esta métrica diz-nos a quantidade de ameaças que passaram despercebidas.

$$TFN = \frac{FN}{FN + VP}$$

A Taxa de Falsos Positivos (TFP) é a proporção de falsos positivos em relação à totalidade de amostras que deveriam ter sido classificadas como negativas [96], ou seja, esta métrica diz-nos a quantidade de vezes que um sistema IDS gerou falsos alertas.

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

1.13 Trabalhos Relacionados

O presente capítulo tem como foco contextualizar o trabalho de investigação no panorama atual da Detecção de Intrusões, ao analisar os principais trabalhos que exploram o uso de algoritmos de *Machine Learning* e *Deep Learning* aplicados à análise de *logs* e tráfego de rede. A revisão incide particularmente sobre duas vertentes, mais especificamente, em modelos de *ensemble learning* e as arquiteturas sequenciais baseadas em redes recorrentes, enfatizando a forma como a literatura tem abordado desafios como o desequilíbrio de classes e a seleção de métricas adequadas de avaliação.

1.13.1 Fundamentação e Desafios

A evolução e sofisticação dos ataques informáticos tem impulsionado a migração dos sistemas de deteção tradicionais, baseados em assinaturas, para abordagens sustentadas por *Machine Learning* e *Deep Learning*, capazes de identificar padrões anómalos e ataques *zero-day* com maior eficácia do que os métodos clássicos de *misuse detection* [97], [98], [99].

Apesar do seu potencial, a literatura destaca dois desafios recorrentes:

- O desequilíbrio extremo de classes, uma vez que as anomalias representam uma fração mínima dos registos de segurança, dificultando a aprendizagem do modelo e levando a uma elevada taxa de falsos negativos;
- A heterogeneidade e natureza sequencial de *logs*, que exigem estratégias avançadas de pré-processamento e modelação para representar adequadamente a informação temporal [100].

1.13.2 Modelos de Ensemble

Os algoritmos de *ensemble* têm sido amplamente estudados pela sua robustez e elevado desempenho em tarefas de deteção. Modelos como o XGBoost e o *Random Forest* destacam-se por apresentarem elevadas taxas de precisão e boa capacidade de generalização [101], [102], [103]. Estudos comparativos recentes apontam o XGBoost como o modelo de maior eficácia global, alcançando *Accuracy* superiores a 98% e demonstrando uma excelente adaptação à identificação de ataques complexos [103], [104].

Não obstante, a literatura reconhece que estes modelos, embora muito eficazes em termos de *accuracy*, nem sempre privilegiam métricas como o *Recall*, essenciais em

contextos de segurança onde a falha na detecção de um ataque pode ter consequências graves [102].

1.13.3 Modelos de Deep Learning Sequencial

As redes neurais profundas, têm emergido como abordagens centrais na análise de *logs* e dados de rede pela sua capacidade de modelar dependências temporais de longo prazo [105], [106], [107].

O mecanismo de *gates* destas redes permite reter e descartar informação de forma seletiva, o que se traduz numa aprendizagem mais eficaz de padrões temporais. Trabalhos experimentais reportam desempenhos expressivos de modelos LSTM e GRU na previsão de anomalias em séries temporais e sequencias de eventos [106], [107].

Estudos mais recentes que combinam redes convolucionais com unidades recorrentes (CNN+GRU, CNN+LSTM) demonstram que a combinação CNN+GRU oferece o melhor equilíbrio entre precisão e eficiência computacional, sendo especialmente adequada para aplicações em tempo real [108].

1.13.4 Síntese e Contributos do Estudo

A análise dos trabalhos relacionados permite concluir que os modelos *ensemble* se destacam pela elevada precisão global, enquanto as redes recorrentes como o GRU demonstram uma aptidão superior para captar relações temporais em dados sequenciais.

Contudo, a literatura apresenta lacunas importantes:

- Foco insuficiente no *Recall* – A maioria dos estudos privilegia métricas globais como a *Accuracy*, negligenciando a minimização de falsos negativos, fator crítico em contextos de cibersegurança.
- Falta de validação em cenários simulados – São raros os estudos que testam a capacidade de generalização dos modelos perante intrusões artificialmente reproduzidas.

Deste modo, a presente investigação procura colmatar estas lacunas, oferecendo uma análise comparativa abrangente entre modelos de *Machine Learning* e *Deep Learning* sob uma perspetiva prática, centrada na redução dos falsos negativos e na avaliação da robustez dos modelos em condições próximas de ambiente reais de cibersegurança.

2 METODOLOGIA E RESULTADOS

2.1 Introdução

O objetivo deste capítulo consiste em avaliar, de forma sistemática, o desempenho dos vários modelos referidos anteriormente. Para além de permitir uma comparação prática entre modelos de diferentes naturezas, esta análise procura também confrontar os resultados obtidos empiricamente com os fundamentos teóricos descritos na literatura, de modo a verificar em que medida ambos se confirmam ou divergem.

Os dados utilizados para o estudo foram disponibilizados pelo Instituto Politécnico de Viseu, constituindo registos reais de segurança com elevada dimensionalidade e complexidade. Esta característica reforça a pertinência da avaliação, uma vez que aproxima os resultados experimentais de um cenário prático de aplicação.

Na análise de desempenho, diversas métricas são consideradas. Contudo, para o presente caso de estudo, a métrica de maior relevância é aquela que se relaciona com a minimização dos falsos negativos. Este fator é determinante, dado que a não deteção de uma anomalia pode provocar consequências mais graves do que a ocorrência de falsos positivos. Assim sendo, privilegia-se o *Recall* como referência principal para a avaliação comparativa dos modelos.

Desta forma, este capítulo visa não apenas apresentar uma análise quantitativa rigorosa, mas também fornecer uma reflexão prática sobre a aplicabilidade dos modelos em cenários reais de cibersegurança, em que a capacidade de identificar anomalias com fiabilidade é um requisito essencial.

Neste sentido, inclui igualmente uma componente de simulação experimental, concebida para avaliar o comportamento do modelo que obteve o melhor desempenho na fase de análise comparativa, perante cenários de intrusão artificialmente reproduzidos.

2.2 Ambiente Computacional

O ambiente computacional para a implementação prática e avaliação dos vários modelos de *Machine Learning* e *Deep Learning* neste trabalho consiste num portátil Asus ROG Strix G512LV. Este sistema encontra-se equipado com um processador Intel® Core™ i7-10870H CPU @ 2,20GHZ (8 núcleos, 16 processadores lógicos), que oferece

a capacidade de processamento necessária para algoritmos que beneficiam da paralelização por CPU, como os modelos baseados em árvores. A memória RAM de 16 GB DDR4 constitui o suporte principal para o carregamento e manipulação de grandes volumes de dados. Para tarefas que exigem aceleração de *hardware*, particularmente na fase de exploração de modelos de *Deep Learning*, o sistema integra uma placa gráfica NVIDIA GeForce RTX 2060 (6 gb GDDR6, 115 W – Turing). O armazenamento do *dataset* e dos resultados é assegurado por um SSD NVMe PCIe M.2 de 1TB, contribuindo para a eficiência nas operações de I/O.

2.3 Pré-processamento dos Dados

Inicialmente, os dados encontravam-se distribuídos por múltiplos ficheiros *Comma Separated Values* (CSV), caracterizando-se por uma elevada dimensionalidade, com a potencial união das colunas de todos os ficheiros a resultar em milhões de atributos. Esta complexidade era agravada pela presença significativa de *Not a Numbers* (NaNs) e por tipos de dados heterogêneos, que variavam entre numéricos, categóricos e de data/hora. Além disso, foram identificadas inconsistências subtis, como a presença de caracteres invisíveis (`\uffeff`) em nomes de colunas e a alta cardinalidade de certas *features* categóricas, como ‘Service’, ‘Source’ e ‘Destination’, que representavam milhares de valores únicos.

A metodologia de pré-processamento foi sistematicamente construída através de diversas fases, procedendo primeiramente ao carregamento e concatenação dos dados. Todos os ficheiros CSV foram lidos individualmente, e uma análise preliminar foi conduzida para identificar as colunas comuns a todos os ficheiros e aquelas que eram incomuns. Posteriormente, os *DataFrames* foram combinados num único *Dataframe* principal. Durante esta etapa, foi implementada uma seleção criteriosa de colunas, optando por manter todas as colunas comuns. Relativamente às colunas incomuns foram selecionadas aquelas que se mostravam potencialmente relevantes para a deteção de anomalias, mesmo que isso implicasse a introdução de NaNs. Crucialmente, as colunas ‘Action’ e ‘Severity’ foram explicitamente protegidas de qualquer remoção nesta fase, dada a sua importância futura na definição de variável *Target*.

Seguiu-se a fase de limpeza de nomes de colunas, onde caracteres especiais, como o `\uffeff`, foram removidos para garantir a consistência e evitar problemas de referência.

O tratamento de valores em falta constitui um passo fundamental. Após a concatenação, as colunas com uma percentagem de NaNs igual ou superior a 70% foram removidas do *dataset*. Este limiar foi estrategicamente definido para reduzir a dimensionalidade de forma pragmática, baseando-se na premissa de que colunas com tão poucos dados válidos dificilmente contribuiriam para a aprendizagem do modelo. Quaisquer NaNs remanescentes, para valores categóricos foram preenchidos com a moda e valores numéricos com a mediana.

A engenharia de *features* a partir de dados de data/tempo e duração foi fundamental para converter informação temporal em formato numérico. Colunas como ‘Time’, ‘Policy Date’, ‘start_time’, ‘lastupdatetime’ e ‘segment_time’ foram transformadas, gerando novas *features* como ano, mês, dia da semana, hora, minuto, segundo e o *timestamp* Epoch. Paralelamente, a coluna ‘elapsed’ foi convertida para ‘elapsed_seconds’, representando o tempo em segundos. Os NaNs resultantes destas transformações foram devidamente preenchidos.

A redução de dimensionalidade e a codificação de variáveis categóricas representaram o maior desafio e a principal otimização do *pipeline*. Primeiramente, colunas identificadoras com cardinalidade extremamente alta, nomeadamente, ‘Id’, ‘Marker’, ‘High Level Log Key’, foram removidas, devido à sua natureza única tornava-as inviáveis como *features* e causavam “*MemoryError*”. Em seguida, para colunas categóricas com alta cardinalidade, mas de relevância preditiva como, ‘Source’ e ‘Destination’, foi aplicado o *Hashing Encoding*. Esta técnica converte valores categóricos em representações numéricas, atribuindo cada categoria a um número fixo de colunas e, portanto, permitiu transformar estas *features* num número fixo e administrável de colunas numéricas controlando eficazmente a dimensionalidade [109]. Para outras colunas categóricas (‘Service’, ‘App Protocol’, ‘Interface’, etc.), foi implementada uma função de agrupamento de categorias raras (‘group_rare_categories’). Esta função consolida categorias que representam menos de 0,5% da distribuição total numa única categoria ‘Other_Rare_Category’ reduzindo significativamente o número de colunas binárias resultantes. Após este agrupamento e o *Hashing Encoding*, o *One-Hot Encoding* foi aplicado às restantes *features* categóricas, convertendo-as em formato binário. Por fim, foi realizada uma remoção de *features* de baixa variância, ao descartar colunas que eram quase constantes (mais de 99,9% dos valores idênticos), pois estas não contribuem para a aprendizagem do modelo.

O processo resultou na criação da variável *Target*. Após uma análise aprofundada sobre a definição mais adequada de “anomalia” no contexto da deteção de ameaças, a coluna ‘Severity’ foi selecionada como a base principal para esta variável, em detrimento da coluna ‘Action’. A decisão fundamentou-se na capacidade da coluna ‘Severity’, em refletir de forma mais direta e intrínseca o risco ou impacto inerente aos eventos, independentemente da resposta reativa do sistema de segurança. Enquanto a coluna ‘Action’ expressa o que o sistema decidiu fazer (‘Accept’, ‘Drop’ ou ‘Detect’), a ‘Severity’ classifica a gravidade percebida do evento (‘Low’, ‘Medium’, ‘High’, ‘Critical’). Ao escolher ‘Severity’ permite que o modelo aprenda a identificar eventos que são verdadeiramente de alto risco, mesmo que o sistema de segurança os tenha, por alguma razão, permitido (‘Action’ = ‘Accept’). Esta abordagem é crucial para detetar anomalias que passaram despercebidas ou Falsos Negativos do próprio sistema, oferecendo uma perspetiva mais abrangente sobre o comportamento anómalo e o risco.

Com base nesta escolha, uma nova coluna binária, ‘Is_Anomaly_Severity’, foi criada, onde eventos classificados com severidade ‘High’ ou ‘Critical’ foram categorizados como 1 para anomalia e os restantes como 0 para normal. Para o *dataset* final, as colunas ‘Action’ e ‘Severity’ originais foram removidas, uma vez que a sua informação já estaria consolidada e representada na variável *Target*, e a sua inclusão como *features* poderia introduzir redundância ou “*data leakage*” indireto. O *DataFrame* completo, com todas as *features* numéricas e a *Target* final, foi armazenado num ficheiro CSV, estando agora apto para a fase de modelagem.

Todo o processo descrito anteriormente não foi isento de dificuldades, exigindo uma abordagem iterativa e de resolução de problemas. Encontros como “*MemoryError*” e “*KeyError*” foram frequentes, refletindo a complexidade de trabalhar com volumes tão grandes e heterogéneos de dados. Embora estes desafios técnicos tenham sido superados, eles sublinham a importância de uma depuração rigorosa.

A principal limitação do *dataset* para a fase de modelagem é a desproporcionalidade extrema de classes na variável ‘Is_Anomaly_Severity’, onde a classe minoritária (‘Anomalia’) representa apenas 0,02% dos registos. Este aspeto exigirá técnicas específicas de balanceamento de classes e métricas de avaliação adequadas durante o treino dos modelos.

2.4 Avaliação dos Modelos

No desenvolvimento do presente trabalho, e dada a complexidade inerente à manipulação de um *dataset* de grande volume, assim como a diversidade de modelos de *Machine Learning* e *Deep Learning* a serem explorados, adotou-se uma estratégia de otimização de modelos faseada e seletiva. Reconhece-se que a otimização de hiperparâmetros para cada modelo é um processo computacionalmente exigente e demorado. Implementar esta otimização exaustiva para todos os modelos base em análise seria um esforço desproporcional em termos de recursos computacionais e tempo. Assim, a metodologia empregada irá consistir, primeiramente, treinar e avaliar cada modelo nas suas configurações base. Esta etapa inicial permitirá uma comparação direta do desempenho intrínseco de cada algoritmo. Posteriormente, apenas o modelo que se demonstrou o mais promissor e relevante dentro do contexto específico será selecionado para uma fase de otimização de hiperparâmetros mais aprofundada.

2.4.1 Machine Learning Supervisionado

A partir desta secção são apresentados os procedimentos experimentais realizados com os modelos de *Machine Learning* Supervisionados. Cada modelo é treinado, avaliado e comparado com base no mesmo conjunto de dados e métricas, de modo a garantir uma análise consistente do seu desempenho.

2.4.1.1 Random Forest

A implementação do modelo *Random Forest* teve início com uma análise preliminar à viabilidade computacional do processo de treino, ao considerar a elevada dimensão e complexidade do *dataset*.

Assim, numa fase exploratória, foi utilizada uma amostra estratificada correspondente a 10% do *dataset* total, de modo a reduzir o custo computacional e garantir representatividade das classes. Esta abordagem provou ser promissora, tendo o modelo alcançado um PR-AUC de 1,00 e *Recall* de 1,00 para a classe de anomalias.

Posteriormente, com o objetivo de reforçar a credibilidade e fiabilidade dos resultados, o treino foi repetido utilizando a totalidade do *dataset*. Após o carregamento dos dados, foi efetuada a separação entre variáveis preditoras (X) e da *Target* (Y), seguindo-se a divisão em conjuntos de treino e teste na proporção de 80% (1.800.240 linhas) e 20% (450.060 linhas), respetivamente, a partir do parâmetro ‘stratify=y’. Esta

configuração assegurou a preservação da distribuição das classes em ambos os subconjuntos.

As variáveis numéricas foram normalizadas ao utilizar o recurso *'StandardScaler'*, garantindo uma escala uniforme e evitando que discrepâncias de magnitude influenciassem o processo de aprendizagem.

O modelo foi configurado com os seguintes parâmetros:

- *'n_estimators=100'*;
- *'class_weight=balanced'*;
- *'random_state=42'*;
- *'n_jobs=-1'*.

A inclusão do parâmetro *'class_weight=balanced'* foi crucial para mitigar a desproporcionalidade extrema de classes, ao atribuir um peso maior às instâncias da classe minoritária, e, assim, eliminando a necessidade de técnicas de sob amostragem como o SMOTE.

Paralelamente, a definição de *'n_jobs= -1'* possibilitou a utilização de todos os núcleos de processamento disponíveis, acelerando substancialmente o treino. Apesar da dimensão do *dataset*, o processo revelou-se eficiente, resultado da capacidade de paralelização do algoritmo Random Forest e dos recursos computacionais utilizados.

Após o treino, o modelo obteve os seguintes resultados:

--- Relatório de Classificação ---				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	450012
1	0.98	0.83	0.90	48
accuracy			1.00	450060
macro avg	0.99	0.92	0.95	450060
weighted avg	1.00	1.00	1.00	450060

Figura 2.1: *Random Forest* – Relatório de Classificação

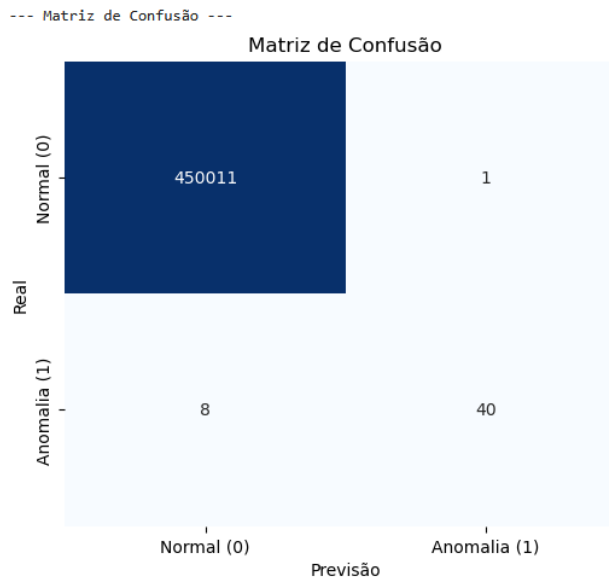


Figura 2.2: *Random Forest* – Matriz de Confusão

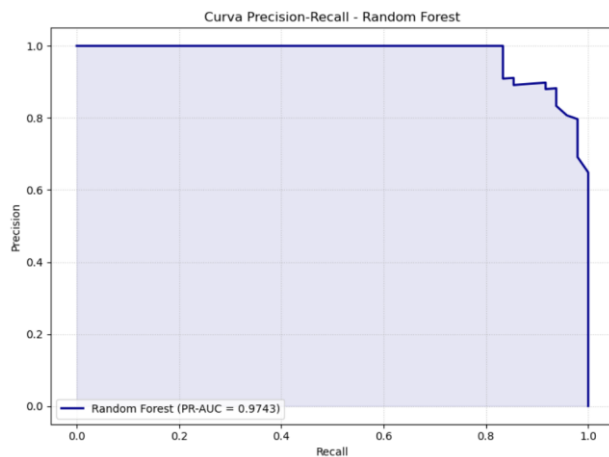


Figura 2.3: *Random Forest* – Curva *Precision_Recall*

O desempenho do modelo é, no geral, excelente para uma primeira abordagem. A alta *Precision* (0,98) para a classe ‘Anomalia’ indica que o modelo é muito eficaz em evitar falsos alarmes, o que é fundamental em sistemas de segurança para não gerar alertas excessivos. O *F1-score* de 0,90 e o PR-AUC de 0,97 demonstram um bom equilíbrio entre *Precision* e *Recall* e uma forte capacidade de distinção entre classes. No entanto, o ponto mais crítico e a principal área para melhoria é o *Recall* de 0,83 para anomalias. Este valor significa que, das 48 anomalias no conjunto de teste, o modelo falhou em detetar 8 delas, gerando falsos negativos.

2.4.1.2 XGBoost

A preparação dos dados para o treino do modelo XGBoost seguiu os mesmos procedimentos metodológicos aplicados ao modelo Random Forest, com os conjuntos de treino ('X_train', 'y_train') e teste ('X_test', 'y_test') já previamente escalados e estratificados.

O modelo foi inicializado com 'objective=binary:logistic' para classificação binária. O número de estimadores foi definido para 100 ('n_estimadores=100') e a taxa de aprendizagem foi ajustada para 0,1 ('learning_rate=0.1'), equilibrando a velocidade de convergência e o controlo de *overfitting*. A abordagem para lidar com a desproporcionalidade de classes foi crucial e específica do XGBoost, utilizando-se o parâmetro 'scale_pos_weight'. Este parâmetro é calculado como a razão entre a contagem da classe maioritária 'Normal' e a contagem da classe minoritária 'Anomalia' no conjunto de treino, conferindo um peso superior aos exemplos da classe minoritária durante o treino. Adicionalmente, o parâmetro 'n_jobs=-1' foi empregue para maximizar a paralelização do treino e 'eval_metric='logloss' para monitorização.

Após o treino, as métricas de desempenho obtidas para o algoritmo XGBoost foram as seguintes:

--- Relatório de Classificação XGBoost ---				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	450012
1	0.81	0.96	0.88	48
accuracy			1.00	450060
macro avg	0.90	0.98	0.94	450060
weighted avg	1.00	1.00	1.00	450060

Figura 2.4: XGBoost – Relatório de Classificação

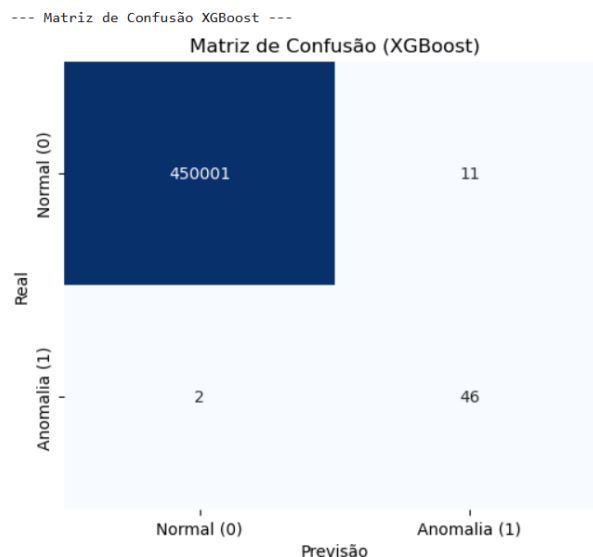


Figura 2.5: XGBoost – Matriz de Confusão

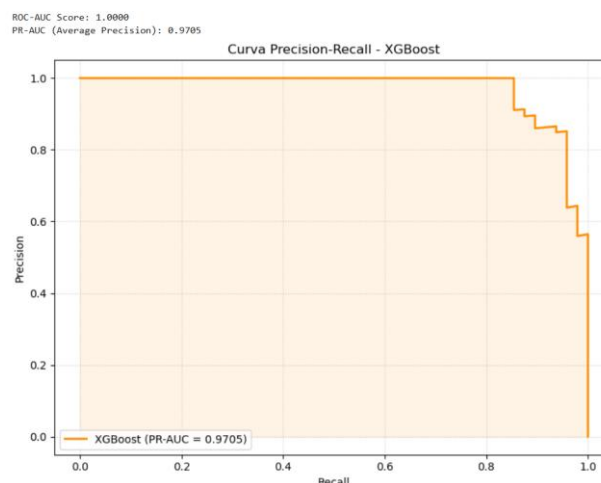


Figura 2.6: XGBoost - Curva *Precision-Recall*

Deste modo, o modelo do XGBoost demonstrou um desempenho altamente promissor e robusto, destacando-se pela sua notável eficácia na detecção de anomalias, mesmo perante a desproporcionalidade extrema de classes. A sua principal força reside no *Recall* excecional de 0,96% para a classe de anomalias. Isto significa que, das 48 anomalias reais no conjunto de teste, o modelo conseguiu identificar 46 delas, resultando em apenas 2 falsos negativos.

Esta elevada taxa de deteção veio acompanhada de uma *Precision* sólida de 0,81. Embora esta *Precision* implique 11 Falsos Positivos no conjunto de teste, o *F1-score* de 0,88 e o PR-AUC de 0,9705 confirmam um forte equilíbrio global entre estas métricas. O ROC-AUC de 1,00 também indica uma excelente capacidade de distinção do modelo.

A eficácia da estratégia ‘scale-pos-weight’ foi evidente, permitindo ao modelo aprender os padrões da classe minoritária de forma eficiente, sem necessidade de técnicas adicionais de *oversampling* ou síntese de dados.

2.4.1.3 Stochastic Gradient Descent

A preparação dos dados segue a mesma metodologia já por padrão estabelecida. A escolha da arquitetura do modelo para a fase de comparação de *baselines* foi um aspeto crítico, particularmente no que diz respeito à abordagem *Support Vector Machine*. Embora os modelos SVM, especialmente os que utilizam *kernels* não lineares, sejam reconhecidos pela sua robustez e eficácia na identificação de fronteiras de decisão complexas, a sua aplicabilidade direta ao presente *dataset* foi avaliada como inviável devido a severas limitações computacionais.

Os modelos SVM padrão, tipicamente implementados através de algoritmos como o *Sequential Minimal Optimization* (SMO), exibem uma complexidade computacional que escala dramaticamente com o número de amostras ($O(n^2)$) a $O(n^3)$, onde n representa o número de amostras no conjunto de treino. Dado que o conjunto de treino neste trabalho compreende aproximadamente 1,8 milhões de linhas ($n \approx 1,8 \times 10^6$) e cerca de 297 *features*, a aplicação de um *Support Vector Classifier* (SVC) tradicional resultaria em tempos de treinos impraticáveis, que se estenderiam por dias ou semanas, e muito provavelmente, em erros de memória devido à necessidade de utilização intensiva de memória RAM para armazenar a matriz de *kernel* e realizar os cálculos subjacentes. Mesmo a utilização de LinearSVC, que é uma implementação de SVM para o caso linear e escala de forma mais favorável, poderia ainda ser excessivamente demorada ou computacionalmente exigente para esta magnitude de dados.

De modo a contornar estas limitações, optou-se pela utilização do *Stochastic Gradient Descent* (SGD), configurado para reproduzir o comportamento de um SVM linear. Diferentemente dos métodos tradicionais, o SGD atualiza os parâmetros do modelo com base em amostras individuais, em vez de processar todo o conjunto de dados a cada iteração.

Esta estratégia reduz significativamente a complexidade computacional, que passa a ser linear ($O(n)$) em relação ao número de instâncias, garantindo eficiência de memória e escalabilidade.

O treino decorreu sem incidentes técnicos, confirmando a viabilidade e eficiência do SGD na aprendizagem de padrões através de um volume de dados desta dimensão. Assim sendo, o modelo após o treino adquiriu os seguintes resultados:

```

--- Relatório de Classificação SGDClassifier ---

```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	450012
1	0.02	1.00	0.04	48
accuracy			0.99	450060
macro avg	0.51	1.00	0.52	450060
weighted avg	1.00	0.99	1.00	450060

Figura 2.7: SGD - Relatório de Classificação

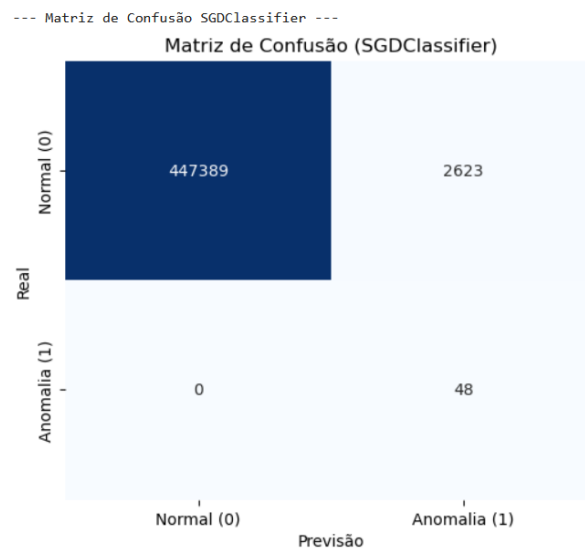


Figura 2.8: SGD – Matriz de Confusão

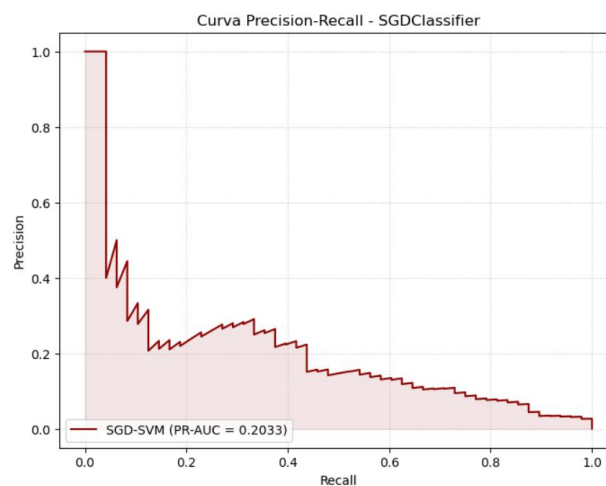


Figura 2.9: SGD - Curva *Precision-Recall*

O modelo SGD demonstrou uma característica notável, alcançando um *Recall* perfeito de 1,00 para a classe ‘Anomalia’. Isto significa que o modelo conseguiu detetar todas as 48 anomalias no conjunto de teste, resultando em 0 falsos negativos. No entanto, este *Recall* perfeito foi obtido a um custo extremamente elevado na *Precision*, que foi de apenas 0,02 para a classe ‘Anomalia’. A matriz de confusão revela que a causa foi o modelo ter gerado 2623 falsos positivos. Esta relação desfavorável entre *Recall* e *Precision* refletiu-se num *F1-score* de 0,04, indicando uma perda significativa de equilíbrio entre as duas métricas.

O PR-AUC de 0,2033, reforça esta conclusão, demonstrando a dificuldade do modelo em manter consistência nas predições sob diferentes limiares de decisão. Curiosamente, o ROC-AUC de 0,9993, indica uma boa separação teórica entre classes, mas esta métrica, por si só, mostrou-se enganosa neste contexto, uma vez que não reflete o impacto dos FP no uso prático.

2.4.1.4 Comparação Analítica

Tabela 2.1: Comparação dos resultados dos modelos *Machine Learning* Supervisionado

	Random Forest	XGBoost	SGD
<i>Precision</i>	0,98	0,81	0,02
<i>Recall</i>	0,83	0,96	1,00
<i>F1-score</i>	0,90	0,88	0,04
Falsos Positivos	1	11	2623
Falsos Negativos	8	2	0
ROC-AUC	1,00	1,00	0,9993
PR-AUC	0,9743	0,9705	0,2033

Ao analisar os resultados, torna-se claro que o SGD, apesar de ter alcançado um *Recall* perfeito, gerou uma *Precision* extremamente baixa, resultando num número elevado de Falsos Positivos. Deste modo, o elevado número de alarmes falsos tornaria o modelo impraticável dentro do teor do problema, onde o ruído excessivo inutilizaria o valor dos alertas. O *F1-score* e o PR-AUC deste modelo confirma este desempenho desequilibrado e, na prática, inadequado.

Entre os modelos *Random Forest* e o XGBoost, a escolha recai sobre o modelo que melhor equilibra as prioridades. Portanto, ao analisar a tabela verifica-se que o

Random Forest apresenta uma elevada *Precision*, retratando a escassez de falsos alarmes. No entanto o seu *Recall* resultou em 8 Falsos Negativos. Quanto ao XGBoost, o modelo demonstra um *Recall* significativamente superior, ao detetar 46 das 48 anomalias reais. Esta melhoria no *Recall* veio à custa de uma *Precision* um pouco menor, que resultou em 11 Falsos Positivos.

Conclui-se então que, embora o XGBoost tenha um valor superior de Falsos Positivos que o *Random Forest*, a sua capacidade de detetar mais 6 anomalias torna-o substancialmente mais valioso para este domínio.

2.4.1.5 Comparação Teórico-prática

A análise teórica previamente realizada, indicava o modelo *Random Forest* como a abordagem mais promissora para a deteção de intrusões e ataques em redes informáticas. No entanto, após a implementação prática, os resultados empíricos demonstraram que o XGBoost é o modelo mais indicado para o contexto. Embora ambos os modelos partilhem fundamentos semelhantes, o algoritmo XGBoost distingue-se pela sua arquitetura de *gradient boosting*, que lhe confere maior sensibilidade na deteção de padrões raros, característica crítica num cenário de extrema desproporcionalidade de classes.

Nos testes realizados, o XGBoost apresentou valores de *Recall* significativamente superiores, assegurando a deteção da maioria das anomalias presente no conjunto de teste. Apesar de gerar um número ligeiramente superior de falsos positivos em comparação com o *Random Forest*, este *trade-off* é aceitável e frequentemente preferível em cenários de segurança, onde a taxa de deteção é primordial.

Para além do desempenho preditivo, o modelo XGBoost demonstrou uma eficiência computacional notável, sendo capaz de processar grandes volumes de dados de forma ágil e estável, o que reforça a sua viabilidade prática em ambientes de monitorização contínua e análise em tempo real.

2.4.1.6 Otimização do XGBoost

A fase de otimização do modelo foi desenvolvida com o objetivo de aperfeiçoar o equilíbrio entre *Precision* e *Recall*, garantindo uma melhoria global na capacidade de deteção de anomalias. Para tal, foi utilizada uma busca aleatória de hiperparâmetros a partir do método ‘RandomizedSearchCV’, que explorou 50 combinações de parâmetros distintas.

O processo recorreu à técnica de validação cruzada estratificada ‘StratifiedKFold (n_splits=3)’, assegurado uma representação equilibrada de classes em cada partição de treino e teste. A métrica de otimização utilizada foi o F1-*score* para a classe minoritária, por refletir de forma mais equilibrada a relação entre deteção (Recall) e fiabilidade de previsões (*Precision*).

Tabela 2.2: Desempenho comparativo entre o modelo XGBoost base e otimizado

	XGBoost Base	XGBoost Otimizado
<i>Precision</i>	0,81	0,85
<i>Recall</i>	0,96	0,96
F1- <i>score</i>	0,88	0,90
Falsos Positivos	11	8
Falsos Negativos	2	2
ROC-AUC	1,00	0,9998
PR-AUC	0,9705	0,9703

A análise comparativa entre o modelo base e o modelo otimizado evidencia que o *Recall* para a classe ‘Anomalia’ manteve-se excecionalmente alto em ambos os casos. Este resultado é particularmente relevante, uma vez que confirmar que o modelo continua a detetar 46 das 48 anomalias do conjunto de teste, preservando a taxa falsos negativos em apenas 2 casos. Assim, a otimização não comprometeu a capacidade do modelo de identificar a maioria das anomalias.

Em contrapartida, observou-se uma melhoria significativa na *Precision*, que aumentou de 0,81 para 0,85 após a otimização. Este ganho traduz-se diretamente numa redução do número de falsos positivos, passando de 11 para apenas 8. Em ambientes operacionais de cibersegurança, esta melhoria tem um impacto prático substancial, pois diminui o número de alarmes falsos contribui para a eficiência e fiabilidade do sistema, reduzindo a fadiga do operador e aumentando a confiança nas previsões do modelo.

O F1-*score* também apresentou uma evolução positiva, passando de 0,88 para 0,90, refletindo uma melhoria global na coerência e estabilidade do desempenho.

Os valores de ROC-AUC e PR-AUC mantiveram-se consistentemente altos, demonstrando que a otimização, embora não tenha alterado de forma expressiva estas

métricas, acabou por melhorar o desempenho do modelo sem comprometer a sua capacidade fundamental na deteção de anomalias.

2.4.2 Machine Learning Não Supervisionado

Através desta secção são apresentados os procedimentos experimentais realizados com os modelos de *Machine Learning* Não Supervisionado. Estes algoritmos foram aplicado sob as mesmas condições metodológicas definidas para os modelos anteriores, permitindo avaliar a sua capacidade de identificar padrões anómalos sem recursos a dados rotulados

2.4.2.1 Isolation Forest

A preparação dos dados para o modelo *Isolation Forest* seguiu o mesmo *pipeline* metodológico aplicado aos modelos supervisionados, com os conjuntos de treino e teste já escalados e estratificados. Contudo, importa salientar que, ao contrário dos algoritmos supervisionados, os modelos não supervisionados não recorrem à variável *Target* durante o processo de treino. O seu objetivo consiste em identificar padrões de comportamento divergentes a partir das próprias características das observações, sem orientação explícita sobre o que constitui uma anomalia.

O modelo foi inicializado com ‘n_estimators=100’ e ‘random_state=42’ para garantir reprodutibilidade. Um parâmetro central neste tipo de abordagem é o ‘contamination’, que define a proporção esperada de anomalias no *dataset*. Numa primeira configuração, optou-se por atribuir a este parâmetro a proporção real de anomalias observadas no *dataset* completo, com um valor aproximado de 0,000106%. No entanto, esta definição revelou-se inviável, resultando numa falha completa na deteção, dado que o modelo, ao assumir uma taxa de anomalias tão baixa, considerou praticamente todos os registos como normais.

Com o intuito de diagnosticar e compreender melhor o comportamento do algoritmo, o parâmetro ‘contamination’ foi subsequentemente ajustado para 0,01 (1%), forçando o modelo a classificar uma percentagem mais ampla de observações como anómalas. Este reajuste não foi concebido como uma tentativa de calibrar o modelo para produção, mas antes como uma abordagem exploratória destinada a avaliar a sua sensibilidade e a sua resposta a diferentes limiares de deteção.

Após a conclusão do treino, procedeu-se à avaliação do algoritmo sobre o conjunto de teste, obtendo os seguintes resultados:

Tabela 2.3: Resultados comparativos do desempenho do modelo *Isolation Forest* com diferentes níveis de ‘contamination’

Métricas	<i>Isolation Forest</i> (0,000106)	<i>Isolation Forest</i> (0,01)
<i>Precision</i>	0,00	0,00
<i>Recall</i>	0,00	0,00
F1-score	0,00	0,00
Falsos Positivos	50	4358
Falsos Negativos	48	48
ROC-AUC	0,7996	0,7996
PR-AUC	0,0004	0,0004

Nas configurações testadas, o modelo demonstrou uma falha crítica na identificação de anomalias. O resultado mais preocupante foi a completa ausência de desempenho perante a classe ‘Anomalia’, evidenciada por uma *Precision*, *Recall* e F1-score de 0,00. Este resultado significa que o modelo foi incapaz de detetar uma única anomalia, ao registar 0 verdadeiros positivos, e consequentemente, classificou todas as 48 anomalias existentes no conjunto de teste como ‘Normal’. Adicionalmente, o modelo gerou 50 Falsos Positivos, ou seja, o modelo classificou incorretamente 50 eventos normais como anomalias, contribuindo para a adição de ruído no sistema. A ineficácia do modelo em identificar a classe minoritária é ainda mais confirmada pelo PR-AUC de 0,0004, um valor extremamente baixo que sublinha a sua incapacidade prática.

Por fim, verificou-se uma discrepância significativa na aplicação do parâmetro ‘contamination’. Apesar de ter sido definido como 0,01, o que na teoria implicaria uma identificação aproximada de 1% das 450,060 amostras como anomalias, o modelo classificou apenas 50 instâncias como tal. Esta diferença acentuada sugere uma possível limitação do algoritmo em adaptar-se às características específicas do conjunto de dados utilizado, indicando uma incompatibilidade entre o parâmetro configurado e o comportamento efetivo do modelo na tarefa de deteção de *outliers*.

Em resumo, a incapacidade do *Isolation Forest* em detetar qualquer anomalia, aliada à geração de falsos positivos torna-o inviável para o objetivo da investigação.

2.4.2.2 K-means

A aplicação do algoritmo *K-means* na detecção de anomalias baseou-se na premissa de que pontos que se encontram “distantes” dos centros dos *clusters* (centroides) ou pertencentes a *clusters* muito pequenos podem ser considerados anomalias.

Tendo em consideração que o número de *clusters* (k) é um dos principais hiperparâmetros do modelo, optou-se por configurar inicialmente com $k = 8$, com o intuito de agrupar os comportamentos normais em poucos padrões dominantes, assumindo que as anomalias surgiriam como desvios isolados a esses grupos. No entanto, esta configuração inicial revelou-se insuficiente, uma vez que o modelo não conseguiu distinguir de forma eficaz entre padrões normais e anómalos.

Com base nesta limitação, o número de *clusters* foi subsequentemente aumentado para $k = 50$, de forma a permitir uma representação mais granular dos comportamentos normais. Partiu-se da hipótese de que uma segmentação mais detalhada do espaço de características possibilitaria uma detecção mais sensível de *outliers*, seja pela distância em relação aos centroides predominantes, seja pela associação a *clusters* residuais de pequena dimensão.

Para a detecção de anomalias no conjunto de teste, cada ponto foi atribuído ao *cluster* com o centroide mais próximo. De seguida, foi calculada a distância euclidiana entre cada ponto e o seu centroide atribuído. Definiu-se um limiar de anomalia com base nos percentis superiores das distâncias. Inicialmente o percentil de 99,99% e, numa segunda tentativa, o percentil de 99,9%. Os pontos com distâncias superiores a estes limiares foram classificados como anómalos, enquanto os restantes foram considerados normais.

As métricas de desempenho resultantes das duas configurações são apresentadas na tabela posterior.

Tabela 2.4: Resultados comparativos de desempenho do modelo K-means com números de *clusters* distintos

Métricas	K-means (8)	K-means (50)
<i>Precision</i>	0,00	0,00
<i>Recall</i>	0,00	0,00
<i>F1-score</i>	0,00	0,00
Falsos Positivos	901	451
Falsos Negativos	48	48
ROC-AUC	0,5631	0,6152
PR-AUC	0,0001	0,0001

O modelo, em ambas as configurações testadas, demonstrou ser incapaz na detecção de anomalias. Tanto com 8 quanto com 50 *clusters*, o modelo registou uma *Precision*, *Recall* e *F1-score* de 0,00 para a classe minoritária. Isto significa que o K-Means não conseguiu detetar nenhuma anomalia real (0 VP), classificando todas as 48 anomalias presentes no conjunto de teste como ‘Normal’.

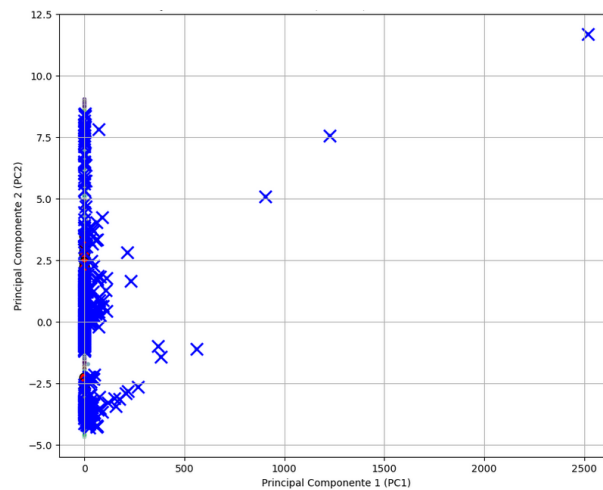


Figura 2.10: Visualização do agrupamento K-means com PCA

A visualização dos *clusters*, como ilustrado no gráfico, confirma esta falha na detecção. Para possibilitar esta representação, recorreu-se à *Principal Component Analysis* (PCA), uma técnica de redução de dimensionalidade que transforma os dados originais em novas variáveis, preservando ao máximo a variabilidade dos dados. Assim sendo, os círculos vermelhos, que representam as anomalias, apresentam-se dispersos e não se sobrepõem aos ‘X’ azuis, que indicam os pontos que o modelo previu anomalias. O

aumento de *clusters* para 50, com a consequente alteração do limiar de anomalia, levou a um aumento substancial no número de Falsos Positivos. Este acréscimo de falsos alarmes, sem qualquer ganho na detecção de anomalias, deteriora ainda mais a utilidade prática do modelo. Embora o ROC-AUC tenha mostrado uma ligeira melhora de 0,5592 para 0,6152, este valor continua a ser fraco e próximo de um classificador aleatório. O PR-AUC permaneceu em 0,0001, confirmando assim a sua ineficácia na identificação da classe minoritária.

A premissa fundamental do *K-means* não se mostrou eficaz para o conjunto de dados em estudo. A sua incapacidade de encontrar anomalias e a tendência para gerar múltiplos falsos positivos tornam-no inadequado para um cenário real de detecção de anomalias e intrusões.

2.4.2.3 DBSCAN

A adoção do algoritmo DBSCAN como abordagem de detecção de anomalias foi motivada pela sua capacidade teórica de identificar *clusters* de densidades variadas e de isolar pontos de baixa densidade (interpretados como ruído), que podem representar potenciais anomalias. A sua configuração dependeu de dois parâmetros fundamentais, nomeadamente:

- ‘eps’ – o raio máximo que define a vizinhança de um ponto;
- ‘min_samples’ – o número mínimo de pontos necessários dentro dessa vizinhança para que um ponto seja considerado “central” e, consequentemente, formar um *cluster*.

Foram realizadas duas implementações distintas, visando avaliar o impacto destes parâmetros na sensibilidade do modelo à detecção de anomalias.

Na primeira configuração, o parâmetro ‘min_samples’ foi definido como 594, valor calculado a partir da regra prática de $2 * \text{número de features}$, justificada pela elevada dimensionalidade do *dataset*. O parâmetro ‘eps’ foi definido como 0,5, valor padrão amplamente utilizado como ponto de partida.

Esta configuração revelou-se excessivamente restritiva. O elevado número mínimo de vizinhos exigido para a formação de um *cluster* impediu que muitos pontos fossem considerados “centrais”. Como consequência, a grande maioria das observações foi classificada como ruído, comprometendo a capacidade do modelo de distinguir entre comportamento normal e anómalo.

Com o propósito de mitigar este efeito, uma segunda tentativa foi realizada, reduzindo ‘min_samples’ para 30, mantendo ‘eps = 0,5’. Esta alteração procurou flexibilizar a definição de densidade mínima, permitindo que mais pontos fossem reconhecidos como pertencentes a *clusters* legítimos.

A expectativa era que este ajuste favorecesse a formação de agrupamentos representativos do comportamento normal, diminuindo o número de falsos positivos e, consequentemente, melhorar a *Precision* do modelo.

Deste modo, os resultados obtidos foram os seguintes:

Tabela 2.5: Resultados comparativos de desempenho do modelo DBSCAN em configurações ‘min_samples’

Métricas	DBSCAN (594)	DBSCAN (30)
<i>Precision</i>	0,00	0,00
<i>Recall</i>	1,00	1,00
F1-score	0,00	0,00
Falsos Positivos	1,783,931	1,662,820
Falsos Negativos	0	0
ROC-AUC	0,7301	0,8425
PR-AUC	0,0002	0,0003

A análise da implementação demonstrou que o DBSCAN, em ambas as configurações, foi ineficaz na detecção de anomalias. Apesar da redução drástica do ‘min_samples’, as métricas de desempenho para a classe ‘Anomalia’ permaneceram virtualmente idênticas.

Em ambas as tentativas, o modelo alcançou um *Recall* perfeito de 1.00, o que significa que todas as 48 ou 190 anomalias no conjunto de treino foram corretamente classificadas como ‘Anomalia’. No entanto, o valor obtido para o *Recall* foi à custa de uma *Precision* de 0,00, indicando que o modelo classificou incorretamente um número elevado de 1,783,931 instâncias normais como anomalias.

A ausência de melhoria nas métricas de desempenho entre as duas tentativas sugere que, para o ‘eps’ de 0,5 e a estrutura dos seus dados, a grande maioria dos pontos já estava muito dispersa, sendo classificada como ruído independentemente da definição mais ou menos restritiva de “ponto central”.

Desta forma, o modelo DBSCAN demonstrou que, apesar da sua capacidade teórica de identificar ruído, acaba por não se adequar como um classificador direto para este problema em questão. O seu desempenho desequilibrado, com um *Recall* perfeito, mas uma *Precision* nula, torna-o inviável para um sistema real de deteção de anomalias e intrusões.

2.4.2.4 Comparação Teórico-prática

A comparação entre as previsões teóricas e os resultados obtidos experimentalmente revela uma discrepância substancial no desempenho dos modelos de *Machine Learning* Não Supervisionado na deteção de anomalias. A literatura atribui ao Isolation Forest uma elevada eficácia e escalabilidade, ao K-means rapidez e simplicidade de implementação, e ao DBSCAN robustez na identificação de padrões de densidade. Todavia, a validação experimental evidenciou limitações críticas em todos.

O Isolation Forest e o K-means não identificaram qualquer anomalia, apresentando vários nulos de *Precision* e *Recall*, o que demonstra, que, neste contexto, as anomalias não se distinguem nem por critérios de isolamento aleatório nem por métricas de distância de centroides.

O DBSCAN, embora tenha alcançado um *Recall* perfeito, fê-lo à custa de um volume insustentável de falsos positivos, inviabilizando a sua utilização operacional. Estes resultados indicam que, em cenários caracterizados por uma elevada desproporcionalidade e complexidade na distribuição das anomalias, abordagens puramente não supervisionadas tendem a ser ineficazes.

2.4.3 Deep Learning Supervisionado

A presente secção aborda a implementação prática dos modelos de *Deep Learning* Supervisionados. São descritos os processos de treino, validação e avaliação das diferentes arquiteturas, assegurando a uniformidade metodológica face às abordagens de *Machine Learning*.

2.4.3.1 LSTM

O modelo LSTM foi selecionado pela sua reconhecida capacidade de capturar dependências de longo prazo em dados sequenciais, característica particularmente relevante no contexto de análise de *logs*, onde a relação temporal entre eventos pode indicar padrões anómalos.

A preparação dos dados iniciou-se através do conjunto previamente estruturado. No entanto, em arquiteturas *Deep Learning* recorrentes, como o LSTM, a estrutura de entrada requer uma entrada tridimensional, de forma a capturar as dependências temporais entre as amostras e refletir a natureza sequencial dos dados. Assim, foi necessário converter os dados do formato 2D para 3D, seguindo a estrutura ('n_samples', 'n_timesteps', 'n_features').

Com o objetivo de reduzir a complexidade computacional e evitar erros de memória, o parâmetro 'n_timesteps' foi definido como 1, eliminando a necessidade de segmentação temporal explícita.

A arquitetura do modelo foi desenvolvida sob uma estrutura Sequential, composta por:

- Uma camada LSTM com 64 unidades (neurónios), responsável por capturar padrões temporais;
- Uma camada *Dropout* de 0,2, aplicada para prevenir *overfitting* através da regularização estocástica;
- Uma camada Dense (saída) com 1 neurónio e função de ativação sigmoide, adequada à tarefa de classificação binária.

De modo a mitigar o impacto da desproporcionalidade de classes, foram calculados e aplicados 'class_weights', ajustando o contributo de cada classe durante o treino.

O algoritmo foi compilado com o otimizador Adam e a função de perda *Binary Cross-Entropy*, tendo o treino sido acelerado por GPU, o que permitiu uma execução eficiente.

Para garantir a estabilidade e generalização do modelo, foi incorporado um *callback* de *EarlyStopping* com 'patience=3', monitorizando a perda de validação e interrompendo o treino em caso de ausência de melhoria significativa, prevenindo assim o sobre ajuste e o desperdício de recursos computacionais.

Desta forma, após o treino do modelo, foram obtido os seguintes resultados:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	450012
1	0.59	0.96	0.73	48
accuracy			1.00	450060
macro avg	0.79	0.98	0.87	450060
weighted avg	1.00	1.00	1.00	450060

Figura 2.11: LSTM – Relatório de Classificação

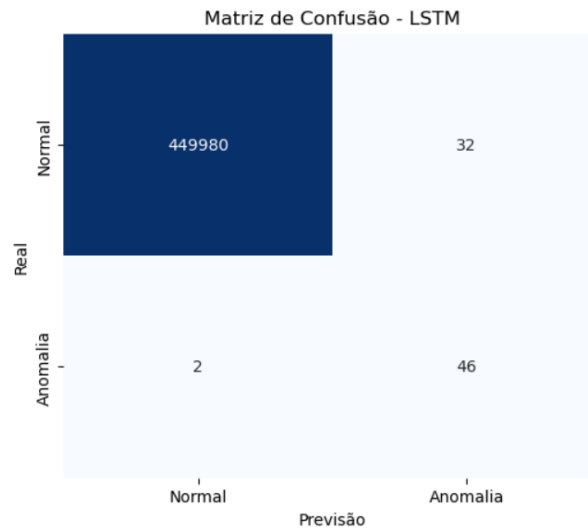


Figura 2.12: LSTM – Matriz de Confusão

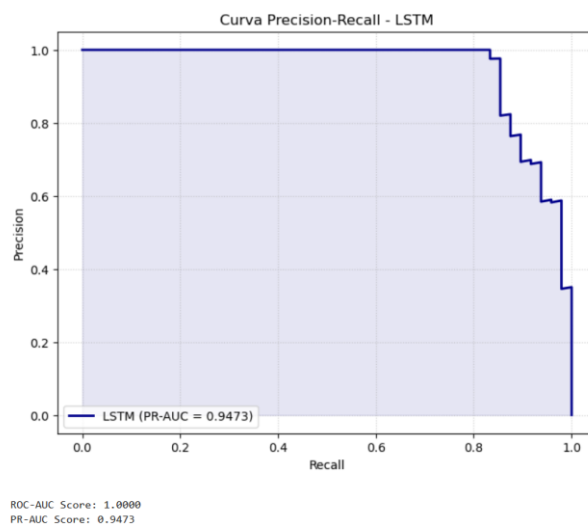


Figura 2.13: LSTM – Curva *Precision-Recall*

O modelo LSTM demonstrou um desempenho consideravelmente forte, alcançando um bom equilíbrio entre as métricas. O *Recall* de 0,96 para anomalias indica que o modelo conseguiu detetar 46 das 48 anomalias, registrando apenas 2 Falsos Negativos. A sua *Precision* de 0,59 reflete um compromisso entre a sensibilidade do modelo e a ocorrência de falsos positivos, o que é expectável num cenário de desbalanceamento extremo onde o modelo é incentivado a não ignorar ameaças raras.

Este *trade-off* entre *Recall* e *Precision* é favorável para um sistema de segurança que prioriza a visibilidade de ataques garantindo que a grande maioria das ameaças seja sinalizada. O ROC-AUC Score de 1,000 e o PR-AUC Score de 0,9473 são excelentes indicadores da sua capacidade de distinção e robustez.

2.4.3.2 BERT

A integração do modelo BERT neste trabalho foi inicialmente considerada com o objetivo de avaliar o seu desempenho em comparação com os modelos clássicos, dado que se trata de um modelo pré-treinado em larga escala. É apontado como referência em tarefas de classificação de texto, ao permitir compreender o contexto semântico de forma mais profunda, superando limitações das abordagens sequenciais tradicionais.

No entanto, a implementação prática revelou constrangimentos significativos de natureza técnica. O pacote *Transformers*, responsável pela disponibilização do modelo, oferece versões distintas para PyTorch e TensorFlow. No caso particular da classe ‘*TFBertForSequenceClassification*’, a sua execução depende da conversão de pesos pré-treinados originalmente disponibilizados em PyTorch para o formato TensorFlow. Esta conversão é efetuada automaticamente pela biblioteca ao definir o parâmetro ‘*from_pt = True*’, que indica que os pesos devem ser carregados a partir de um modelo pré-treinado em PyTorch.

Foi precisamente neste ponto que surgiram incompatibilidades. A versão atual do keras (3.x) deixou de suportar determinadas funções de baixo nível, como `keras.backend.set_value`, que são utilizadas internamente pela biblioteca *Transformers* durante o processo de conversão dos pesos. Assim, o carregamento do modelo resultava sistematicamente em erros de compatibilidade entre *frameworks*. Foram realizadas diversas tentativas, nomeadamente, a instalação de versões anteriores de TensorFlow (2.12.x) e a utilização do pacote alternativo tf-Keras, mais próximo do Keras 2. Estas tentativas mostraram-se inviáveis no ambiente experimental utilizado, devido à indisponibilidade de versões compatíveis para o SO e ao surgimento de novos conflitos entre bibliotecas.

A resolução definitiva deste problema exigiria a reescrita integral do código de treino e avaliação do modelo BERT em PyTorch, uma vez que a versão TensorFlow deixou de ser funcional nas configurações atuais. Tal implicaria não apenas alterar a forma de construção e compilação do modelo, mas também adaptar todo o *pipeline* de preparação de dados, otimização e avaliação, de modo a alinhar-se com a API nativa do

PyTorch. Embora tecnicamente viável, tal reestruturação comprometeria a validade da comparação entre modelos, pois deixaria de ser claro se as diferenças de desempenho resultariam do modelo em si ou da *framework* utilizada.

Em suma, a impossibilidade de implementar o BERT neste trabalho não se deveu a limitações conceituais do modelo, mas sim a questões de compatibilidade entre *frameworks* (TensorFlow/Keras e PyTorch) e à consequente necessidade de reescrever todo o código experimental em PyTorch. Não obstante, a tentativa de implementação revelou-se instrutiva, na medida em que ilustra os desafios concretos da aplicação de arquiteturas de ponta em ambientes reais.

2.4.3.3 GRU

O modelo *Gated Recurrent Unit* foi implementado como alternativa ao LSTM, com o intuito de avaliar o impacto de uma arquitetura recorrente mais leve na detecção de anomalias. A principal motivação para esta escolha reside na eficiência do GRU em contextos de elevada carga computacional, uma vez que a sua estrutura simplificada (combina os mecanismos de atualização e de reinicialização num único vetor de controlo) reduz o número total de parâmetros e acelera o treino, mantendo, contudo, uma capacidade comparável para capturar dependências temporais de longo prazo.

A arquitetura adotada seguiu a mesma estrutura base utilizada no modelo LSTM, construída através de um modelo *Sequential*. Assim, manteve-se uma configuração equivalente, composta por uma camada GRU com 64 unidades, seguida de uma camada *Dropout* de 0,2 para reduzir o risco de *overfitting*. A camada de saída (*Dense*) foi igualmente definida com um único neurónio e função de ativação sigmoide, adequada à tarefa de classificação binária.

Com base neste processo, obtiveram-se os seguintes resultados:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	450012
1	0.70	0.94	0.80	48
accuracy			1.00	450060
macro avg	0.85	0.97	0.90	450060
weighted avg	1.00	1.00	1.00	450060

Figura 2.14: GRU – Relatório de Classificação

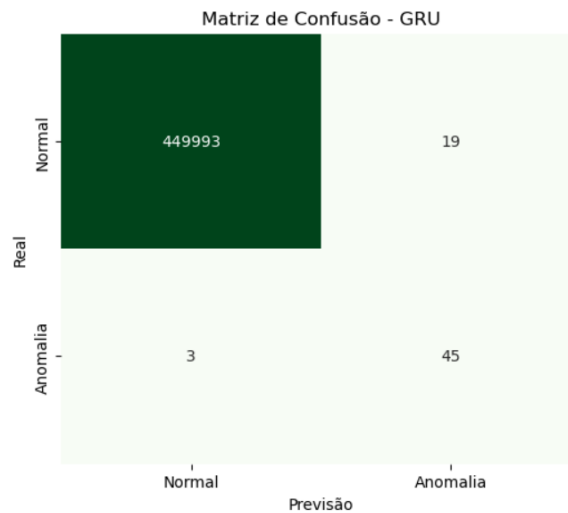


Figura 2.15: GRU – Matriz de Confusão

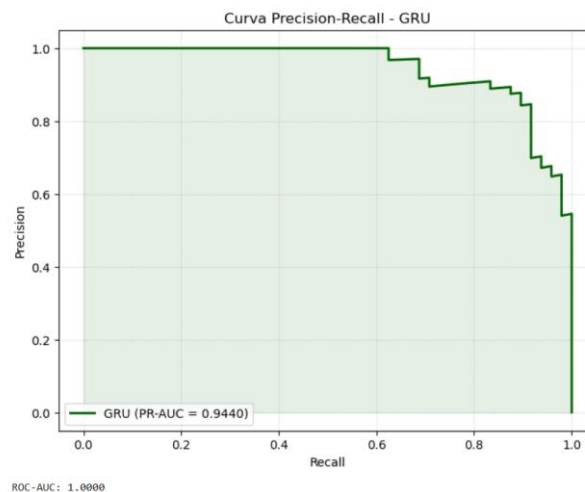


Figura 2.16: GRU – Curva *Precision-Recall*

O relatório de classificação mostra que o modelo GRU alcançou um *Recall* de 0,94. Este resultado é particularmente relevante, pois significa que o modelo conseguiu identificar corretamente 96% das anomalias presentes no conjunto de teste. Em termos práticos, das 48 anomalias existentes, apenas três não foram detetadas. Esta elevada taxa de deteção é fundamental em cenários críticos, onde a falha em reconhecer uma anomalia pode ter consequências severas.

Relativamente à *Precision* da classe anomalia, o valor obtido foi de 0,70. Este resultado indica que, entre todas as instâncias que o modelo classificou com anomalia, cerca de 70% correspondiam, de facto, a anomalias reais. Embora este valor seja mais modesto em comparação com o *Recall*, é considerado aceitável, especialmente em contextos de deteção de anomalias, onde a prioridade costuma ser maximizar o *Recall* para reduzir o risco de deixar passar eventos anómalos.

O F1-score da classe anomalia, que combina *Precision* e *Recall* numa métrica única, atingiu o valor de 0,80. Este resultado traduz um equilíbrio adequado entre a capacidade do modelo em identificar anomalias e a fiabilidade das previsões realizadas, reforçando a robustez do desempenho obtido.

A matriz de confusão auxilia na contextualização destes valores. O modelo registou 45 verdadeiros positivos, identificando corretamente quase todas as anomalias, e apenas 3 falsos negativos, ou seja, anomalias que não foram reconhecidas como tal. Por outro lado, surgiram 19 falsos positivos, casos em que o modelo classificou instâncias normais como anómalas, e 449,983 verdadeiros negativos, evidenciando a elevada precisão na identificação de instâncias normais.

No que diz respeito às métricas baseadas em curvas de desempenho, o modelo obteve um ROC-AUC de 1,0000, valor perfeito que indica uma separação completa entre as duas classes (anomalias e normais). Embora as métricas como o ROC-AUC possam ser menos informativas em contextos de *datasets* desproporcionais, este resultado confirma a excelente capacidade discriminatória do modelo.

Complementarmente, o PR-AUC (0,9534) constitui uma métrica mais adequada nestes cenários e reforça a conclusão de que o modelo GRU é altamente eficaz na deteção da classe minoritária, assegurando um desempenho notavelmente elevado mesmo perante a forte desproporção entre as classes.

Em síntese, os resultados demonstram que o modelo não só consegue identificar a esmagadora maioria das anomalias, como também apresenta métricas de desempenho muito robustas em contextos de dados desproporcionais. Embora exista algum custo em termos de falsos positivos, este *trade-off* é geralmente aceitável em sistemas de deteção de anomalias, uma vez que a prioridade reside na minimização dos falsos negativos.

2.4.3.4 Comparação Analítica

Tabela 2.6: Comparação dos resultados dos modelos *Deep Learning* Supervisionados

	LSTM	BERT	GRU
<i>Precision</i>	0,59	-	0,70
<i>Recall</i>	0,96	-	0,94
F1-score	0,73	-	0,80
Falsos Positivos	32	-	19
Falsos Negativos	2	-	3
ROC-AUC	1,00	-	1,00
PR-AUC	0,9473	-	0,9440

De acordo com os resultados obtidos, é possível verificar que o modelo LSTM demonstrou um equilíbrio notável entre *Precision* e *Recall*, resultando num F1-score de 0,89. Este desempenho traduziu-se numa baixa incidência de falsos negativos e de falsos positivos, evidenciando a sua robustez enquanto modelo consistente e fiável. A métrica PR-AUC de 0,9470 reforça a sua capacidade de distinguir eficazmente a classe minoritária, mesmo perante um *dataset* altamente desproporcional. Este comportamento mostra que o LSTM consegue manter um compromisso sólido entre maximizar a taxa de deteção de anomalias e minimizar os falsos alarmes, sendo um aspeto crucial em sistemas de monitorização de segurança.

Relativamente ao modelo GRU, embora apresente uma arquitetura mais simples e computacionalmente eficiente, destacou-se principalmente pelo seu valor elevado no *Recall*. Na prática, isto significou a correta identificação de 46 das 48 anomalias presentes no conjunto de teste. Contudo, esta maior sensibilidade veio acompanhada de uma redução da *Precision*, resultante de um número mais elevado de Falsos Positivos. Apesar deste custo, o valor obtido para o PR-AUC confirma que o GRU é extremamente eficaz na deteção de instâncias da classe minoritária, mesmo que à custa de algum aumento no ruído gerado.

Deste modo, a escolha entre os dois modelos depende das prioridades do sistema de deteção. O LSTM apresenta-se como a solução mais equilibrada, ideal para cenários em que a precisão das previsões é tão importante quanto a taxa de deteção, reduzindo o risco de fadiga do operador devido a falsos alarmes. O GRU, por sua vez, mostra-se mais

adequado em contextos onde a minimização de falsos negativos é a prioridade absoluta, aceitando-se o *trade-off* de lidar com um número superior de falsos positivos.

2.4.3.5 Comparação Teórico-prática

A comparação entre as expectativas teóricas e os resultados práticos evidencia a importância da validação experimental no contexto da detecção de anomalias. Do ponto de vista teórico, o BERT é apontado como a solução mais promissora, dado o seu papel de referência em tarefas de processamento de linguagem natural. O LSTM, por sua vez, surge como um modelo robusto e equilibrado, enquanto o GRU destaca-se pela sua simplicidade e eficiência computacional.

Na prática, contudo, os resultados demonstraram que o GRU foi o modelo mais indicado para o problema em estudo, alcançando um *Recall* de 0,96 e reduzindo quase ao mínimo os falsos negativos. O LSTM revelou-se uma alternativa consistente e equilibrada, com bom compromisso entre *Precision* e *Recall*, adequado a cenários em que a estabilidade é prioritária. O BERT não pôde ser implementado devido a incompatibilidade técnicas e à necessidade de migração para *PyTorch*, mas permanece como uma referência teórica e uma perspectiva para trabalhos futuros.

2.4.3.6 Otimização GRU

A otimização do modelo GRU teve como principal objetivo melhorar o equilíbrio entre as métricas de *Precision* e *Recall*, procurando reforçar a capacidade do modelo em detetar anomalias de forma fiável e consistente. No modelo base, verificava-se uma discrepância entre estas métricas, o que motivou uma reestruturação arquitetónica orientada para um melhor compromisso entre a sensibilidade e a especificidade da detecção.

A estratégia de otimização consistiu em modificar a arquitetura da rede, evoluindo de uma única camada GRU para uma configuração empilhada composta por duas camadas sucessivas. A primeira camada foi definida com 64 unidades e o parâmetro “*return_sequences=True*”, permitindo a passagem das sequências temporais à camada seguinte, enquanto a segunda camada, com 32 unidades, foi responsável por aperfeiçoar as dependências temporais aprendidas.

Entre as duas camadas GRU foram aplicadas camadas *Dropout*, com o objetivo de reduzir o risco de sobre-ajuste. A camada de saída manteve a estrutura anterior, composta por um único neurónio com ativação sigmóide, adequada à tarefa de classificação binária. O processo de treino seguiu a metodologia previamente

estabelecida, recorrendo ao otimizador Adam, à função de perda ‘binary_crossentropy’ e à aplicação de *class weights* para mitigar a desproporcionalidade entre classes. O treino foi acelerado por GPU e monitorizado através do mecanismo *EarlyStopping* com ‘patience=3’, assegurando a estabilidade e eficiência do processo.

Tabela 2.7: Desempenho comparativo entre o modelo GRU base e otimizado

Métricas	GRU Base	GRU Otimizado
<i>Precision</i>	0,70	0,91
<i>Recall</i>	0,94	0,85
<i>F1-score</i>	0,80	0,88
Falsos Positivos	19	4
Falsos Negativos	3	7
ROC-AUC	1,00	1,00
PR-AUC	0,9440	0,9253

Ao analisar os resultados, verifica-se uma alteração significativa no equilíbrio entre as métricas de desempenho. A *Precision* registou uma melhoria notável, passando de 0,59 para 0,91, o que se traduziu numa redução substancial do número de Falsos Positivos, que diminuíram de 32 para apenas 4. Em paralelo, o *F1-score* evoluiu de 0,73 para 0,88, refletindo uma melhoria geral na coerência do modelo e na sua estabilidade preditiva. No entanto, este ganho em precisão foi acompanhado por uma ligeira redução no *Recall*, que passou de 0,96 para 0,85, o que implicou um aumento dos falsos negativos de 2 para 7.

Apesar desta diminuição, o modelo otimizado estabeleceu um ponto de equilíbrio mais favorável, privilegiando a fiabilidade e a consistência das previsões. Relativamente às restantes métricas, os níveis mantiveram-se elevados com o ROC-AUC a permanecer perfeito e o PR-AUC apresentou apenas uma ligeira variação, passando de 0,9834 para 0,9253.

Em síntese, a otimização do algoritmo não resultou num avanço absoluto em todas as métricas, mas permitiu redefinir de forma eficaz o compromisso entre *Recall* e *Precision*. O modelo final mostrou-se mais estável e confiável, reduzindo significativamente o número de alarmes falsos e apresentando um comportamento mais equilibrado e adequado a cenários de deteção de anomalias.

2.4.4 Deep Learning Não Supervisionado

Nesta secção são conduzidas as experiências correspondentes aos modelos de *Deep Learning* não supervisionados. São apresentados as etapas de configuração, treino e análise dos resultados, com o intuito de avaliar a sua eficácia prática no contexto do *dataset* em estudo.

2.4.4.1 VAE

O *Variational Autoencoder* foi implementado com o objetivo de aprender uma representação compacta da “normalidade” dos dados, utilizando o erro de reconstrução como critério principal para a deteção de anomalias.

O algoritmo foi treinado exclusivamente com instâncias da classe normal, partindo do pressuposto de que este subconjunto representa o comportamento legítimo do sistema. Assim, qualquer registo que apresente um erro de reconstrução significativamente superior ao padrão aprendido é interpretado como potencial anomalia.

A arquitetura do modelo seguiu a estrutura clássica dos VAEs, composta por um codificador, um espaço latente e um decodificador. O codificador mapeia os dados originais para distribuições gaussianas no espaço latente, enquanto o decodificador tenta reconstruir as entradas a partir dessas representações comprimidas. Para aumentar a capacidade representativa do modelo, foram ajustadas as dimensões intermédias (128 neurónios nas camadas densas e 10 dimensões no espaço latente), permitindo uma melhor captação das variações subtis nos dados normais.

O treino foi conduzido com o otimizador Adam, ao longo de 100 épocas, integrando um mecanismo de *Early Stopping* com ‘patience=10’ para evitar *overfitting*. A avaliação baseou-se no erro de reconstrução, definindo-se um limiar de deteção calculado a partir dos percentis do conjunto de treino. As instâncias cujo erro ultrapassava esse limiar foram classificadas como anómalas.

Após o treino, o modelo foi avaliado no conjunto de teste, obtendo os seguintes resultados:

```

--- Relatório de Classificação (VAE) ---
      precision    recall  f1-score   support

     0         1.00      1.00      1.00     450012
     1         0.00      0.00      0.00         48

 accuracy         1.00     450060
 macro avg       0.50      0.50      0.50     450060
 weighted avg    1.00      1.00      1.00     450060

```

Figura 2.17: VAE – Relatório de Classificação

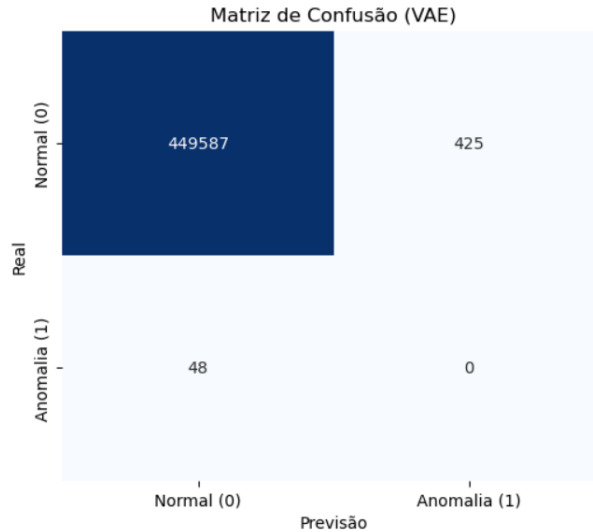


Figura 2.18: VAE – Matriz de Confusão

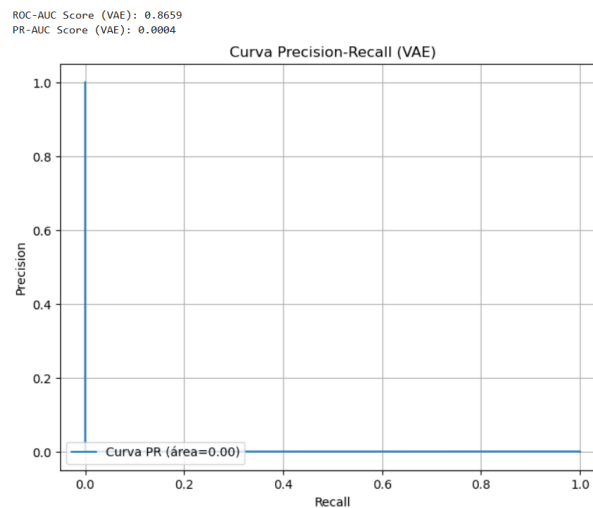


Figura 2.19: VAE – Curva *Precision-Recall*

Apesar da solidez metodológica, os resultados obtidos revelaram-se insatisfatórios. As métricas de avaliação, nomeadamente o *Recall*, indicam uma incapacidade clara do modelo em identificar anomalias de forma consistente. Na melhor das tentativas, o *Recall* não ultrapassou 0,19, tendo chegado a valores próximos de zero em diversos *thresholds*. Isto significa que o modelo falhou em detetar a grande maioria

das instâncias anómalas, classificando-as erradamente como normais. É ainda de maior gravidade, o facto de que o erro de reconstrução associado às anomalias não diferia significativamente do erro produzido para instâncias normais, ou seja, o VAE conseguiu reconstruir as anomalias com a mesma eficácia que os dados normais, tornando o critério do limiar ineficaz enquanto mecanismo de deteção. As restantes métricas, como a área sob a curva *Precision-Recall*, confirmam este diagnóstico, evidenciando valores muito baixos e reforçando a conclusão de que a abordagem não se revelou adequado ao *dataset* em análise.

A principal limitação identificada prende-se com a natureza intrínseca dos dados. As anomalias, apesar de raras, não apresentavam padrões suficientemente distintos que dificultassem a sua reconstrução. Deste modo, o modelo *Variational Autoencoder*, cuja lógica assenta precisamente na dificuldade em reproduzir instâncias fora do comportamento normal, não encontrou evidências suficiente para diferenciar classes.

Perante este cenário, a investigação conduzida permite concluir que o VAE não constitui uma abordagem eficaz para a deteção de anomalias neste contexto específico.

2.4.4.2 LogBERT

O LogBERT foi inicialmente considerado como uma possível abordagem para a deteção de anomalias neste trabalho, dado que representa uma especialização do modelo BERT direcionada para o domínio dos *logs*. A sua principal vantagem conceptual reside no facto de, ao contrário dos modelos clássicos ou sequenciais, ser capaz de modelar de forma explícita o contexto e a sequência dos eventos, capturando dependências de longo alcance que podem ser cruciais para identificar padrões anómalos. Teoricamente, esta característica colocaria o LogBERT como uma opção de ponta para a tarefa em estudo.

Contudo, a sua implementação prática revelou-se inviável, e as razões para tal encontram-se diretamente ligadas às dificuldades encontradas previamente com a integração do BERT. Tal como foi descrito anteriormente, a tentativa de utilização do algoritmo BERT em TensorFlow/Keras defrontou-se em incompatibilidade persistentes com a biblioteca *Transformers*, resultando em erros recorrentes no carregamento de modelos pré-treinados. A única alternativa consistente passaria por uma reimplementação integral em PyTorch, o que não apenas exigiria a adaptação de todo o *pipeline* de treino e avaliação, mas também comprometeria entre modelos, uma vez que todos os restantes algoritmos foram desenvolvidos em TensorFlow.

Neste contexto, o LogBERT herdaria inevitavelmente os mesmos constrangimentos técnicos. Sendo uma extensão direta do modelo BERT, enfrenta não apenas os elevados custos computacionais típicos das arquiteturas *Transformer*, mas também as mesmas limitações de compatibilidade entre *frameworks*. A sua integração implicaria, tal como no BERT, abandonar o ambiente uniforme baseado em TensorFlow e introduzir uma dependência exclusiva de PyTorch, criando assimetria metodológica no trabalho. Esta disparidade teria consequências sérias para a validade da comparação entre modelos, uma vez que deixaria de ser claro se as diferenças de desempenho se deviam à qualidade do algoritmo ou às especificidades da *framework* utilizada.

2.4.4.3 DeepLog

O algoritmo DeepLog foi implementado com o propósito de avaliar a eficácia de uma abordagem baseada em *autoencoders* recorrentes, que recorre a unidades GRU para a modelação de padrões de comportamento normal em *logs* de rede. Fundamentado na premissa de que instancias anómalas apresentam um erro de reconstrução superior às normais, o modelo utiliza este erro como critério principal de deteção de anomalias.

A arquitetura foi desenvolvida segundo uma estrutura sequencial composta por duas camadas GRU empilhada, com 128 e 64 unidades, respetivamente, seguidas de uma camada *TimeDistributed*, responsável pela reconstrução das sequências originais.

Devido à desproporcionalidade extrema das classes, foi aplicada uma amostragem estratégica de forma a equilibrar parcialmente o treino. O conjunto de treino final integrou 100,000 amostras da classe ‘Normal’ e 190 instâncias da classe ‘Anomalia’, correspondendo a cerca de 5,56% do *dataset* de treino original.

O modelo foi treinado utilizando o MSE como função de perda, com o otimizador Adam, e integrado com *EarlyStopping* de ‘patience=5’. Após o treino, o desempenho foi avaliado com base no erro de reconstrução.

A avaliação do modelo revelou um desempenho baixo, como podemos verificar de acordo com as métricas seguintes:

```

--- Relatório de Classificação do Modelo Deeplog ---
      precision    recall  f1-score   support

      0.0         1.00      1.00      1.00     450012
      1.0         0.00      0.00      0.00         48

 accuracy          1.00     450060
 macro avg          0.50      0.50      0.50     450060
 weighted avg          1.00      1.00      1.00     450060

```

Figura 2.20: *DeepLog* – Relatório de Classificação

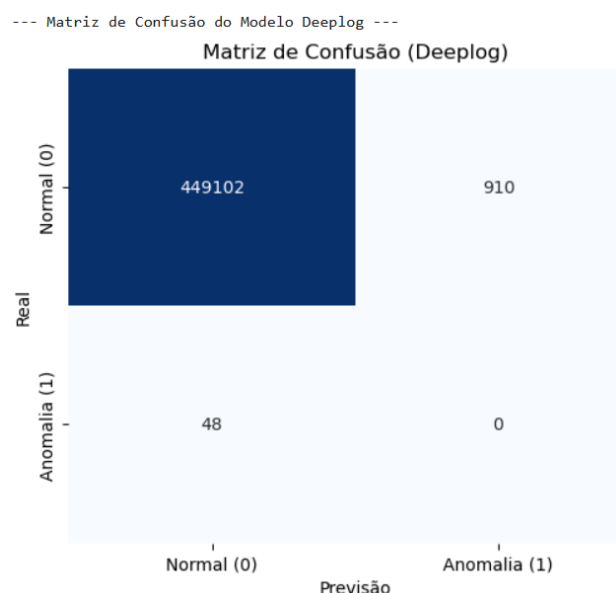


Figura 2.21: *Deeplog* - Matriz de Confusão

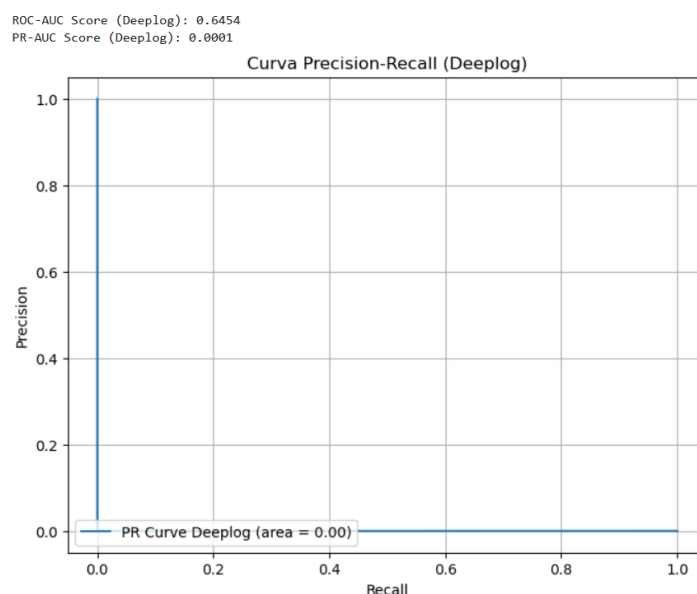


Figura 2.22: *DeepLog* - Curva *Precision-Recall*

As métricas de desempenho demonstram que o modelo falhou completamente na detecção de anomalias, apresentando valores nulos de *Recall*, *Precision* e *F1-score*. Em termos práticos, isto significa que o *Deeplog* classificou todas as 48 anomalias como eventor normais, não identificando qualquer instância anômala. O PR-AUC de 0,0001 reforça este diagnóstico, indicando que o modelo se comportou de forma estatisticamente equivalente ao caso na distinção de classes.

Esta limitação não decorre de aspetos técnicos como o volume de treino ou parâmetros de otimização, mas antes de uma incompatibilidade estrutural entre a natureza do modelo e as características do *dataset*. O *Deeplog* assume implicitamente que as

anomalias apresentam padrões de reconstrução substancialmente distintos da normalidade, o que não se verificou neste conjunto de dados, onde o comportamento anômalo não é suficientemente distinto em termos de sequencialidade temporal.

2.4.4.4 Comparação Teórico-prática

A comparação entre as expectativas teóricas e os resultados práticos dos modelos *Deep Learning* Não Supervisionado revelou uma discrepância entre o potencial esperado e o desempenho obtido. Teoricamente, os modelos descritos anteriormente são referidos na literatura pela sua capacidade de aprender representações latentes e identificar anomalias com base em desvios do comportamento normal.

Na prática, contudo, os resultados demonstraram que tanto o VAE como o DeepLog falharam na detecção de anomalias, apresentando valores nulos de *Recall* e *Precision* para a classe minoritária. O LogBERT, embora promissor em termo teóricos, não pode ser implementado devido a incompatibilidade técnica entre *frameworks*. Assim, conclui-se que, apesar do seu valor conceptual, os modelos *Deep Learning* Não Supervisionado mostraram limitações significativas no contexto de análise, evidenciando a dependência destas abordagens da qualidade e estrutura das *features* disponíveis.

2.4.5 Síntese da Avaliação de Desempenho e Implicações Metodológicas

A análise comparativa dos diferentes modelos de *Machine Learning* e *Deep Learning* permitiu compreender de forma mais clara as implicações práticas das decisões metodológicas tomadas ao longo da investigação. De modo geral, os modelos supervisionados destacaram-se por apresentar desempenhos consistentes e robustos, refletindo a eficácia das estratégias de treino baseadas em rótulos explícitos.

Em contraste, os modelos não supervisionados, revelaram limitações significativas na sua capacidade de distinguir eventos normais de anómalos. Esta diferença decorre essencialmente da natureza do *dataset* e das transformações aplicadas durante o pré-processamento, que reduziram a presença de variáveis comportamentais essenciais à identificação de desvios estruturais. Sem acesso a estas relações explícitas ou dependências temporais fortes, os modelos não supervisionados tenderam a reconstruir tanto dados normais como anómalos com a mesma eficiência, tornando-se incapazes de efetuar uma separação clara entre as classes.

Desta forma, a análise metodológica evidencia que o sucesso dos modelos supervisionados advém diretamente da existência de uma *Target* bem definida e de um processo de treino guiado por *feedback* explícito, enquanto os métodos não

supervisionados dependem fortemente da riqueza contextual e da variabilidade informacional das *features*. Estes resultados sustentam a decisão de prosseguir para a fase de simulações experimentais com o modelo de melhor desempenho, de modo a avaliar a sua resposta perante cenários de intrusão artificialmente gerados.

2.5 Simulação de Cenários de Ataque e Análise de Desempenho

A fase de simulação de cenários de ataque e análise de desempenho representa a etapa final do processo de validação experimental deste trabalho. Foi selecionado o modelo XGBoost, identificado no capítulo anterior como a abordagem mais promissora. O desempenho inicial do modelo no conjunto de teste, com métrica de *Recall* e *Precision* elevadas, validou a sua eficácia na deteção de anomalias com base na ‘Severity’. Contudo, para um sistema de deteção de intrusões, a verdadeira eficácia de um modelo reside na sua capacidade de identificar diferentes tipos de ataques em cenários práticos e dinâmicos.

Neste sentido, foram desenvolvidas e implementadas simulações de ataques cibernéticos comuns, injetando eventos sintéticos no conjunto de dados de teste para avaliar o desempenho do modelo contra ameaças específicas.

A metodologia adotada consiste em replicar os padrões de ataque mais relevantes e testar a capacidade do modelo de os detetar, analisando as métricas de desempenho para cada cenário. De forma a garantir uma avaliação abrangente, cada simulação foi executada em três intensidades de injeção distintas, nomeadamente, baixa, média e alta. Esta abordagem permite observar como o modelo se comporta à medida que a magnitude da ameaça aumenta, fornecendo uma visão mais completa sobre a sua resiliência e sensibilidade.

2.5.1 Simulação de DDoS

No âmbito do processo de validação experimental, foi conduzida uma simulação de DDoS com o objetivo de avaliar a capacidade do modelo XGBoost em reconhecer anomalias resultantes de um aumento súbito e coordenado do volume de tráfego. O cenário foi implementado sobre o conjunto de dados de teste, modificado de forma controlada para reproduzir o padrão característico deste tipo de ataque.

Para a sua execução, foi selecionado aleatoriamente um endereço de IP de destino, designado como o alvo do ataque. De modo a replicar o comportamento típico de um ataque DDoS, foram gerados endereços de IP de origem sintética, distintos e aleatórios, responsáveis por direcionar um grande volume de tráfego para o IP alvo.

A Tabela 2.8 apresenta as métricas de desempenho do modelo XGBoost otimizado no cenário de simulação de DDoS, em comparação com a seu desempenho de base no conjunto de teste original.

Tabela 2.8: Simulação de DDoS em três intensidades de injeção distintas

	Base (Sem Ataque)	DDoS (Baixo)	DDoS (Médio)	DDoS (Alto)
<i>Precision</i>	0,85	0,85	0,85	0,85
<i>Recall</i>	0,96	0,08	0,01	0,01
F1-score	0,90	0,15	0,02	0,01
Falsos Positivos	8	8	8	8
Falsos Negativos	2	499	2500	5000
ROC-AUC	0,99	0,72	0,66	0,66
PR-AUC	0,97	0,08	0,01	0,01

Os resultados evidenciaram um contraste significativo entre o desempenho do modelo no cenário base e nas simulações de ataque. No conjunto original, o XGBoost apresentou resultados sólidos, conforme demonstrado na tabela anterior, revelando uma excelente capacidade de detecção das anomalias de ‘Severity’ presentes nos dados. No entanto, quando exposto aos eventos artificiais de DDoS, o desempenho do modelo deteriorou-se drasticamente.

O *Recall*, caiu abruptamente para 0,09 na simulação de baixa intensidade e aproximou-se de zero nas restantes. Este colapso na taxa de detecção traduziu-se num número extremamente elevado de Falsos Negativos, indicando que a esmagadora maioria dos eventos de ataque foi classificada como tráfego normal. Em contrapartida, a *Precision* e o número de Falsos Positivos mantiveram-se praticamente inalterados em relação ao cenário base, evidenciando que o modelo continuou a classificar corretamente os registos normais, mas falhou completamente em reconhecer o comportamento anómalo introduzido.

A análise posterior permitiu compreender a origem desta falha. Durante o processo de pré-processamento do *dataset*, variáveis essenciais para a detecção de padrões de tráfego, nomeadamente, ‘Source IP’, ‘Destination IP’ e ‘timestamp’, foram removidas. O foco inicial do trabalho estava centrado na identificação de anomalias de gravidade e não em comportamentos de rede. Portanto, quando o código de simulação injetou IPs

sintéticos para a reprodução do ataque, o modelo ignorou completamente esses valores, pois nunca os tinha visto nem aprendido a utilizá-los como *features* relevantes durante o treino.

2.5.2 Simulação de Exfiltração de Dados

Após a simulação de DDoS, cujos resultados evidenciaram a incapacidade do modelo em detetar anomalias comportamentais, foi conduzida uma nova simulação centrada num ataque de Exfiltração de Dados.

Este tipo de ataque caracteriza-se por uma transferência anormalmente elevada de informação e, ao contrário do DDoS, baseia-se essencialmente em picos de volume de dados. Em teoria, este comportamento deveria alinhar-se com o tipo de anomalias de gravidade que o modelo foi treinado para reconhecer.

O objetivo desta experiência foi, portanto, avaliar se o modelo conseguiria identificar um ataque baseado em volume, mesmo na ausência de *features* contextuais de rede, como endereços de IP e *timestamps*.

A simulação foi realizada sobre um conjunto de dados de teste, replicando o padrão típico de uma exfiltração de dados, em que um sistema interno envia quantidades invulgarmente grandes de informação para o exterior. Deste modo, foram gerado eventos sintéticos através de uma amostra do conjunto de teste original, e os valores dos campos *Sent Bytes* e *Server Outbound Bytes* foram amplificados de forma significativa para representar a sobrecarga provocada por uma transferência anómala de dados.

A Tabela 2.9 apresenta as métricas de desempenho para a presente simulação, comparando-as com os resultados obtidos no conjunto de teste original.

Tabela 2.9: Simulação de Exfiltração de Dados em três intensidades de injeção distintas

	Base (Sem Ataque)	Exfiltração (Baixo)	Exfiltração (Médio)	Exfiltração (Alto)
<i>Precision</i>	0,85	0,85	0,85	0,86
<i>Recall</i>	0,96	0,09	0,02	0,01
<i>F1-score</i>	0,90	0,17	0,04	0,02
Falsos Positivos	8	8	8	8
Falsos Negativos	2	451	2251	4498
ROC-AUC	0,99	0,53	0,50	0,50
PR-AUC	0,97	0,09	0,02	0,02

Os resultados confirmaram a tendência observada no cenário anterior, o modelo falhou de forma categórica em detetar o ataque simulado. O *Recall* para a classe anómala caiu para valores próximos de 0,01, revelando que praticamente todos os eventos de exfiltração foram classificados como normais. O número de FN aumentou proporcionalmente ao volume de ataques injetados, demonstrando que o modelo não identificou as transferências de dados simulados.

A *Precision* e o número de FP permaneceram estáveis, mantendo-se próximos dos valores base, o que confirma que o modelo continuou a classificar corretamente os eventos normais, mas permaneceu insensível às novas anomalias de volume.

2.5.3 Simulação de SQL Injection

Com o intuito de aprofundar a análise de desempenho do modelo XGBoost, foi conduzida uma simulação de ataque SQL Injection, selecionada pela sua relevância prática e pela natureza distinta das anomalias que representa. Este tipo de ataque explora vulnerabilidades em aplicações *web* para manipular consultas de base de dados, estando, por isso, associado ao conteúdo e ao serviço de rede.

O objetivo desta experiência foi avaliar se o modelo seria capaz de identificar anomalias deste género, mesmo tendo sido treinado com um conjunto de *features* predominantemente orientadas à gravidade e ao volume de dados.

Desta forma, foram injetados eventos sintéticos com valores anómalos nas colunas ‘Service’ e ‘App Protocol’, simulando a ocorrência de solicitações suspeitas a serviços e aplicações *web*. Como nas simulações realizadas anteriormente, os *Send Bytes* e *Server Inbound Bytes* foram amplificados. Paralelamente, os campos ‘Service’ e ‘App Protocol’ destes eventos foram alterados para valores raros no *dataset*, refletindo padrões atípicos associados a este tipo de intrusão.

A tabela 2.10 apresenta as métricas de desempenho para o cenário SQL Injection, comparadas com o desempenho base no conjunto de teste original.

Tabela 2.10: Simulação de SQL Injection em três intensidades de injeção distintas

	Base (Sem Ataque)	SQL Injection (Baixo)	SQL Injection (Médio)	SQL Injection (Alto)
<i>Precision</i>	0,85	0,85	0,85	0,86
<i>Recall</i>	0,96	0,09	0,02	0,01
F1-score	0,90	0,17	0,04	0,02
Falsos Positivos	8	8	8	8
Falsos Negativos	2	451	2251	4498
ROC-AUC	0,99	0,53	0,50	0,50
PR-AUC	0,97	0,09	0,02	0,02

Os resultados seguiram o mesmo padrão observado nas simulações anteriores. O *Recall* do modelo sofreu uma queda abrupta, aproximando-se de 0,01 em todos os níveis de intensidade, o que indica que o modelo foi incapaz de detetar a maioria dos eventos de ataques injetados. O número de FN foi praticamente igual ao número total de anomalias simuladas, evidenciando que as alterações introduzidas não foram reconhecidas como comportamentos anómalos.

Por outro lado, a *Precision* e o número de FP permaneceram estáveis e idênticos aos valores de base, o que sugere que o modelo continuou a classificar corretamente o tráfego normal, mas não identificou os padrões correspondentes ao ataque.

Esta falha deve-se, mais uma vez, ao processo de pré-processamento e seleção de variáveis, que excluiu colunas críticas para a deteção de anomalias associadas ao conteúdo das comunicações, nomeadamente ‘Service’ e ‘App Protocol’.

Como resultado, o modelo foi privado de informações semânticas que caracterizam o comportamento típico de ataques deste tipo, reduzindo a sua capacidade de generalização para padrões não vistos durante o treino.

2.5.4 Análise da Simulação

O conjunto de simulações realizadas neste capítulo permitiu validar, de forma experimental, a eficácia e as limitações do modelo XGBoost na deteção de diferentes tipos de anomalias em ambiente controlado.

Através da reprodução dos três cenários distintos, foi possível observar o comportamento do modelo perante padrões de ameaça com características estruturais e semânticas diversas.

Apesar do seu desempenho considerável no conjunto de teste original, onde alcançou métrica de *Precision* e *Recall* elevadas para anomalias de ‘Severity’, o modelo revelou incapacidade de generalização quanto confrontado com ataques simulados. Em todos os cenários, o *Recall* desceu para valores residuais, demonstrando que o modelo não reconheceu as anomalias injetadas, mesmo quando estas envolviam variações significativas nos valores de tráfego ou em atributos de serviço.

A manutenção de valores estáveis de *Precision* e de Falsos Positivos indica que o modelo continuou a identificar corretamente os registos normais, mas falhou sistematicamente na deteção das novas instâncias de ataque.

A análise transversal das simulações permite concluir que esta limitação não decorre da escolha do algoritmo, mas sim da representação dos dados utilizados no treino. O processo de pré-processamento, ao remover variáveis contextuais como ‘Source IP’, ‘Destination IP’, ‘timestamp’, ‘Service’ e ‘App Protocol’, eliminou a informação necessária para a deteção de padrões de comportamento e de contexto, essenciais para a identificação de ataques reais.

Consequentemente, o modelo foi treinado para reconhecer anomalias de severidade intrínseca, mas não anomalias relacionais ou comportamentais, o que explica o seu fraco desempenho nos cenários simulados.

Os resultados obtidos constituem uma validação empírica das limitações dos modelos supervisionados baseados em *features* estáticas, como o XGBoost, no domínio da deteção de intrusões.

Em suma, a presente análise demonstra os efeitos das opções de pré-processamento sobre o desempenho dos modelos de deteção, evidenciando a necessidade de estratégias mais equilibradas entre simplificação dos dados e preservação de informação contextual.

2.6 Operacionalização e Integração em sistemas de Segurança

Após a fase de desenvolvimento, treino e validação dos modelos de aprendizagem automática, onde se avaliou a capacidade analítica do motor de deteção perante dados reais e cenário simulados, torna-se fundamental discutir a sua aplicação prática num contexto operacional. O valor de um modelo de IA em cibersegurança é potenciado pela

sua capacidade de integração com as ferramentas já existentes nas infraestruturas. Assim, este capítulo descreve a arquitetura conceptual necessária para converter o motor de análise num IDS funcional e a sua subsequente integração com plataformas de gestão de eventos, abordando os desafios de comunicação, normalização de dados e a eficácia na resposta a incidentes.

2.6.1 Enquadramento do Motor de IA na Arquitetura IDS

Para uma melhor compreensão de como o modelo desenvolvido se transforma num IDS operacional, utilizamos a arquitetura padrão de *Common Intrusion Detection Framework* (CIDF), que como já foi referido anteriormente, divide o sistema em quatro blocos funcionais:

- *E-box* – Corresponde à fonte dos dados. No contexto deste trabalho, são os *logs* brutos da instituição. Num cenário real, esta *box* seria responsável pela recolha continua do tráfego.
- *A-box* – Este é o núcleo do presente trabalho. O motor de IA assume o papel de “caixa de análise”, processando os dados da *E-box* para detetar anomalias. Como demonstrado nas simulações, a eficácia desta *box* está dependente da qualidade e contexto dos dados recebidos.
- *D-box* – Responsável por armazenar os eventos e as classificações feitas pela *A-box* para análise histórica e auditoria, permitindo que o sistema aprenda com eventos passados.
- *C-box* – É o modulo de resposta. Quando a *A-box* identifica uma severidade elevada, a *C-box* dispararia um alerta ou bloquearia um IP.

2.6.2 Integração com Ecossistemas SIEM

Enquanto o IDS trata da deteção técnica, o SIEM atua como a camada de gestão superior. A integração do moto desenvolvido com um SIEM permite:

- Normalização via *JavaScript Object Notation* (JSON) – A conversão das previsões do modelo num formato estruturado, facilitando a leitura por plataformas como o *Elastic Stack* ou *Splunk*.
- Redução de Ruído (*Alert Flooding*) – Através da conversão do *score* contínuo de severidade em categorias discretas (Baixo, Médio, Alto), evita-se que o analista de segurança seja inundado por alertas irrelevantes.

- Correlação de Eventos – O SIEM permite cruzar a “visão de severidade” do modelo com outras fontes (*firewalls* ou antivírus), colmatando a falta de contexto, que foi identificada com uma limitação nas simulações.

2.6.3 Viabilidade e Desafios de Implementação

A implementação prática do *pipeline* de integração entre o motor de IA desenvolvido e um ambiente operacional não foi realizada no âmbito deste trabalho, uma vez que tal etapa exigiria infraestruturas de produção e conhecimentos de engenharia de *software* que ultrapassem o objetivo analítico e os recursos disponíveis para este trabalho. No entanto, a viabilidade desta solução é sustentada pela arquitetura conceptual apresentada, embora a sua transição para um ambiente real enfrente desafios técnicos significativos, muitos dos quais foram identificados durante as fases de treino e simulação.

Um dos principais obstáculos reside na gestão de recursos e *hardware*. Conforme evidenciado anteriormente, o processamento de volumes massivos de dados, na ordem dos milhões de registos, resultou em erros críticos de memória no ambiente computacional utilizado. Em ambiente operacional, a viabilidade do sistema depende de infraestruturas escaláveis que permitam ao motor XGBoost processar os dados sem a necessidade de suprimir colunas contextuais, como endereços de IP e *timestamps*, que se revelaram essenciais para a deteção de ataques relacionais.

Somado a isto, a natureza da saída do modelo introduz um desafio de interpretabilidade. O motor final produz um valor contínuo que representa a severidade, o que, apesar de adequado para a modelação, pode originar o fenómeno de *alert flooding* em sistemas SIEM, que operam maioritariamente com categorias discretas. Para mitigar este risco e evitar a fadiga dos analistas, propõe-se uma transformação baseada em limiares definidos, mapeando o *score* matemático para níveis de risco operacionais.

Esta interoperabilidade seria concretizada através da estruturação dos eventos no formato JSON, padrão amplamente aceite pela indústria para garantir que o motor de análise comunica eficazmente com plataformas de monitorização centralizada. Por fim, as limitações de generalização observadas nas simulações de DDoS e *SQL Injection* sublinham que a viabilidade de um IDS completo depende de uma estratégia de pré-processamento que equilibre a simplificação dos dados com a preservação de informação semântica e comportamental.

CONCLUSÃO

A presente investigação permitiu validar o potencial da Inteligência Artificial na área da cibersegurança, cumprindo o objetivo central de avaliar modelos de *Machine Learning* e *Deep Learning* na detecção de anomalias em *logs* de rede. Através da análise comparativa, o modelo XGBoost demonstrou ser a solução mais robusta, alcançando um *Recall* de 0,96 e um equilíbrio superior entre a *Precision* e a taxa de detecção. Este resultado comprova que o motor de análise desenvolvido é altamente eficaz na triagem de ventos baseada em severidade, permitindo identificar riscos críticos com um número reduzido de falsos negativos.

No entanto, o processo de validação experimental, especificamente no Capítulo 2.5 (“Simulação de Cenários de Ataque e Análise de Desempenho”), revelou que a eficácia do sistema está intrinsecamente ligada à representação dos dados. A necessidade de suprimir variáveis contextuais por limitações de memória durante o treino impactou a capacidade do modelo em detetar ataques comportamentais como DDoS e *SQL Injection*. Esta descoberta não invalida o trabalho realizado, mas fornece um contributo científico importante ao demonstrar que um IDS inteligente exige um equilíbrio entre a eficiência computacional e a preservação do contexto dos eventos de rede.

Em suma, o trabalho culminou numa arquitetura conceptual que integra o motor de IA em ecossistemas SIEM, visando a automação da monitorização e a redução da carga de trabalho dos analistas. O trabalho aqui apresentado constitui, assim, um passo solido na modernização das defesas digitais da instituição, oferecendo uma base inteligente para a proteção de infraestruturas críticas contra ameaças contemporâneas.

TRABALHOS FUTUROS

A realização desta investigação permitiu identificar diversas oportunidades de expansão e melhoria, que podem servir de base para investigações futuras neste domínio. Como ponto de partida fundamental, sugere-se a transição da análise estática para uma implementação em tempo real. Isto envolveria a criação de uma infraestrutura física onde o motor de análise desenvolvido pudesse ser testado em ambiente de produção, permitindo avaliar o impacto de latência no processamento de eventos e a eficácia da comunicação via JSON com plataformas SIEM, como o *Elastic Stack* ou o *Splunk*.

Um desafio que se revela de grande importância abordar futuramente, prende-se com a gestão de recursos computacionais e a retenção de contexto. Dado que os erros de memória sentidos durante este trabalho forçaram a supressão de variáveis como endereços de IP e *timestamps*, investigações posteriores deverão focar-se na implementação de técnicas de processamento distribuído ou no uso de *hardware* com maior capacidade de memória. Esta evolução permitiria treinar modelos que não se limitem apenas à severidade intrínseca do *log*, mas que consigam correlacionar fluxos de rede, colmatando as lacunas de deteção identificadas no Capítulo 2.5.

Adicionalmente, propõe-se a exploração de novos paradigmas de aprendizagem, nomeadamente o *Incremental Learning*. Esta abordagem permitiria que o IDS se adaptasse continuamente a novas ameaças sem a necessidade de re-treinamentos completos do modelo, algo vital num panorama de cibersegurança onde surgem ataques *zero-day* com elevada frequência. Seria igualmente pertinente integrar mecanismos de *Explainable Artificial Intelligence* (XAI), para que os analistas de segurança possam compreender não apenas se um evento é perigoso, mas por que razão o modelo classificou desta forma, facilitando a triagem e a resposta a incidentes.

Por fim, sugere-se o alargamento do âmbito da solução através da integração de mecanismos de resposta automática (*C-box*). A evolução do motor de deteção para um IPS permitiria que, perante a classificação de um evento como crítico, o sistema pudesse interagir diretamente com a firewall ou outros ativos de rede para bloquear IPs maliciosos de modo autónomo, fechando o ciclo de segurança e promovendo uma infraestrutura de rede mais resiliente e inteligente.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] J. C. O. Pinto, “Sistema de detecção de intrusão em redes informáticas,” Instituto Superior de Engenharia do Porto, 2009. Accessed: Jan. 13, 2025. [Online]. Available: <https://recipp.ipp.pt/bitstreams/e42123e4-a4b9-4c79-bdb2-536223ee6cae/download>
- [2] C. França, “Sistema de detecção de ataque as redes de computadores.” Accessed: Jan. 13, 2025. [Online]. Available: <https://www.linkedin.com/pulse/sistema-de-detec%C3%A7%C3%A3o-ataque-redes-computadores-ca%C3%ADque-fran%C3%A7a/>
- [3] B. M. Leiner *et al.*, “A brief history of the internet,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 22–31, Oct. 2009, doi: 10.1145/1629607.
- [4] I. Figueiredo, “História das redes de computadores.” Accessed: Jan. 20, 2025. [Online]. Available: <https://www.oficinadanet.com.br/post/10123-historia-das-redes-de-computadores>
- [5] E. Monteiro and F. Boavida, *Engenharia de Redes Informáticas*, 7a Edição. FCA - Editora de Informática, 2005.
- [6] “Segurança das redes e da informação: Proposta de abordagem de uma política europeia,” *ANACOM*, pp. 1–24.
- [7] A. Zúquete, *Segurança em Redes Informáticas*, 1a Edição. FCA - Editora De Informática, 2007.
- [8] “O que é a Ameaça Interna?” Accessed: Feb. 03, 2025. [Online]. Available: <https://www.microsoft.com/pt-pt/security/business/security-101/what-is-insider-threat>
- [9] “O que é uma ameaça interna?” Accessed: Feb. 03, 2025. [Online]. Available: <https://www.fortinet.com/br/resources/cyberglossary/insider-threats>
- [10] “Ameaças externas, internas e risco digital: qual a diferença?” Accessed: Feb. 03, 2025. [Online]. Available: <https://blog.axur.com/pt-br/amea%C3%A7as-externas-internas-e-risco-digital-qual-a-diferen%C3%A7a>
- [11] A. Nadeem and M. P. Howarth, “A survey of manet intrusion detection & prevention approaches for network layer attacks,” *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 2027–2045, 2013, doi: 10.1109/SURV.2013.030713.00201.

- [12] H. Bhatt, “Cyber Security Attack Types - Active And Passive Attacks.” Accessed: Feb. 03, 2025. [Online]. Available: <https://www.encryptionconsulting.com/active-and-passive-attacks/>
- [13] “What Is an Eavesdropping Attack? Definition | Proofpoint UK.” Accessed: Feb. 04, 2025. [Online]. Available: <https://www.proofpoint.com/uk/threat-reference/eavesdropping>
- [14] “O que é um ataque DDoS?” Accessed: Feb. 04, 2025. [Online]. Available: <https://www.microsoft.com/pt-pt/security/business/security-101/what-is-a-ddos-attack>
- [15] M. Kosinki, “What Is Ransomware? | IBM.” Accessed: Feb. 04, 2025. [Online]. Available: <https://www.ibm.com/think/topics/ransomware>
- [16] “O que é um Cavalo de Tróia?” Accessed: Feb. 04, 2025. [Online]. Available: <https://www.checkpoint.com/pt/cyber-hub/cyber-security/what-is-trojan/>
- [17] C. Stouffer, “Spyware: What it is and how to protect yourself.” Accessed: Feb. 04, 2025. [Online]. Available: <https://us.norton.com/blog/malware/spyware>
- [18] “Como se proteger de golpes e ataques de phishing?” Accessed: Feb. 04, 2025. [Online]. Available: <https://www.kaspersky.com.br/resource-center/preemptive-safety/phishing-prevention-tips>
- [19] “What is MITM (Man in the Middle) Attack.” Accessed: Feb. 04, 2025. [Online]. Available: <https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/>
- [20] “What is SQL Injection? Tutorial & Examples | Web Security Academy.” Accessed: Feb. 04, 2025. [Online]. Available: <https://portswigger.net/web-security/sql-injection>
- [21] J. Maurício and S. Pinheiro, “Ameaças e Ataques aos Sistemas de Informação: Prevenir e Antecipar,” *Cadernos UniFOA*, vol. 3, no. 5, pp. 11–21, Mar. 2007, doi: 10.47385/CADUNIFOA.V3.N5.885.
- [22] R. Samrin and D. Vasumathi, “Review on anomaly based network intrusion detection system,” *International Conference on Electrical, Electronics, Communication Computer Technologies and Optimization Techniques, ICEECCOT 2017*, vol. 2018-January, pp. 141–147, Jul. 2017, doi: 10.1109/ICEECCOT.2017.8284655.
- [23] R. Grewal, “Architecture of NIDS.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.researchgate.net/figure/Architecture-of-NIDS_fig1_338458738
- [24] “O que é: Network Intrusion Detection System (NIDS).” Accessed: Feb. 14, 2025. [Online]. Available: <https://forense.io/glossario/o-que-e-network-intrusion-detection-system-nids/>
- [25] R. Borges, “Sistemas de Detecção e de Prevenção de Intrusos | mobicareofficial.” Accessed: Feb. 20, 2025. [Online]. Available:

<https://medium.com/mobicareofficial/sistemas-de-detec%C3%A7%C3%A3o-e-de-preven%C3%A7%C3%A3o-de-intrusos-princ%C3%ADpios-e-aplica%C3%A7%C3%A3o-de-entropia-9a33680f3948>

- [26] K. Scarfone and P. Mell, “Special Publication 800-94 Guide to Intrusion Detection and Prevention Systems (IDPS) Recommendations of the National Institute of Standards and Technology,” 2002, doi: 10.6028/NIST.SP.800-94.
- [27] I. Ghafir, V. Prenosil, J. Svoboda, and M. Hammoudeh, “A survey on network security monitoring systems,” *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016*, pp. 77–82, Oct. 2016, doi: 10.1109/W-FICLOUD.2016.30.
- [28] A. Tasneem, A. Kumar, and S. Sharma, “Intrusion Detection Prevention System using SNORT,” *Int. J. Comput. Appl.*, vol. 181, no. 32, pp. 21–24, Dec. 2018, doi: 10.5120/IJCA2018918280.
- [29] A. Hay, D. Cid, and R. Bray, “OSSEC Host-Based Intrusion Detection Guide”, Accessed: Jun. 06, 2025. [Online]. Available: www.sans.edu
- [30] A. Pal Singh and M. Deep Singh, “Analysis of Host-Based and Network-Based Intrusion Detection System,” *International Journal of Computer Network and Information Security*, vol. 6, no. 8, pp. 41–47, Jul. 2014, doi: 10.5815/IJCNIS.2014.08.06.
- [31] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, “Deep learning for anomaly detection in log data: A survey,” *Machine Learning with Applications*, vol. 12, p. 100470, Jun. 2023, doi: 10.1016/J.MLWA.2023.100470.
- [32] “Event Log Monitoring Tool - A Tutorial.” Accessed: May 06, 2025. [Online]. Available: https://www.manageengine.com/network-monitoring/Eventlog_Tutorial_Part_I.html
- [33] L. Girardin and D. Brodbeck, “A Visual Approach for Monitoring Logs.” Accessed: Apr. 02, 2025. [Online]. Available: https://www.usenix.org/legacy/event/lisa98/full_papers/girardin/girardin_html/girardin.html
- [34] M. Astekin, H. Zengin, and H. Sözer, “Evaluation of Distributed Machine Learning Algorithms for Anomaly Detection from Large-Scale System Logs: A Case Study,” *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, pp. 2071–2077, Jul. 2018, doi: 10.1109/BIGDATA.2018.8621967.
- [35] Y. Y. Aung and M. M. Min, “An analysis of random forest algorithm based network intrusion detection system,” *Proceedings - 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2017*, pp. 127–132, Aug. 2017, doi: 10.1109/SNPD.2017.8022711.
- [36] L. Podlowski and M. Kozłowski, “Application of XGBoost to the cyber-security problem of detecting suspicious network traffic events,” *Proceedings -*

- 2019 IEEE International Conference on Big Data, *Big Data 2019*, pp. 5902–5907, Dec. 2019, doi: 10.1109/BIGDATA47090.2019.9006586.
- [37] S. Yen and M. Moh, “Intelligent Log Analysis Using Machine and Deep Learning,” <https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-5225-8100-0.ch007>, pp. 154–189, 2019, doi: 10.4018/978-1-5225-8100-0.CH007.
 - [38] P. Castro, F. Santos, and P. Lopes, “Artificial intelligence models for log event analysis,” *Millenium - Journal of Education, Technologies, and Health*, vol. 2025, no. 20e, pp. e41569–e41569, Oct. 2025, doi: 10.29352/MILL0220E.41569.
 - [39] “Elastic — The Search AI Company | Elastic.” Accessed: May 28, 2025. [Online]. Available: <https://www.elastic.co/>
 - [40] P. Shukla and S. Kumar, *Learning Elastic Stack 7.0: Distributed search, analytics, and visualization using Elasticsearch, Logstash, Beats and Kibana*, 2a. Packt, 2019. Accessed: May 27, 2025. [Online]. Available: <https://books.google.com.br/books?hl=pt-PT&lr=&id=AzqbDwAAQBAJ&oi=fnd&pg=PP1&dq=~+Learning+Elastic+Stack+7.0.+Packt+Publishing.&ots=5r9DMdaGjh&sig=uMKpcztuOHuX1ODgV2HKAzWekVY#v=onepage&q=~%20Learning%20Elastic%20Stack%207.0.%20Packt%20Publishing.&f=false>
 - [41] “What Is Splunk & What Does It Do? A Splunk Intro | Splunk.” Accessed: May 28, 2025. [Online]. Available: https://www.splunk.com/en_us/blog/learn/what-splunk-does.html
 - [42] M. Hristov, M. Nenova, G. Iliev, and D. Avresky, “Integration of Splunk Enterprise SIEM for DDoS Attack Detection in IoT,” *2021 IEEE 20th International Symposium on Network Computing and Applications, NCA 2021*, 2021, doi: 10.1109/NCA53618.2021.9685977.
 - [43] “Graylog Documentation.” Accessed: May 28, 2025. [Online]. Available: <https://go2docs.graylog.org/current/home.htm>
 - [44] R.-D. Şuştic, A. Moraru, A.-B. Rus, and V. Dobrota, “PERFORMANCE EVALUATION OF ELK STACK VERSUS GRAYLOG AS OPEN-SOURCE LOG MANAGEMENT TOOLS,” vol. 62, no. 1, p. 2022, 2022.
 - [45] G. González-Granadillo, S. González-Zarzosa, and R. Diaz, “Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures,” *Sensors 2021, Vol. 21, Page 4759*, vol. 21, no. 14, p. 4759, Jul. 2021, doi: 10.3390/S21144759.
 - [46] P. Radoglou-Grammatikis *et al.*, “SPEAR SIEM: A Security Information and Event Management system for the Smart Grid,” *Computer Networks*, vol. 193, p. 108008, Jul. 2021, doi: 10.1016/J.COMNET.2021.108008.
 - [47] M. Cinque, D. Cotroneo, and A. Pecchia, “Challenges and Directions in Security Information and Event Management (SIEM),” *Proceedings - 29th IEEE*

International Symposium on Software Reliability Engineering Workshops, ISSREW 2018, pp. 95–99, Nov. 2018, doi: 10.1109/ISSREW.2018.00-24.

- [48] S. Bhatt, P. K. Manadhata, and L. Zomlot, “The operational role of security information and event management systems,” *IEEE Secur. Priv.*, vol. 12, no. 5, pp. 35–41, 2014, doi: 10.1109/MSP.2014.103.
- [49] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1285–1298, Oct. 2017, doi: 10.1145/3133956.3134015/SUPPL_FILE/MINDU-DEEPLOGANOMALY.MP4.
- [50] I. Belcic and C. Stryker, “What Is Supervised Learning? | IBM.” Accessed: Apr. 04, 2025. [Online]. Available: <https://www.ibm.com/think/topics/supervised-learning>
- [51] M. A. El Mrabet, K. El Makkaoui, and A. Faize, “Supervised Machine Learning: A Survey,” in *2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)*, IEEE, Dec. 2021, pp. 1–10. doi: 10.1109/CommNet52204.2021.9641998.
- [52] P. A. A. Resende and A. C. Drummond, “A Survey of Random Forest Based Methods for Intrusion Detection Systems,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, May 2018, doi: 10.1145/3178582.
- [53] D. Amaratunga, J. Cabrera, and Y. S. Lee, “Enriched random forests,” *Bioinformatics*, vol. 24, no. 18, pp. 2010–2014, Sep. 2008, doi: 10.1093/BIOINFORMATICS/BTN356.
- [54] J. Abellán, C. J. Mantas, and J. G. Castellano, “A Random Forest approach using imprecise probabilities,” *Knowl. Based. Syst.*, vol. 134, pp. 72–84, Oct. 2017, doi: 10.1016/J.KNOSYS.2017.07.019.
- [55] W. Feng, C. Ma, G. Zhao, and R. Zhang, “FSRF: An Improved Random Forest for Classification,” *Proceedings of 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications, AEECA 2020*, pp. 173–178, Aug. 2020, doi: 10.1109/AEECA49918.2020.9213456.
- [56] M. Himalaya Gohiya, M. Harsh Lohiya, and M. Kailash Patidar, “A SURVEY OF XGBOOST SYSTEM,” *International Journal of Advanced Technology & Engineering Research (IJATER)*, vol. 8, no. 3, 2018, Accessed: Apr. 06, 2025. [Online]. Available: www.ijater.com
- [57] G. Abdiyeva-Aliyeva, J. Aliyev, and U. Sadigov, “Application of classification algorithms of Machine learning in cybersecurity,” *Procedia Comput. Sci.*, vol. 215, pp. 909–919, Jan. 2022, doi: 10.1016/J.PROCS.2022.12.093.
- [58] T. L. Marinho and <http://lattes.cnpq.br/5348870751897882>, “Otimização de hiperparâmetros do XGBoost utilizando meta-aprendizagem,” Dec. 2021, Accessed: Apr. 06, 2025. [Online]. Available: <http://www.repositorio.ufal.br/jspui/handle/123456789/9851>

- [59] M. Somvanshi, P. Chavan, S. Tambade, and S. V. Shinde, "A review of machine learning techniques using decision tree and support vector machine," *Proceedings - 2nd International Conference on Computing, Communication, Control and Automation, ICCUBEA 2016*, Feb. 2017, doi: 10.1109/ICCUBEA.2016.7860040.
- [60] W. : Www and A. Pradhan, "International Journal of Emerging Technology and Advanced Engineering SUPPORT VECTOR MACHINE-A Survey," vol. 2, no. 8, 2012, Accessed: Apr. 19, 2025. [Online]. Available: www.ijetae.com
- [61] H. Liu and B. Lang, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey," *Applied Sciences 2019, Vol. 9, Page 4396*, vol. 9, no. 20, p. 4396, Oct. 2019, doi: 10.3390/APP9204396.
- [62] Bernhard. Schölkopf and A. J. . Smola, "Learning with kernels : support vector machines, regularization, optimization, and beyond," p. 626, 2002.
- [63] P. Singh and P. A. Meshram, "Survey of density based clustering algorithms and its variants," *Proceedings of the International Conference on Inventive Computing and Informatics, ICICI 2017*, pp. 920–926, May 2018, doi: 10.1109/ICICI.2017.8365272.
- [64] H. V. Singh, A. Girdhar, and S. Dahiya, "A Literature survey based on DBSCAN algorithms," *Proceedings - 2022 6th International Conference on Intelligent Computing and Control Systems, ICICCS 2022*, pp. 751–758, 2022, doi: 10.1109/ICICCS53718.2022.9788440.
- [65] O. Kulkarni and A. Burhanpurwala, "A Survey of Advancements in DBSCAN Clustering Algorithms for Big Data," *2024 3rd International Conference on Power Electronics and IoT Applications in Renewable Energy and its Control, PARC 2024*, pp. 106–111, 2024, doi: 10.1109/PARC59193.2024.10486339.
- [66] M. Parilama, D. Lopez, and N. C. Senthilkumar, "(PDF) A Survey on Density Based Clustering Algorithms for Mining Large Spatial Databases," *International Journal of Advanced Science and Technology, June, 2011*, vol. Vol 31, Jun. 2011, Accessed: Apr. 09, 2025. [Online]. Available: https://www.researchgate.net/publication/266490040_A_Survey_on_Density_Based_Clustering_Algorithms_for_Mining_Large_Spatial_Databases
- [67] S. Na, L. Xumin, and G. Yong, "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm," in *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, IEEE, Apr. 2010, pp. 63–67. doi: 10.1109/IITSI.2010.74.
- [68] Y. Li and H. Wu, "A Clustering Method Based on K-Means Algorithm," *Phys. Procedia*, vol. 25, pp. 1104–1109, Jan. 2012, doi: 10.1016/J.PHPRO.2012.03.206.
- [69] R. C. de Amorim, "A Survey on Feature Weighting Based K-Means Algorithms," *J. Classif.*, vol. 33, no. 2, pp. 210–242, Jul. 2016, doi: 10.1007/S00357-016-9208-4/METRICS.

- [70] M. Ahmed, R. Seraj, and S. M. S. Islam, “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation,” *Electronics (Basel)*., vol. 9, no. 8, p. 1295, Aug. 2020, doi: 10.3390/electronics9081295.
- [71] W.-L. Zhao, C.-H. Deng, and C.-W. Ngo, “k -means: A revisit,” *Neurocomputing*, vol. 291, pp. 195–206, May 2018, doi: 10.1016/j.neucom.2018.02.072.
- [72] Y. Chabchoub, M. U. Togbe, A. Boly, and R. Chiky, “An In-Depth Study and Improvement of Isolation Forest,” *IEEE Access*, vol. 10, pp. 10219–10237, 2022, doi: 10.1109/ACCESS.2022.3144425.
- [73] W. S. Al Farizi, I. Hidayah, and M. N. Rizal, “Isolation Forest Based Anomaly Detection: A Systematic Literature Review,” *2021 8th International Conference on Information Technology, Computer and Electrical Engineering, ICITACEE 2021*, pp. 118–122, 2021, doi: 10.1109/ICITACEE53184.2021.9617498.
- [74] H. Xu, G. Pang, Y. Wang, and Y. Wang, “Deep Isolation Forest for Anomaly Detection,” *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12591–12604, Dec. 2023, doi: 10.1109/TKDE.2023.3270293.
- [75] D. Xu, Y. Wang, Y. Meng, and Z. Zhang, “An improved data anomaly detection method based on isolation forest,” *Proceedings - 2017 10th International Symposium on Computational Intelligence and Design, ISCID 2017*, vol. 2, pp. 287–291, Feb. 2018, doi: 10.1109/ISCID.2017.202.
- [76] “Supervised, unsupervised and deep learning | by Claudio Masolo | TDS Archive | Medium.” Accessed: Apr. 19, 2025. [Online]. Available: <https://medium.com/data-science/supervised-unsupervised-and-deep-learning-aa61a0e5471c>
- [77] J. Holdsworth and M. Scapicchio, “What Is Deep Learning? | IBM.” Accessed: Apr. 19, 2025. [Online]. Available: <https://www.ibm.com/think/topics/deep-learning>
- [78] G. Van Houdt, C. Mosquera, and G. Nápoles, “A review on the long short-term memory model,” *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5929–5955, Dec. 2020, doi: 10.1007/S10462-020-09838-1/TABLES/1.
- [79] A. Graves, “Long Short-Term Memory,” pp. 37–45, 2012, doi: 10.1007/978-3-642-24797-2_4.
- [80] H. Chung and K. S. Shin, “Genetic Algorithm-Optimized Long Short-Term Memory Network for Stock Market Prediction,” *Sustainability 2018, Vol. 10, Page 3765*, vol. 10, no. 10, p. 3765, Oct. 2018, doi: 10.3390/SU10103765.
- [81] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, Jun. 2014, doi: 10.3115/v1/d14-1179.
- [82] T. R. de Mello, “Comparativo entre redes neurais recorrentes GRU e LSTM para a predição de instrumentos financeiros,” 2021.

- [83] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/NECO.1997.9.8.1735.
- [84] C. Hashemi-Pour and B. Lutkevich, "What is the BERT language model? | Definition from TechTarget." Accessed: Apr. 15, 2025. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model>
- [85] H. Wang, J. Li, and Z. Li, "AI-Generated Text Detection and Classification Based on BERT Deep Learning Algorithm," May 2024, Accessed: Apr. 19, 2025. [Online]. Available: <https://arxiv.org/abs/2405.16422v1>
- [86] Y. Zhang, J. Lin, L. Zhao, X. Zeng, and X. Liu, "A novel antibacterial peptide recognition algorithm based on BERT," *Brief. Bioinform.*, vol. 22, no. 6, pp. 1–11, Nov. 2021, doi: 10.1093/BIB/BBAB200.
- [87] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2021-July, Jul. 2021, doi: 10.1109/IJCNN52387.2021.9534113.
- [88] A. Asperti, D. Evangelista, and E. Loli Piccolomini, "A Survey on Variational Autoencoders from a Green AI Perspective," *SN Comput. Sci.*, vol. 2, no. 4, pp. 1–23, Jul. 2021, doi: 10.1007/S42979-021-00702-9/FIGURES/20.
- [89] D. Bergmann and C. Stryker, "What is a Variational Autoencoder? | IBM." Accessed: Apr. 20, 2025. [Online]. Available: <https://www.ibm.com/think/topics/variational-autoencoder>
- [90] "Variational AutoEncoders | GeeksforGeeks." Accessed: Apr. 20, 2025. [Online]. Available: <https://www.geeksforgeeks.org/variational-autoencoders/>
- [91] R. Zemouri *et al.*, "Recent Research and Applications in Variational Autoencoders for Industrial Prognosis and Health Management: A Survey," *Proceedings - 2022 Prognostics and Health Management Conference, PHM-London 2022*, pp. 193–203, 2022, doi: 10.1109/PHM2022-LONDON52454.2022.00042.
- [92] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1285–1298, Oct. 2017, doi: 10.1145/3133956.3134015/SUPPL_FILE/MINDU-DEEPLOGANOMALY.MP4.
- [93] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, Jan. 2021, doi: 10.1002/ETT.4150.
- [94] G. Pintado, "Confusion Matrix. The Confusion Matrix is a predicted... | by Gloria Pintado | Medium." Accessed: Oct. 31, 2025. [Online]. Available: <https://medium.com/@karmely.1999/confusion-matrix-f7f8b88b2ae8>

- [95] E. Altulaihan, M. A. Almaiah, and A. Aljughaiman, "Anomaly Detection IDS for Detecting DoS Attacks in IoT Networks Based on Machine Learning Algorithms," *Sensors* 2024, Vol. 24, Page 713, vol. 24, no. 2, p. 713, Jan. 2024, doi: 10.3390/S24020713.
- [96] H. Liu and B. Lang, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey," *Applied Sciences* 2019, Vol. 9, Page 4396, vol. 9, no. 20, p. 4396, Oct. 2019, doi: 10.3390/APP9204396.
- [97] J. L. Gutierrez-Garcia, E. Sanchez-DelaCruz, and M. del P. Pozos-Parra, "A Review of Intrusion Detection Systems Using Machine Learning: Attacks, Algorithms and Challenges," *Lecture Notes in Networks and Systems*, vol. 652 LNNS, pp. 59–78, 2023, doi: 10.1007/978-3-031-28073-3_5.
- [98] "Evolution and advancements in intrusion detection systems: from traditional methods to deep learning and federated learning approaches," *ACCENTS Transactions on Information Security*, vol. 9, no. 36, Jul. 2024, doi: 10.19101/TIS.2024.935002.
- [99] A. H. Ali *et al.*, "Unveiling machine learning strategies and considerations in intrusion detection systems: a comprehensive survey," *Front. Comput. Sci.*, vol. 6, p. 1387354, Jun. 2024, doi: 10.3389/FCOMP.2024.1387354/FULL.
- [100] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100470, Jun. 2023, doi: 10.1016/J.MLWA.2023.100470.
- [101] V. Sharma and M. Kumar, "Comparative Analysis of Machine Learning Models for Intrusion Detection Systems," *Panamerican Mathematical Journal*, vol. 35, no. 3s, pp. 273–285, Jan. 2025, doi: 10.52783/PMJ.V35.I3S.3891.
- [102] Z. Fan and Z. You, "Research on network intrusion detection based on XGBoost algorithm and multiple machine learning algorithms," 2024, doi: 10.54254/2753-8818/31/20241171.
- [103] Vikrant Sharma, "Comparative Analysis of Machine Learning Models for Intrusion Detection Systems," *Panamerican Mathematical Journal*, vol. 35, no. 3s, pp. 273–285, Jan. 2025, doi: 10.52783/PMJ.V35.I3S.3891.
- [104] S. S. Dhaliwal, A. Al Nahid, and R. Abbas, "Effective Intrusion Detection System Using XGBoost," *Information* 2018, Vol. 9, Page 149, vol. 9, no. 7, p. 149, Jun. 2018, doi: 10.3390/INFO9070149.
- [105] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, "A survey on anomaly detection for technical systems using LSTM networks," *Comput. Ind.*, vol. 131, p. 103498, Oct. 2021, doi: 10.1016/J.COMPIND.2021.103498.
- [106] "What is LSTM - Long Short Term Memory? - GeeksforGeeks." Accessed: Nov. 10, 2025. [Online]. Available: <https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/>

- [107] A. J. A. Immastephy and K. Punitha, "A Systematic Review on Network Intrusion Detection System based on machine learning and deep learning approach," *E3S Web of Conferences*, vol. 540, p. 14006, Jun. 2024, doi: 10.1051/E3SCONF/202454014006.
- [108] L. Thomas, B. Iv, and A. Vakanski, "A Comparative Analysis of Machine Learning and Deep Learning Approaches for Network Intrusion Detection".
- [109] A. Alaref, "Hash Encoding(Or Feature Hashing)." Accessed: Oct. 20, 2025. [Online]. Available: <https://www.kaggle.com/code/adnanalaref/hash-encoding-or-feature-hashing>

APÊNDICE A

Preparação dos Dados (Pipeline)

```
import pandas as pd
import os
from IPython.display import display
from category_encoders import HashingEncoder
```

0. Funções Auxiliares

```
'''Nota: Foi realizado um ajuste no threshold para 0.5% para ser mais agressivo no agrupamento de categorias raras,
na tentativa de evitar MemoryError e reduzir o número de colunas.'''

def group_rare_categories(df, column_name, threshold=0.005):
    if column_name not in df.columns or df[column_name].dtype != 'object':
        print(f"Aviso: Coluna '{column_name}' não encontrada ou não é do tipo 'object'.")
        return df

    print(f"Agrupando categorias raras na coluna '{column_name}' (limiar: {threshold*100:.3f}%)...")
    value_counts = df[column_name].value_counts(normalize=True)
    rare_categories = value_counts[value_counts < threshold].index.tolist()

    if rare_categories:
        df[column_name] = df[column_name].replace(rare_categories, 'Other_Rare_Category')
        print(f" - {len(rare_categories)} categorias raras agrupadas em 'Other_Rare_Category' em '{column_name}'.")
        print(f" - Novas contagens de valores únicos em '{column_name}':")
        display(df[column_name].value_counts().head(10))
    else:
        print(f" - Nenhuma categoria rara encontrada em '{column_name}' abaixo do limiar.")
    return df
```

1. Lista de Ficheiros

```
: print("--- 1.: Listagem de Ficheiros ---")

folder_path = "."

# --- Lista os nomes de todos os ficheiros CSV ---
all_files = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if f.endswith('.csv')]

if not all_files:
    print(f"ATENÇÃO: Não foram encontrados ficheiros CSV na pasta '{os.path.abspath(folder_path)}'.")
    print("Verifique se os ficheiros estão na pasta correta ou ajuste o 'folder_path'.")
    raise FileNotFoundError("Nenhum ficheiro CSV encontrado. Verifique o caminho e a presença dos ficheiros.")
else:
    print(f"Ficheiros CSV encontrados na pasta '{os.path.abspath(folder_path)}':")
    for idx, f in enumerate(all_files):
        print(f"[{idx}] - {f}") # Mostra um índice para cada ficheiro

# (As variáveis 'all_files' e 'folder_path' ficarão na memória)
```

2. Análise Individual dos Ficheiros

```
file_index_to_analyze = 0 # Começa com 0 para o primeiro ficheiro e assim sucessivamente

pd.set_option('display.max_rows', None)

if file_index_to_analyze < len(all_files):
    file_path = all_files[file_index_to_analyze]
    file_name = os.path.basename(file_path)

    print(f"#####")
    print(f"## ANALISANDO FICHEIRO: {file_name}")
    print(f"#####")

    try:
        # Carregar o ficheiro CSV com low_memory=False para evitar problemas de tipos mistos
        df_individual = pd.read_csv(file_path, low_memory=False)

        print(f"\n --> Informações Gerais para '{file_name}':")
        print(f"    Número de linhas: {len(df_individual)}")
        print(f"    Número de colunas: {len(df_individual.columns)}")
        print(f"    Formato (linhas, colunas): {df_individual.shape}")

        print(f"\n --> Primeiras 10 linhas de '{file_name}':")
        display(df_individual.head(10))

        print(f"\n --> Resumo de Tipos de Dados e Não-Nulos para '{file_name}':")
        df_individual.info()

        # Contagem de NaN's por coluna de forma individual para este ficheiro
        nan_individual = df_individual.isnull().sum()
        nan_individual = nan_individual[nan_individual > 0].sort_values(ascending=False) # Ordenar por mais NaNs para facilitar a visualização

        if not nan_individual.empty:
            print(f"\n --> Contagem de NaN's em '{file_name}' concluída:")
            display(nan_individual.to_frame(name='NaN Count')) # Exibe como um DataFrame de coluna única
        else:
            print(f"\n --> Não há valores NaN em '{file_name}'.")

        print("\n" + "="*80) # Separador visual entre ficheiros

    except Exception as e:
        print(f"ERRO: Não foi possível carregar ou analisar o ficheiro {file_name}.")
        print(f"Detalhes do erro: {e}")
        print("Verifique se o ficheiro está acessível.")
else:
    print(f"Erro: O índice {file_index_to_analyze} está fora do alcance da lista de ficheiros ({len(all_files)} ficheiros encontrados).")
```

3. Carregamento de Todos os Ficheiros e Identificação de Colunas

```
print("--- 3: Carregamento de Todos os Ficheiros e Identificação de Colunas ---")

dataframes_raw = {} # Dicionário para guardar cada DataFrame original
all_columns_sets = [] # Lista para guardar os conjuntos de colunas de cada ficheiro

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_rows', 100)

print("\n--- A carregar ficheiros ---")
for file_path in all_files:
    file_name = os.path.basename(file_path)
    print(f"\nA carregar ficheiro: {file_name}")
    try:
        temp_df = pd.read_csv(file_path, low_memory=False)
        dataframes_raw[file_name] = temp_df # <-- ESTA LINHA VAI GUARDAR OS DATAFRAMES!
        all_columns_sets.append(set(temp_df.columns)) # Guarda o conjunto de colunas

        print(f" - Linhas: {len(temp_df)}, Colunas: {len(temp_df.columns)}")

    except Exception as e:
        print(f"Erro ao carregar o ficheiro {file_name}: {e}")
        print("Este ficheiro será ignorado.")

if not dataframes_raw:
    print("\nErro crítico: Nenhum DataFrame foi carregado com sucesso. Verifique a integridade dos CSVs ou o 'folder_path'.")
    raise ValueError("Falha ao carregar DataFrames. Nenhum dado disponível.")

# 1.4 Identificar Colunas Comuns e Incomuns
if len(all_columns_sets) > 1:
    common_columns = set.intersection(*all_columns_sets)
    all_unique_columns = set.union(*all_columns_sets)
    uncommon_columns = all_unique_columns - common_columns

    print("\n--- RESUMO DE COLUNAS: Comuns vs. Incomuns em Todos os Ficheiros ---")
    print(f"Número de colunas comuns a TODOS os ficheiros: {len(common_columns)}")
    print("Colunas Comuns:")
    print(sorted(list(common_columns)))

    print(f"Número de colunas incomuns (não estão em todos os ficheiros): {len(uncommon_columns)}")
    print("Colunas Incomuns:")
    print(sorted(list(uncommon_columns)))
elif len(all_columns_sets) == 1:
    print("\n--- RESUMO DE COLUNAS: Apenas Um Ficheiro ---")
    print("Apenas um ficheiro CSV foi encontrado. Não há colunas incomuns a comparar.")
    print("Colunas do ficheiro:", sorted(list(all_columns_sets[0])))
else:
    print("\nErro: Nenhum DataFrame foi carregado com sucesso para análise de colunas.")
```

3.1. Percentagem de NaNs nas Colunas Incomuns

```
# Verificação de segurança:
if 'dataframes_raw' not in locals() or not dataframes_raw:
    print("Erro: 'dataframes_raw' não está definido ou está vazio. Por favor, execute a Célula 2 primeiro.")
elif 'uncommon_columns' not in locals():
    print("Erro: 'uncommon_columns' não está definido. Por favor, execute a Célula 2 para identificar as colunas comuns/incomuns.")
else:
    print("\n--- Análise de NaN nas Colunas Incomuns por Ficheiro ---")

    # Ajustar opções de display para esta saída específica (garantir que todas as linhas aparecem)
    pd.set_option('display.max_rows', None)
    pd.set_option('display.width', None)

    for file_name, df_data in dataframes_raw.items():
        print(f"\n### Ficheiro: {file_name} ###")

        # Obter colunas incomuns que estão realmente presentes neste ficheiro específico
        uncommon_cols_in_this_file = [col for col in uncommon_columns if col in df_data.columns]

        if uncommon_cols_in_this_file:
            # Calcular NaNs para apenas as colunas incomuns deste ficheiro
            nan_counts_in_uncommon = df_data[uncommon_cols_in_this_file].isnull().sum()
            nan_counts_in_uncommon = nan_counts_in_uncommon[nan_counts_in_uncommon > 0].sort_values(ascending=False)

            if not nan_counts_in_uncommon.empty:
                total_rows_in_file = len(df_data)
                nan_percentages_in_uncommon = (nan_counts_in_uncommon / total_rows_in_file) * 100

                # Combinar contagens e percentagens num DataFrame para exibição
                summary_df = pd.DataFrame({
                    'NaN Count': nan_counts_in_uncommon,
                    '% NaN': nan_percentages_in_uncommon.map('{:.2f}%'.format) # Formatar percentagem
                })
                display(summary_df)
            else:
                print("Nenhuma coluna incomum presente neste ficheiro tem valores NaN.")
```

3.2. Percentagem de NaNs nas Colunas Comuns

```
# Verificação de segurança:
if 'dataframes_raw' not in locals() or not dataframes_raw:
    print("Erro: 'dataframes_raw' não está definido ou está vazio.")
elif 'common_columns' not in locals():
    print("Erro: 'common_columns' não está definido.")
else:
    print("\n--- Análise de NaN nas Colunas Comuns por Ficheiro ---")

    # Ajustar opções de display para esta saída específica (garantir que todas as linhas aparecem)
    pd.set_option('display.max_rows', None)
    pd.set_option('display.width', None)

    for file_name, df_data in dataframes_raw.items():
        print(f"\n### Ficheiro: {file_name} ###")

        # Obter colunas comuns que estão realmente presentes neste ficheiro específico
        common_cols_in_this_file = [col for col in common_columns if col in df_data.columns]

        if common_cols_in_this_file:
            # Calcular NaNs para apenas as colunas comuns deste ficheiro
            nan_counts_in_common = df_data[common_cols_in_this_file].isnull().sum()
            nan_counts_in_common = nan_counts_in_common[nan_counts_in_common > 0].sort_values(ascending=False)

            if not nan_counts_in_common.empty:
                total_rows_in_file = len(df_data)
                nan_percentages_in_common = (nan_counts_in_common / total_rows_in_file) * 100

                # Combinar contagens e percentagens num DataFrame para exibição
                summary_df = pd.DataFrame({
                    'NaN Count': nan_counts_in_common,
                    '% NaN': nan_percentages_in_common.map('{:.2f}%'.format) # Formatar percentagem
                })
                display(summary_df)
            else:
                print("Nenhuma coluna comum presente neste ficheiro tem valores NaN.")
        else:
            print("Nenhuma coluna comum está presente neste ficheiro.")
```

4. Pré-processamento

4.1. Combinação Inicial e Limpeza Básica

```
print("\n--- 4.1: Combinação Inicial e Limpeza Básica (NaNs Otimizado e Protegido) ---")

# 2.1 Definir Colunas Desejadas (Comuns + Incomuns Seleccionadas)
if 'all_columns_sets' not in locals() or not all_columns_sets:
    print("Erro: As variáveis não estão definidas ou estão vazias.")
    raise ValueError("Variáveis ausentes. Execute a célula 1 primeiro.")
else:
    common_columns = set.intersection(*all_columns_sets) # Re-obter common_columns

    selected_uncommon_columns = [
        'Machine Name',
        # Manter 'Machine Name' pois tinha 64.09% NaNs (abaixo do limiar de 70%).
    ]

    all_desired_columns = list(common_columns.union(set(selected_uncommon_columns)))

    # 2.2 Adicionar 'Action' e 'Severity' à lista de colunas desejadas se ainda não estiverem
    if 'Action' not in all_desired_columns:
        all_desired_columns.append('Action')
    if 'Severity' not in all_desired_columns: # Adicionar Severity também, para futura target
        all_desired_columns.append('Severity')

    print(f"\nNúmero total de colunas a manter (comuns + seleccionadas antes da limpeza final): {len(all_desired_columns)}")

    # 2.3 Filtrar DataFrames e Concatenar
    list_of_dfs_filtered = []
    for df_name, df_data in dataframes_raw.items():
        cols_to_keep_in_this_df = [col for col in all_desired_columns if col in df_data.columns]
        list_of_dfs_filtered.append(df_data[cols_to_keep_in_this_df])

    df = pd.concat(list_of_dfs_filtered, ignore_index=True)
    print(f"DataFrame inicial combinado. Linhas: {df.shape[0]}, Colunas: {df.shape[1]}")

# --- Início da Limpeza e Pré-processamento ---

print("\n--- Início da Limpeza de NaNs e Pré-processamento ---")

# 2.4 Limpar nomes de colunas (remover caractere especial BOM)
df.columns = df.columns.str.replace('\u00ff', '', regex=False)
print("Nomes de colunas limpos.")

# 2.5 Remover Colunas com Alta Percentagem de NaNs (Única Passagem: >= 70%)
nan_threshold_for_removal = 70 # <-- LIMiar de REMOÇÃO DE NaN AJUSTADO PARA 70%
nan_percentages_to_check = (df.isnull().sum() / len(df)) * 100

# LISTA DE COLUNAS A PROTEGER DA REMOÇÃO POR NaN
cols_to_protect_from_nan_drop = ['Action', 'Severity'] # Proteger as colunas target

# Identificar colunas a remover, excluindo as protegidas
cols_to_drop_final_nan_removal = [
    col for col in nan_percentages_to_check[nan_percentages_to_check >= nan_threshold_for_removal].index.tolist()
    if col not in cols_to_protect_from_nan_drop
]

if cols_to_drop_final_nan_removal:
    df.drop(columns=cols_to_drop_final_nan_removal, inplace=True)
    print(f"Colunas removidas (>= {nan_threshold_for_removal}% NaN, exceto protegidas): {len(cols_to_drop_final_nan_removal)}. Colunas atuais: {df.shape[1]}")
else:
    print(f"Nenhuma coluna removida nesta passagem (>= {nan_threshold_for_removal}% NaN).")
```

4.2. Processamento de Datas/Tempos e Elapsed

```
print("\n--- 4.2: Processamento de Datas/Tempos e Elapsed ---")

# 2.7 Processar Colunas de Data/Tempo
date_time_columns = ['Time', 'Policy Date', 'start_time', 'lastupdatetime', 'segment_time']
for col in date_time_columns:
    if col in df.columns:
        try:
            df[col] = pd.to_datetime(df[col], errors='coerce', utc=True)
            df[f'{col}_Year'] = df[col].dt.year
            df[f'{col}_Month'] = df[col].dt.month
            df[f'{col}_DayOfWeek'] = df[col].dt.dayofweek
            df[f'{col}_Hour'] = df[col].dt.hour
            df[f'{col}_Minute'] = df[col].dt.minute
            df[f'{col}_Second'] = df[col].dt.second
            df[f'{col}_Epoch'] = df[col].astype('int64') // 10**9
            for new_col in [f'{col}_Year', f'{col}_Month', f'{col}_DayOfWeek', f'{col}_Hour', f'{col}_Minute', f'{col}_Second', f'{col}_Epoch']:
                if df[new_col].isnull().any(): df[new_col] = df[new_col].fillna(df[new_col].median()) # Corrigido fillna
            df.drop(columns=[col], inplace=True)
            print(f"Coluna '{col}': características de data/tempo extraídas e original removida. Colunas atuais: {df.shape[1]}")
        except Exception as e:
            print(f"ERRO ao processar '{col}': {e}. Removendo.")
            if col in df.columns: df.drop(columns=[col], inplace=True)
    else:
        print(f"Coluna '{col}' (data/tempo) não encontrada.")

# 2.8 Processar Coluna 'elapsed' (converter para segundos)
if 'elapsed' in df.columns:
    def time_to_seconds(time_str):
        if pd.isna(time_str) or not isinstance(time_str, str): return 0
        parts = str(time_str).split(':')
        if len(parts) == 3: h, m, s = int(parts[0]), int(parts[1]), int(parts[2]); return h * 3600 + m * 60 + s
        return 0
    df['elapsed_seconds'] = df['elapsed'].apply(time_to_seconds)
    if df['elapsed_seconds'].isnull().any(): df['elapsed_seconds'] = df['elapsed_seconds'].fillna(df['elapsed_seconds'].median()) # Corrigido fillna
    df.drop(columns=['elapsed'], inplace=True)
    print(f"Coluna 'elapsed': convertida para 'elapsed_seconds' e original removida. Colunas atuais: {df.shape[1]}")
else:
    print("Coluna 'elapsed' não encontrada. Ignorado.")
```

4.3. Preenchimento Final de NaNs e Agrupamento de Categorias

```
print("\n--- 4.3: Preenchimento Final de NaNs e Agrupamento de Categorias ---")

# 2.9 Preencher NaN's restantes (após todas as remoções e processamentos)
nan_percentages_final_fill_check = (df.isnull().sum() / len(df)) * 100
columns_to_fill_final = nan_percentages_final_fill_check[nan_percentages_final_fill_check > 0].index.tolist()
if columns_to_fill_final:
    print("\nPreenchendo NaN's restantes nas colunas:")
    for col in columns_to_fill_final:
        if df[col].dtype == 'object':
            mode_val = df[col].mode()[0]
            df[col] = df[col].fillna(mode_val) # Corrigido fillna
            print(f"- Coluna '{col}' (object): preenchida com moda ({mode_val})")
        elif df[col].dtype in ['float64', 'int64', 'int32', 'bool']:
            median_val = df[col].median()
            df[col] = df[col].fillna(median_val) # Corrigido fillna
            print(f"- Coluna '{col}' (numérica): preenchida com mediana ({median_val})")
        else: # Fallback para tipos inesperados
            print(f"- AVISO: Coluna '{col}' com tipo '{df[col].dtype}' não tratada. Preenchida com 'UNKNOWN_VALUE'.")
            df[col] = df[col].fillna('UNKNOWN_VALUE') # Corrigido fillna
    else:
        print("\nNão há mais valores NaN para preencher no DataFrame. Dados limpos de NaN's!")

# 2.10 Agrupar Categorias Raras e Aplicar One-Hot Encoding
print("\n--- Agrupando Categorias Raras e Aplicando One-Hot Encoding ---")
object_cols_for_ohe_initial = df.select_dtypes(include='object').columns.tolist()

# NOTA: 'Action' e 'Severity' não são removidas daqui, pois serão usadas como target.
# Estas duas colunas serão protegidas da remoção por NaNs
# Elas também serão excluídas do One-Hot Encoding
if 'Action' in object_cols_for_ohe_initial: object_cols_for_ohe_initial.remove('Action')
if 'Severity' in object_cols_for_ohe_initial: object_cols_for_ohe_initial.remove('Severity')

print(f"Colunas 'object' para One-Hot Encoding (antes do agrupamento): {object_cols_for_ohe_initial}")

# Lista de colunas a aplicar o agrupamento de raras categorias antes do OHE
# Aumentado o limiar para 0.5% para ser mais agressivo na redução de colunas
high_cardinality_cols_to_group = [
    'Service', 'App Protocol', 'Interface', 'Origin', 'Client Inbound Interface',
    'Interface Name', 'Policy Name', 'Action', 'Destination Zone', 'Source Zone',
    'Layer Name', 'Log Server Origin', 'Protocol', 'Type', 'Blade', 'Direction of Connection'
]

for col in high_cardinality_cols_to_group:
    if col in df.columns and col in object_cols_for_ohe_initial:
        df = group_rare_categories(df, col, threshold=0.005) # <-- LIMIAR AUMENTADO AQUI (0.5%)
```

4.4. One-Hot Encoding, Hashing Encoding e Verificação de Tipos

NOTA: Dada a elevada cardinalidade das variáveis categóricas e a limitação de memória RAM para o One-Hot Encoding, optou-se pela técnica de Feature Hashing, permitindo representar o espaço de atributos de forma compacta e eficiente.

```
print("\n--- 4.4.: One-Hot Encoding Final, Hashing Encoding e Verificação de Tipos ---")

# --- Remover Colunas Identificadoras Problemáticas (ID, Marker, High Level Log key) ---
cols_to_remove_critical = ['ID', 'Marker', 'High Level Log key']
existing_cols_to_remove = [col for col in cols_to_remove_critical if col in df.columns]

if existing_cols_to_remove:
    df.drop(columns=existing_cols_to_remove, inplace=True)
    print(f"Colunas identificadoras removidas (devido à cardinalidade extrema): {existing_cols_to_remove}")
    print(f"Número de colunas restantes no DataFrame após remoção crítica: {df.shape[1]}")
else:
    print("\nNenhuma das colunas identificadoras críticas (ID, Marker, High Level Log key) encontrada ou já removida.")

# --- Aplicar Hashing Encoding para 'Destination' e 'Source' ---
# Hashing Encoding é usado para colunas com altíssima cardinalidade para controlar o número de features de saída.
# Foi definido um número fixo de colunas de saída (n_components).
hashing_cols = ['Destination', 'Source']
num_components_dest = 100 # Número de colunas de saída para Destination
num_components_src = 50 # Número de colunas de saída para Source

for col in hashing_cols:
    if col in df.columns and df[col].dtype == 'object':
        print(f"Aplicando Hashing Encoding à coluna '{col}'...")
        n_comp = num_components_dest if col == 'Destination' else num_components_src

        hasher = HashingEncoder(cols=[col], n_components=n_comp)
        df_hashed = hasher.fit_transform(df[col]) # Transformar apenas a coluna

        # Renomear as novas colunas com um prefixo claro
        df_hashed.columns = [f'{col}_hashed_{i}' for i in range(n_comp)]

        # Juntar as novas colunas hashed ao DataFrame principal
        df = pd.concat([df.drop(columns=[col]), df_hashed], axis=1)
        print(f" - Coluna '{col}' transformada em {n_comp} features hashed. Colunas atuais: {df.shape[1]}")
    else:
        print(f"Aviso: Coluna '{col}' não encontrada ou não é do tipo 'object' para Hashing Encoding. Ignorado.")

# --- Aplicar One-Hot Encoding Final (para as restantes colunas 'object' após agrupamento) ---
object_cols_for_ohc_final = df.select_dtypes(include='object').columns.tolist()

# EXCLUIR 'Action' e 'Severity' do OHE, pois são as nossas targets
cols_to_exclude_from_ohc = ['Action', 'Severity']
object_cols_for_ohc_final = [col for col in object_cols_for_ohc_final if col not in cols_to_exclude_from_ohc]

if object_cols_for_ohc_final:
    print(f"Aplicando One-Hot Encoding final às colunas restantes: {object_cols_for_ohc_final}")
    df = pd.get_dummies(df, columns=object_cols_for_ohc_final, drop_first=True, dtype=bool)
    print(f"DataFrame após One-Hot Encoding. Número total de colunas: {df.shape[1]}")
else:
    print("\nNão há colunas 'object' restantes para One-Hot Encoding (exceto as targets).")

# --- Remover Features de Baixa Variância (após todas as codificações) ---
print("\n--- Removendo Features de Baixa Variância ---")
cols_to_drop_low_variance = []
total_rows = len(df)
variance_threshold = 0.999

for col in df.columns:
    if df[col].dtype in ['bool', 'int', 'float']: # Aplicar a colunas binárias/numéricas
        value_counts = df[col].value_counts(normalize=True)
        if not value_counts.empty:
            if value_counts.iloc[0] > variance_threshold:
                # Não remover as colunas que serão targets
                if col not in ['Action', 'Severity', 'Is_Anomaly_Action', 'Is_Anomaly_Severity']:
                    cols_to_drop_low_variance.append(col)
            elif len(value_counts) == 1: # Coluna constante
                if col not in ['Action', 'Severity', 'Is_Anomaly_Action', 'Is_Anomaly_Severity']:
                    cols_to_drop_low_variance.append(col)

if cols_to_drop_low_variance:
    df.drop(columns=cols_to_drop_low_variance, inplace=True)
    print(f"Colunas de baixa variância removidas: {len(cols_to_drop_low_variance)}. Colunas atuais: {df.shape[1]}")
else:
    print("Nenhuma coluna de baixa variância encontrada para remover.")

# --- Verificação final dos tipos de dados ---
print("\nVerificação final dos tipos de dados (após todo o pré-processamento):")
display(df.dtypes.value_counts().to_frame(name='Count of Dtype'))
```

4.5. Criação e Preparação da Coluna Target 'Severity'

```
print("\n--- 4.5: Criação e Preparação da Coluna Target 'Severity' ---")

# Criar Is_Anomaly_Severity
if 'Severity' in df.columns:
    print("Coluna 'Severity' encontrada no DataFrame.")

    # Preencher NaNs na coluna 'Severity' antes de criar a target
    # Isto garante que o .apply() não falha com valores NaN, tratando-os como não-anomalia.
    severity_col_temp = df['Severity'].fillna('UNKNOWN_SEVERITY')

    # Definir 'High' e 'Critical' como anomalias (1). Outras severidades serão 0.
    # Pode ajustar 'anomalous_severities_list' conforme a sua definição de anomalia.
    anomalous_severities_list = ['High', 'Critical']

    df['Is_Anomaly_Severity'] = severity_col_temp.apply(lambda x: 1 if x in anomalous_severities_list else 0).astype(int)

    # Confirmar que a coluna foi criada
    if 'Is_Anomaly_Severity' in df.columns:
        print(f"Coluna 'Is_Anomaly_Severity' criada com sucesso.")
        print(f"Distribuição:\n{df['Is_Anomaly_Severity'].value_counts(normalize=True)}")
    else:
        print("ERRO: 'Is_Anomaly_Severity' não foi criada apesar de 'Severity' estar presente.")

else: # Este bloco else só é acionado se 'Severity' NÃO estiver em df.columns
    print("AVISO CRÍTICO: Coluna 'Severity' não encontrada no DataFrame para criar 'Is_Anomaly_Severity'.")
    print("Verifique se a Célula 3.1 removeu indevidamente a coluna 'Severity'.")
    # Para evitar o KeyError subsequente se a coluna não foi criada
    df['Is_Anomaly_Severity'] = -1 # Atribuir um valor temporário para evitar crash se não houver Severity

print(f"\nDataFrame atual tem {df.shape[0]} linhas e {df.shape[1]} colunas (incluindo a nova target).")
```

5. Análises

5.1. Análise do DataFrame Final

```
print("\n--- 5.1: VERIFICAÇÃO DO DATAFRAME FINAL (Antes de Salvar) ---")

if 'df' not in locals() or df is None:
    print("Erro: DataFrame 'df' não encontrado. Execute as células anteriores.")
elif 'Is_Anomaly_Severity' not in df.columns:
    print("Erro: Coluna 'Is_Anomaly_Severity' não encontrada. Execute a Célula 4.5")
else:
    print(f"\nFormato final do DataFrame: {df.shape[0]} linhas, {df.shape[1]} colunas.")
    display(df.head())

    print("\nTipos de dados das colunas:")
    display(df.dtypes.value_counts().to_frame(name='Contagem'))

    print("\nColunas do tipo 'object' restantes (deve ser uma lista vazia ou targets não convertidas):")
    display(df.select_dtypes(include='object').columns.tolist())

    print(f"\nDistribuição da coluna target 'Is_Anomaly_Severity':")
    display(df['Is_Anomaly_Severity'].value_counts(normalize=True))

    print(f"\nLista de todas as colunas no DataFrame (top 20):")
    print(df.columns.tolist()[:20])

print("\n--- FIM DA VERIFICAÇÃO ---")
```

5.2. Análise de Colunas Constantes e Quase Constantes

```
print("\n--- 5.2: ANÁLISE DE COLUNAS CONSTANTES E QUASE CONSTANTES ---")

if 'df' not in locals() or df is None:
    print("Erro: DataFrame 'df' não encontrado. Execute as células anteriores.")
else:
    total_rows = len(df)
    print(f"Total de linhas no DataFrame: {total_rows}")
    print(f"Número total de colunas a analisar: {df.shape[1]}")

    constant_columns = []
    near_constant_columns = []

    # Definir o limiar para "quase constante"
    near_constant_threshold = 0.999

    for col in df.columns:
        if col == 'Is_Anomaly_Severity' or col == 'Is_Anomaly_Action':
            continue

        num_unique_values = df[col].nunique()

        if num_unique_values == 1:
            # Coluna é constante (apenas 1 valor único)
            constant_columns.append(col)
        elif num_unique_values > 1:
            # Não é constante, mas pode ser quase constante
            # Pega a proporção da categoria mais frequente
            most_frequent_proportion = df[col].value_counts(normalize=True).iloc[0]

            if most_frequent_proportion > near_constant_threshold:
                near_constant_columns.append(col)

    print("\n--- Resultados da Análise ---")

    if constant_columns:
        print(f"\n(len(constant_columns)) Colunas IDENTIFICADAS COMO CONSTANTES (1 valor único):")
        print(sorted(constant_columns))
        # Opcional: Mostrar o valor constante para as primeiras 5
        print("\nExemplo de valores para colunas constantes:")
        for col in constant_columns[:5]:
            display(f"- Coluna '{col}': Valor = {df[col].iloc[0]}")
    else:
        print("\nNenhuma coluna identificada como constante.")

    if near_constant_columns:
        print(f"\n(len(near_constant_columns)) Colunas identificadas como quase constantes (>{near_constant_threshold*100}% dos valores são iguais):")
        print(sorted(near_constant_columns))
        # Opcional: Mostrar a proporção e o valor mais frequente para as primeiras 5
        print("\nExemplo de valores para colunas quase constantes:")
        for col in near_constant_columns[:5]:
            most_freq_val = df[col].value_counts().index[0]
            most_freq_prop = df[col].value_counts(normalize=True).iloc[0]
            display(f"- Coluna '{col}': Valor mais frequente = {most_freq_val}, Proporção = {most_freq_prop:.4f}")
    else:
        print(f"\nNenhuma coluna identificada como quase constante (>{near_constant_threshold*100}% dos valores iguais).")

    print("\n--- FIM DA ANÁLISE ---")
```

6. Guardar CSV

```
print("\n--- Guardar ficheiro em formato CSV ---")

# Criar uma cópia do DataFrame
df_severity_target = df.copy()

# Remover as colunas originais 'Action' e 'Severity' (se ainda existirem, pois agora são a base da target ou não são features)
# e qualquer outra coluna target que não seja 'Is_Anomaly_Severity'
cols_to_drop_for_severity_target = [col for col in ['Action', 'Severity', 'Is_Anomaly_Action'] if col in df_severity_target.columns]
df_severity_target.drop(columns=cols_to_drop_for_severity_target, inplace=True, errors='ignore')

final_csv_filename_severity = 'final_preprocessed_logs_target_severity.csv'
final_csv_filepath_severity = final_csv_filename_severity

print(f"\n--- Guardando DataFrame com 'Is_Anomaly_Severity' como Target ---")
print(f"Seu DataFrame para 'Severity' tem {df_severity_target.shape[0]} linhas e {df_severity_target.shape[1]} colunas.")
print(f"Tentando guardar o DataFrame em formato CSV em '{os.path.abspath(final_csv_filepath_severity)}'...")
try:
    df_severity_target.to_csv(final_csv_filepath_severity, index=False)
    print(f"DataFrame para 'Severity' guardado com sucesso em formato CSV!")
    print(f"O ficheiro '{final_csv_filename_severity}' está agora pronto para ser carregado para a modelagem de ML/DL.")
except Exception as e:
    print(f"Erro ao guardar o ficheiro CSV para 'Severity': {e}")
```

APÊNDICE B

Modelos de Machine Learning Supervisionado – Códigos Fonte

APÊNDICE **B1**

Random Forest – Código Fonte

0. Bibliotecas e Dependências

```
# Manipulação de dados
import pandas as pd
import numpy as np
import os

# Modelagem e Pré-processamento
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    classification_report, confusion_matrix,
    roc_auc_score, precision_recall_curve, auc
)

# Visualização
import matplotlib.pyplot as plt
import seaborn as sns

# Configurações de exibição
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

1. Carregamento e Preparação do Dataset

NOTA: Nesta etapa, é realizado o carregamento do dataset consolidado. O procedimento inclui a separação da variável dependente (`Is_Anomaly_Severity`) e a validação de integridade dos dados, assegurando que apenas atributos numéricos e booleanos sejam mantidos para a fase de treino. Nesta etapa, é realizado o carregamento do dataset consolidado. O procedimento inclui a separação da variável dependente (`Is_Anomaly_Severity`) e a validação de integridade dos dados, assegurando que apenas atributos numéricos e booleanos sejam mantidos para a fase de treino.

```
: # Definição do caminho do arquivo
csv_path = 'final_preprocessed_logs_target_severity.csv'

if not os.path.exists(csv_path):
    raise FileNotFoundError(f"Ficheiro {csv_path} não encontrado.")

# Leitura do dataset
df_model = pd.read_csv(csv_path, low_memory=False)
print(f"Dataset carregado: {df_model.shape[0]} registros e {df_model.shape[1]} atributos.")

# Definição do vetor alvo e matriz de características
target_col = 'Is_Anomaly_Severity'

if target_col in df_model.columns:
    X = df_model.drop(columns=[target_col])
    y = df_model[target_col]
else:
    raise KeyError(f"Coluna alvo '{target_col}' não encontrada no dataset.")

# Filtragem de tipos de dados (Saneamento final)
non_numeric = X.select_dtypes(excludes=['number', 'bool']).columns.tolist()
if non_numeric:
    X = X.drop(columns=non_numeric)

print(f"Distribuição da classe alvo:\n{y.value_counts(normalize=True)}")
```

2. Divisão de Amostras e Escalonamento

NOTA: O dataset é segmentado em conjuntos de treino (80%) e teste (20%), utilizando amostragem estratificada para preservar a proporção da classe minoritária. Embora o Random Forest seja robusto a variações de escala, o escalonamento pode ser aplicado para fins de consistência metodológica.

```
# Divisão estratificada (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)

# Identificação de atributos numéricos para normalização
numerical_cols = X_train.select_dtypes(include=['number']).columns

# Normalização (StandardScaler)
scaler = StandardScaler()
X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])

print(f"Dimensões do Treino: {X_train.shape}")
print(f"Dimensões do Teste: {X_test.shape}")
```

3. Treino do Modelo Random Forest

NOTA: Implementação do classificador Random Forest. Dada a natureza desequilibrada do problema de detecção de anomalias, utiliza-se o parâmetro `class_weight='balanced'`, que ajusta os pesos das classes inversamente à sua frequência, conferindo maior relevância à classe minoritária durante a construção das árvores de decisão.

```
# Inicialização do modelo com pesos balanceados
rf_model = RandomForestClassifier(
    n_estimators=100,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

# Fase de ajuste (treino)
rf_model.fit(X_train, y_train)
```

4. Avaliação do Modelo Random Forest

NOTA: A avaliação do modelo é realizada através de métricas adequadas a datasets desbalanceados. Além do relatório de classificação e matriz de confusão, são analisadas as curvas ROC (Receiver Operating Characteristic) e Precision-Recall, sendo esta última particularmente relevante para identificar a eficácia na detecção de anomalias raras.

```
# 1. Geração de Previsões e Probabilidades
# 'y_pred' é utilizado para métricas de classe; 'y_proba' para métricas de limiar (threshold)
y_pred = rf_model.predict(X_test)
y_proba = rf_model.predict_proba(X_test)[:, 1]

# 2. Relatório de Classificação (Precision, Recall e F1-Score)
# Essencial para observar o desempenho individual na classe minoritária (Anomalias)
print("Relatório de Classificação Final:")
print(classification_report(y_test, y_pred))

# 3. Cálculo da Área Sob as Curvas (AUC)
roc_auc = roc_auc_score(y_test, y_proba)
precision, recall, _ = precision_recall_curve(y_test, y_proba)
pr_auc = auc(recall, precision)

print(f"Métricas de Área:")
print(f" - ROC-AUC: {roc_auc:.4f}")
print(f" - PR-AUC (Average Precision): {pr_auc:.4f}")

# 4. Matriz de Confusão
# Fornece a contagem absoluta de Falsos Positivos e Falsos Negativos
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - Random Forest')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 5. Visualização da Curva Precision-Recall
# Curva prioritária para análise em datasets altamente desbalanceados
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkblue', lw=2, label=f'Random Forest (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkblue')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - Random Forest')
plt.legend(loc='lower left')
plt.grid(True, linestyle=':', alpha=0.7)
plt.tight_layout()
plt.show()
```

APÊNDICE B2

XGBoost – Código Fonte

3. Config. e Treino do Modelo XGBoost

NOTA: Implementação do classificador XGBoost. Para mitigar o desequilíbrio extremo entre classes, utiliza-se o hiperparâmetro `scale_pos_weight`. Este parâmetro atribui um peso superior à classe positiva (anomalias), calculado como o rácio entre instâncias negativas e positivas, penalizando erros de classificação na classe minoritária durante o processo de otimização.

```
# Cálculo do rácio para balanceamento de classes
ratio = float(y_train.value_counts()[0]) / y_train.value_counts()[1]

# Inicialização do classificador XGBoost
xgb_model = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=6,
    learning_rate=0.1,
    scale_pos_weight=ratio, # Tratamento de desequilíbrio
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss',
    n_jobs=-1
)

# Treino do modelo
xgb_model.fit(X_train, y_train)
```

4. Avaliação do Modelo XGBoost

```
# Previsões e Probabilidades
y_pred = xgb_model.predict(X_test)
y_proba = xgb_model.predict_proba(X_test)[:, 1]

# 1. Relatório de Classificação
print("Relatório de Classificação:")
print(classification_report(y_test, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - XGBoost')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Métricas de Área sob a Curva (AUC)
roc_auc = roc_auc_score(y_test, y_proba)
precision, recall, _ = precision_recall_curve(y_test, y_proba)
pr_auc = auc(recall, precision)

print(f"ROC-AUC Score: {roc_auc:.4f}")
print(f"PR-AUC (Average Precision): {pr_auc:.4f}")

# 4. Visualização da Curva Precision-Recall
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkorange', lw=2, label=f'XGBoost (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkorange')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - XGBoost')
plt.legend(loc="lower left")

plt.grid(True, linestyle=':', alpha=0.7)
plt.tight_layout()
plt.show()
```

APÊNDICE B3

Stochastic Gradient Descent – Código Fonte

3. Implementação do Modelo SGD

NOTA: Treino do classificador via Gradiente Descendente Estocástico (SGD). O parâmetro `loss='hinge'` é utilizado para implementar uma Support Vector Machine (SVM) linear. A estratégia de balanceamento de classes (`class_weight='balanced'`) é fundamental para compensar o enviesamento causado pelo desequilíbrio do dataset.

```
|: # Inicialização do classificador linear via SGD
sgd_model = SGDClassifier(
    loss='hinge',          # Implementação de SVM linear
    penalty='l2',          # Regularização Ridge
    class_weight='balanced', # Ajuste automático de pesos por classe
    random_state=42,
    n_jobs=-1
)

# Ajuste do modelo ao conjunto de treino
sgd_model.fit(X_train, y_train)
```

4. Avaliação do Modelo

```
# Geração de previsões
y_pred = sgd_model.predict(X_test)

# Para o SGD com perda 'hinge', as probabilidades são calculadas via distância ao hiperplano
y_scores = sgd_model.decision_function(X_test)

# 1. Relatório de Classificação
print("Métricas de Classificação:")
print(classification_report(y_test, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - SGDClassifier')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test, y_scores)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkred', label=f'SGD-SVM (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkred')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - SGDClassifier')
plt.legend(loc='lower left')
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()
```

APÊNDICE **B4**

XGBoost Otimizado – Código Fonte

5. Otimização de Hiperparâmetros com RandomizerSearchCV

```
] : from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
from sklearn.metrics import make_scorer, f1_score
import matplotlib.pyplot as plt
import numpy as np # Para np.arange
import xgboost as xgb # Adicionar este import aqui caso a célula seja corrida isoladamente no futuro

print("--- 5: Otimização de Hiperparâmetros ---")

if 'X_train' not in locals() or 'y_train' not in locals():
    print("ERRO: As variáveis de treino (X_train, y_train) não estão definidas. Execute as Células 1 e 2.")
    raise NameError("Variáveis de treino ausentes.")

# 1. Definir o Modelo Base XGBoost
# n_jobs=1 para evitar MemoryError durante a validação cruzada em paralelo
# use_label_encoder=False e eval_metric='logloss' para evitar warnings/erros.

# Calcula o scale_pos_weight com base na distribuição real do y_train
neg_count = y_train.value_counts()[0]
pos_count = y_train.value_counts()[1]
scale_pos_weight_value = neg_count / pos_count
print(f"Calculado scale_pos_weight: {scale_pos_weight_value:.2f}")

xgb_model_base = xgb.XGBClassifier(
    objective='binary:logistic',
    scale_pos_weight=scale_pos_weight_value,
    random_state=42,
    n_jobs=1, # Definido para 1 (sem paralelização) para evitar erro de memória durante a busca
    use_label_encoder=False,
    eval_metric='logloss' # Métrica de avaliação interna para o XGBoost
)

# 2. Definir a Grelha de Hiperparâmetros para Pesquisar
# Estes são exemplos. Pode ajustar os intervalos e os tipos de parâmetros.
# Para RandomizedSearchCV, não precisamos de testar TODOS os valores.
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500], # Número de árvores
    'learning_rate': [0.01, 0.05, 0.1, 0.2], # Taxa de aprendizagem
    'max_depth': [3, 5, 7, 10], # Profundidade máxima da árvore
    'subsample': [0.6, 0.8, 1.0], # Fração de amostras para treinar cada árvore
    'colsample_bytree': [0.6, 0.8, 1.0], # Fração de features para treinar cada árvore
    'gamma': [0, 0.1, 0.2], # Redução mínima da perda para fazer uma divisão
    'min_child_weight': [1, 5, 10] # Peso mínimo total da instância necessária num filho
}

# 3. Definir a Estratégia de Validação Cruzada (Cross-Validation)
# StratifiedKFold é essencial para manter a proporção de classes em cada fold
cv_strategy = StratifiedKFold(n_splits=3, shuffle=True, random_state=42) # n_splits=3 para acelerar a busca inicial

# 4. Definir a Métrica de Avaliação para a Busca (Scoring)
# make_scorer permite usar uma métrica específica para a otimização
# F1-score é uma boa métrica para problemas desbalanceados (balanceia Precision e Recall)
scorer = make_scorer(f1_score, pos_label=1) # Foca no F1-score da classe 'Anomalia'

# 5. Configurar o RandomizedSearchCV
# n_iter: número de iterações (combinações de parâmetros) a testar.
# Quanto maior, mais completo, mas mais demorado. Começaremos com 50.
random_search_xgb = RandomizedSearchCV(
    estimator=xgb_model_base,
    param_distributions=param_dist,
    n_iter=50, # Número de combinações aleatórias a testar

    cv=cv_strategy,
    scoring=scorer,
    verbose=2, # Nível de detalhe da saída
    random_state=42,
    n_jobs=1, # DEFINIDO PARA 1 (sem paralelização) para evitar erro de memória durante a busca paralela
    error_scores='raise' # Para ver erros durante o treino de cada combinação
)

print("\nIniciando a busca por hiperparâmetros ótimos com RandomizedSearchCV...")
# A busca é feita APENAS nos dados de TREINO.
random_search_xgb.fit(X_train, y_train)

print("\nBusca por hiperparâmetros concluída!")

# 6. Apresentar os Melhores Resultados
print("\n--- Melhores Hiperparâmetros Encontrados ---")
print(random_search_xgb.best_params_)

print("\n--- Melhor F1-score (Classe 1) com Cross-Validation ---")
print(f"F1-score (Anomalia) médio no CV: {random_search_xgb.best_score_:.4f}")

# O melhor modelo treinado com os parâmetros ótimos
best_xgb_model = random_search_xgb.best_estimator_
```

6. Avaliação do Modelo XGBoost Otimizado

```
#from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
#import matplotlib.pyplot as plt
#import seaborn as sns
#import pandas as pd

print("--- 6: Avaliação do Modelo XGBoost Otimizado ---")

if 'best_xgb_model' not in locals():
    print("ERRO: O modelo XGBoost otimizado não está definido.")
    raise NameError("Modelo otimizado não treinado.")

# 1. Fazer Previsões no Conjunto de Teste com o Modelo Otimizado
print("\nFazendo previsões no conjunto de teste com o modelo otimizado...")
y_pred_optimized_xgb = best_xgb_model.predict(X_test) # Previsões de classe (0 ou 1)
y_pred_proba_optimized_xgb = best_xgb_model.predict_proba(X_test)[: , 1] # Probabilidades da classe 1 (anomalia)

# 2. Avaliar o Desempenho do Modelo Otimizado
print("\n--- Relatório de Classificação do Modelo XGBoost Otimizado ---")
print(classification_report(y_test, y_pred_optimized_xgb))

print("\n--- Matriz de Confusão do Modelo XGBoost Otimizado ---")
cm_optimized_xgb = confusion_matrix(y_test, y_pred_optimized_xgb)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_optimized_xgb, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)', yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão (XGBoost Otimizado)')
plt.show()

roc_auc_optimized_xgb = roc_auc_score(y_test, y_pred_proba_optimized_xgb)
print(f"\nROC-AUC Score (XGBoost Otimizado): {roc_auc_optimized_xgb:.4f}")

precision_optimized_xgb, recall_optimized_xgb, _ = precision_recall_curve(y_test, y_pred_proba_optimized_xgb)
pr_auc_optimized_xgb = auc(recall_optimized_xgb, precision_optimized_xgb)
print(f"PR-AUC Score (XGBoost Otimizado): {pr_auc_optimized_xgb:.4f}")

plt.figure(figsize=(8, 6))
plt.plot(recall_optimized_xgb, precision_optimized_xgb, label=f'PR Curve XGBoost Otimizada (area = {pr_auc_optimized_xgb:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (XGBoost Otimizado)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()

print("\n--- Avaliação do Modelo XGBoost Otimizado Concluída ---")
```

APÊNDICE C

Modelos de Machine Learning Não Supervisionado – Códigos Fonte

APÊNDICE C1

Isolation Forest 1 e Isolation Forest 2 – Código Fonte

3. Implementação do Modelo Isolation Forest

NOTA: O algoritmo Isolation Forest é instanciado para a detecção de anomalias baseada em isolamento. O parâmetro de contaminação é ajustado conforme a proporção real de anomalias identificada no dataset, permitindo que o modelo defina o limiar de decisão de forma estatisticamente fundamentada.

```
: # Cálculo da taxa de contaminação baseada na classe minoritária
contamination_rate = y.value_counts(normalize=True)[1]

# Inicialização do Modelo Isolation Forest
iso_forest_model = IsolationForest(
    n_estimators=100,
    contamination=contamination_rate,
    random_state=42,
    n_jobs=-1
)

# Ajuste do modelo ao conjunto de treino
iso_forest_model.fit(X_train)
```

4. Avaliação do Modelo

```
# Geração de previsões no conjunto de teste
# Mapeamento: -1 (anomalia) -> 1; 1 (normal) -> 0
y_pred_raw = iso_forest_model.predict(X_test)
y_pred = np.where(y_pred_raw == -1, 1, 0)

# Cálculo da pontuação de anomalia (anomaly score) para métricas de curva
# Nota: Inverte-se o sinal pois scores menores indicam anomalias no Isolation Forest
y_scores = -iso_forest_model.decision_function(X_test)

# 1. Relatório de Classificação
print("Métricas de Classificação:")
print(classification_report(y_test, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - Isolation Forest')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Análise da Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test, y_scores)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkgreen', lw=2, label=f'Isolation Forest (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkgreen')
plt.xlabel('Recall')
plt.ylabel('Precision')

plt.title('Curva Precision-Recall - Isolation Forest')
plt.legend(loc='lower left')
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()

print(f"ROC-AUC: {roc_auc_score(y_test, y_scores):.4f}")
print(f"PR-AUC: {pr_auc:.4f}")
```

3. Parametrização e Ajuste

```
# Definimos 10% como taxa fixa para o Isolation Forest 2
contamination_value = 0.01

print(f"(Contaminação: {contamination_value * 100}%)")

# 1. Inicializar o modelo com o nome que tu queres
iso_forest_model = IsolationForest(
    n_estimators=100,
    max_samples='auto',
    contamination=contamination_value,
    random_state=42,
    n_jobs=-1
)

# 2. Ajuste do modelo ao conjunto de treino
iso_forest_model.fit(X_train)
```

4. Avaliação do Modelo Isolation Forest

```
# Geração de previsões (Mapeamento: -1 -> 1; 1 -> 0)
y_pred_raw = iso_forest_model.predict(X_test)
y_pred = np.where(y_pred_raw == -1, 1, 0)

# Cálculo de scores de anomalia (inversão do sinal para métricas de probabilidade)
y_scores = -iso_forest_model.decision_function(X_test)

# 1. Relatório de Classificação
print("Métricas de Classificação:")
print(classification_report(y_test, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - Isolation Forest 2')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Visualização da Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test, y_scores)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkorange', lw=2, label=f'Isolation Forest (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkorange')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - Isolation Forest 2')
plt.legend(loc="lower left")
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()
```

APÊNDICE C2

K-means 1 e K-means 2 – Código Fonte

3. Implementação do Modelo K-means

```
# Inicialização do modelo K-Means
# Nota: O número de clusters (k) pode ser ajustado com base no método do cotovelo (Elbow Method)
kmeans_model = KMeans(n_clusters=8, random_state=42, n_init=10)

# Ajuste do modelo ao conjunto de treino
kmeans_model.fit(X_train)
```

4. Avaliação do Modelo

NOTA: A detecção de anomalias via K-Means baseia-se na distância euclidiana mínima de cada instância ao centroide mais próximo. Instâncias que apresentam distâncias superiores a um determinado limiar estatístico (percentil) são classificadas como anomalias.

```
# Cálculo das distâncias para o centroide mais próximo
distances_test = kmeans_model.transform(X_test)
min_distances = np.min(distances_test, axis=1)

# Definição do limiar de anomalia (Thresholding) baseado em percentil
threshold = np.percentile(min_distances, 99.8)
y_pred = (min_distances > threshold).astype(int)

# 1. Relatório de Classificação
print("Métricas de Classificação:")
print(classification_report(y_test, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - K-Means')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Análise da Curva Precision-Recall
# Nota: Para a curva PR, utilizamos as distâncias negativas (scores de anomalia)
precision, recall, _ = precision_recall_curve(y_test, min_distances)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='purple', lw=2, label=f'K-Means (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='purple')
plt.xlabel('Recall')

plt.ylabel('Precision')
plt.title('Curva Precision-Recall - K-Means')
plt.legend(loc="lower left")
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()

print(f"ROC-AUC Score: {roc_auc_score(y_test, min_distances):.4f}")
print(f"PR-AUC Score: {pr_auc:.4f}")
```

3. Modelação por Agrupamento

NOTA: A implementação do algoritmo K-Means visa capturar a estrutura natural dos dados normais. O número de clusters (k=50) é seleccionado para permitir uma modelação granular de diferentes perfis de comportamento nos logs. O treino é realizado exclusivamente com o conjunto de treino.

```
# Inicialização e treino do modelo K-Means
kmeans_model = KMeans(
    n_clusters=50,
    random_state=42,
    n_init='auto'
)

# Ajuste do modelo
kmeans_model.fit(X_train)
```

4. Avaliação do Modelo

```
# Atribuição de clusters e cálculo de distâncias
test_cluster_labels = kmeans_model.predict(X_test)

# Cálculo da distância de cada ponto ao seu centroide atribuído
distances = np.array([
    np.linalg.norm(X_test.iloc[i] - kmeans_model.cluster_centers_[test_cluster_labels[i]])
    for i in range(len(X_test))
])

# Definição do limiar de anomalia (Thresholding)
threshold = np.percentile(distances, 99.9)
y_pred = (distances > threshold).astype(int)

# 1. Relatório de Classificação
print("Relatório de Classificação:")
print(classification_report(y_test, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão - K-Means')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test, distances)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='purple', lw=2, label=f'K-Means (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='purple')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - K-Means')
plt.legend(loc="lower left")
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()

print(f"ROC-AUC Score: {roc_auc_score(y_test, distances):.4f}")
```

APÊNDICE C3

DBSCAN 1 e DBSCAN 2 – Código Fonte

3. Estimativa de Parâmetro 'eps'

```
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import numpy as np

print("--- 3: Estimativa de Parâmetro 'eps' ---")

if 'X_train' not in locals():
    print("ERRO: A variável X_train não está definida.")
    raise NameError("Variáveis de treino ausentes.")

# 1. Definir o número de vizinhos (min_samples)
# Para o DBSCAN, min_samples é o número mínimo de pontos para formar um cluster central.
# Uma regra prática para alta dimensionalidade é min_samples = 2 * n_features.
n_features = X_train.shape[1]
min_samples_dbscan = 2 * n_features # = 2 * 297 = 594

print(f"Estimando 'eps' com min_samples = {min_samples_dbscan} (2 * número de features).")

# 2. Encontrar as distâncias aos 'min_samples_dbscan' vizinhos mais próximos
# n_neighbors deve ser pelo menos min_samples_dbscan.
neigh = NearestNeighbors(n_neighbors=min_samples_dbscan)
nbrs = neigh.fit(X_train)
distances, indices = nbrs.kneighbors(X_train)

# 3. Ordenar as distâncias e plotar
# Plotamos as distâncias do (min_samples_dbscan)-ésimo vizinho mais próximo.
distances = np.sort(distances[:, min_samples_dbscan-1], axis=0) # Distância ao (k-ésimo) vizinho

plt.figure(figsize=(10, 6))
plt.plot(distances)
plt.xlabel("Pontos de Dados (ordenados por distância)")
plt.ylabel(f"Distância ao {min_samples_dbscan}-ésimo Vizinho Mais Próximo")

plt.title(f"Gráfico da Distância ao {min_samples_dbscan}-ésimo Vizinho (para Estimativa de eps)")
plt.grid(True)
plt.show()
```

4. Treino do Modelo DBSCAN

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import seaborn as sns

print("--- 4: Treino do Modelo DBSCAN ---")

if 'X_train' not in locals():
    print("ERRO: A variável X_train não está definida.")
    raise NameError("Variáveis de treino ausentes.")

# 1. Inicializar o Modelo DBSCAN
# eps: O raio da vizinhança. Este valor é fundamental e deve ser escolhido com base no gráfico da célula 3.
# min_samples: O número mínimo de pontos para formar um cluster central. (2 * n_features, ou ajustado).

# Use o valor de min_samples calculado na célula anterior
n_features = X_train.shape[1]
min_samples_dbscan = 2 * n_features # = 594

# Escolha do eps: Este é o valor que você deve estimar do gráfico da Célula 10.0.
# Se o gráfico não tem um cotovelo claro, pode precisar de experimentar.
# Para começar, um valor pequeno como 0.5 (se os dados forem escalados) pode ser uma tentativa.
# Ou pode usar um valor ainda menor se o gráfico sugerir.
# Vamos usar 0.5 como um ponto de partida genérico se o usuário não fornecer um.
eps_value = 0.5 # <- AJUSTE ESTE VALOR COM BASE NO GRÁFICO DA CÉLULA 10.0!

dbscan_model = DBSCAN(
    eps=eps_value,
    min_samples=min_samples_dbscan,
    n_jobs=-1 # Usar todos os cores do processador
)

print(f"\nModelo DBSCAN inicializado: {dbscan_model}")

# 2. Treinar o Modelo
dbscan_model.fit(X_train)
print("Treino concluído!")
```

5. Avaliação do Modelo DBSCAN

```
from sklearn.neighbors import NearestNeighbors # Necessário para scores de anomalia para ROC/PR

print("--- 5: Avaliação do Modelo DBSCAN ---")

if 'dbscan_model' not in locals():
    print("ERRO: O modelo não está definido.")
    raise NameError("Modelo não treinado.")

# Reconstruir rótulos para o treino (dbscan_model.labels_)
# Converter os rótulos de cluster (-1, 0, 1...) para a nossa target 0/1 (Normal/Anomalia)
# -1 (ruído) -> 1 (anomalia)
# 0, 1, 2... (clusters) -> 0 (normal)
y_pred_dbscan_train = np.where(dbscan_model.labels_ == -1, 1, 0)
```

```
# Para o relatório de classificação e matriz de confusão:
# Mapear os rótulos do DBSCAN para 0/1 (Ruído = Anomalia = 1, Cluster = Normal = 0)
# Comparar com y_train.
y_pred_dbscan_train_labels = np.where(dbscan_model.labels_ == -1, 1, 0)
```

```
# Para ROC-AUC e PR-AUC: usar a distância ao k-ésimo vizinho mais próximo como score de anomalia
# Acha-se que pontos com maior distância ao k-ésimo vizinho são outliers.
# Precisamos de calcular as distâncias do X_train para o X_train, e usar o k do min_samples_dbscan
min_samples_dbscan_for_scores = dbscan_model.min_samples # Reutiliza min_samples_dbscan do modelo
neigh_scores = NearestNeighbors(n_neighbors=min_samples_dbscan_for_scores)
nbrs_scores = neigh_scores.fit(X_train)
distances_scores, _ = nbrs_scores.kneighbors(X_train)
# A pontuação de anomalia é a distância ao k-ésimo vizinho mais próximo. Invertamos o sinal para que maior = mais anômalo.
anomaly_scores_dbscan_train = distances_scores[:, min_samples_dbscan_for_scores-1]
```

```
# Relatório de Classificação e Matriz de Confusão para X_train
print("\n--- Relatório de Classificação DBSCAN (Avaliado no Conjunto de Treino) ---")
print(classification_report(y_train, y_pred_dbscan_train_labels))

print("\n--- Matriz de Confusão DBSCAN (Avaliado no Conjunto de Treino) ---")
cm_dbscan_train = confusion_matrix(y_train, y_pred_dbscan_train_labels)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_dbscan_train, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão (DBSCAN - Treino)')
plt.show()

# ROC-AUC Score (para o conjunto de treino)
roc_auc_dbscan_train = roc_auc_score(y_train, anomaly_scores_dbscan_train)
print(f"\nROC-AUC Score (DBSCAN - Treino): {roc_auc_dbscan_train:.4f}")

# Curva Precision-Recall (PR Curve) e PR-AUC (para o conjunto de treino)
precision_dbscan_train, recall_dbscan_train, _ = precision_recall_curve(y_train, anomaly_scores_dbscan_train)
pr_auc_dbscan_train = auc(recall_dbscan_train, precision_dbscan_train)
print(f"\nPR-AUC Score (DBSCAN - Treino): {pr_auc_dbscan_train:.4f}")

plt.figure(figsize=(8, 6))
plt.plot(recall_dbscan_train, precision_dbscan_train, label=f'PR Curve DBSCAN (area = {pr_auc_dbscan_train:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (DBSCAN - Treino)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```

4: Treino do Modelo DBSCAN

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import seaborn as sns

print("--- 4: Treino do Modelo DBSCAN ---")

if 'X_train' not in locals():
    print("ERRO: A variável X_train não está definida. ")
    raise NameError("Variáveis de treino ausentes.")

# 1. Inicializar o Modelo DBSCAN
eps_value = 0.5

# min_samples: O número mínimo de pontos para formar um cluster central.
# Ajustamos para um valor fixo e menor, por exemplo, 30.
min_samples_dbscan_adjusted = 30 # <- MUDANÇA AQUI: min_samples ajustado para 30

dbscan_model = DBSCAN(
    eps=eps_value,
    min_samples=min_samples_dbscan_adjusted, # Usar o valor ajustado
    n_jobs=-1 # Usar todas as cores do processador
)

print(f"\nModelo DBSCAN inicializado: {dbscan_model}")

# 2. Treinar o Modelo
dbscan_model.fit(X_train)
print("Treino concluído!")
```

5: Avaliação do Modelo DBSCAN

```
: from sklearn.neighbors import NearestNeighbors # Necessário para scores de anomalia para ROC/PR

print("--- 5: Avaliação do Modelo DBSCAN ---")

if 'dbscan_model' not in locals():
    print("ERRO: O modelo não está definido.")
    raise NameError("Modelo não treinado.")

y_pred_dbscan_train = np.where(dbscan_model.labels_ == -1, 1, 0)

print("\n--- AVALIAÇÃO DO DBSCAN EM X_TRAIN (Foco na Identificação de Ruído/Anomalias) ---")
print("DBSCAN não tem um método 'predict' direto para dados de teste que não foram usados no fit. ")
print("Vamos avaliar a sua capacidade de identificar anomalias (ruído) dentro do próprio conjunto de treino.")

# Para o relatório de classificação e matriz de confusão:
y_pred_dbscan_train_labels = np.where(dbscan_model.labels_ == -1, 1, 0)

# Para ROC-AUC e PR-AUC: usar a distância ao k-ésimo vizinho mais próximo como score de anomalia
min_samples_dbscan_for_scores = dbscan_model.min_samples
neigh_scores = NearestNeighbors(n_neighbors=min_samples_dbscan_for_scores)
nbrs_scores = neigh_scores.fit(X_train)
distances_scores, _ = nbrs_scores.kneighbors(X_train)
# A pontuação de anomalia é a distância ao k-ésimo vizinho mais próximo. Invertamos o sinal para que maior = mais anômalo.
anomaly_scores_dbscan_train = distances_scores[:, min_samples_dbscan_for_scores-1]

# Relatório de Classificação e Matriz de Confusão para X_train
print("\n--- Relatório de Classificação DBSCAN (Avaliado no Conjunto de Treino) ---")
print(classification_report(y_train, y_pred_dbscan_train_labels))

print("\n--- Matriz de Confusão DBSCAN (Avaliado no Conjunto de Treino) ---")
cm_dbscan_train = confusion_matrix(y_train, y_pred_dbscan_train_labels)

plt.figure(figsize=(6, 5))
sns.heatmap(cm_dbscan_train, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão (DBSCAN - Treino)')
plt.show()

# ROC-AUC Score (para o conjunto de treino)
roc_auc_dbscan_train = roc_auc_score(y_train, anomaly_scores_dbscan_train)
print(f"\nROC-AUC Score (DBSCAN - Treino): {roc_auc_dbscan_train:.4f}")

# Curva Precision-Recall (PR Curve) e PR-AUC (para o conjunto de treino)
precision_dbscan_train, recall_dbscan_train, _ = precision_recall_curve(y_train, anomaly_scores_dbscan_train)
pr_auc_dbscan_train = auc(recall_dbscan_train, precision_dbscan_train)
print(f"PR-AUC Score (DBSCAN - Treino): {pr_auc_dbscan_train:.4f}")

plt.figure(figsize=(8, 6))
plt.plot(recall_dbscan_train, precision_dbscan_train, label=f'PR Curve DBSCAN (area = {pr_auc_dbscan_train:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (DBSCAN - Treino)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```

APÊNDICE D

Modelos de Deep Learning Supervisionado – Código Fonte

APÊNDICE **D1**

LSTM – Código Fonte

2. Preparação para LSTM (Estrutura 3D)

```
: # Definimos n_timesteps = 1 para garantir a detecção individual de cada log
# e evitar a perda de anomalias raras por segmentação.
n_timesteps = 1
n_features = X_train.shape[1]

X_train_lstm = X_train.reshape(-1, n_timesteps, n_features)
X_test_lstm = X_test.reshape(-1, n_timesteps, n_features)

# Converter labels para arrays numpy
y_train_lstm = np.array(y_train)
y_test_lstm = np.array(y_test)
```

4. Implementação do Modelo

```
model = Sequential([
    LSTM(64, input_shape=(n_timesteps, n_features), return_sequences=False),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Cálculo de pesos de classe para lidar com o desbalanceamento extremo
weights = class_weight.compute_class_weight(
    'balanced',
    classes=np.unique(y_train_lstm),
    y=y_train_lstm
)
class_weights = dict(enumerate(weights))

# Callback para evitar overfitting
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)

print("\nIniciando o treino do modelo...")
history = model.fit(
    X_train_lstm, y_train_lstm,
    epochs=50,
    batch_size=32, # Batch size menor para melhor aprendizagem de padrões raros
    validation_split=0.2,
    class_weight=class_weights,
    callbacks=[early_stop],
    verbose=1
)
```

5. Avaliação do Modelo

```
# Previsões
y_proba = model.predict(X_test_lstm).ravel()
y_pred = (y_proba > 0.5).astype(int)

# 1. Relatório de Classificação
print("Relatório de Classificação")
print(classification_report(y_test_lstm, y_pred))

# 2. Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test_lstm, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal', 'Anomalia'], yticklabels=['Normal', 'Anomalia'])
plt.title('Matriz de Confusão - LSTM')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test_lstm, y_proba)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkblue', lw=2, label=f'LSTM (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkblue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - LSTM')
plt.legend(loc="lower left")
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()

# 4. Métricas Adicionais
print(f"\nROC-AUC Score: {roc_auc_score(y_test_lstm, y_proba):.4f}")
print(f"PR-AUC Score: {pr_auc:.4f}")
```

APÊNDICE D2

GRU – Código Fonte

2. Reestruturação para Redes Recorrentes (3D Tensors)

NOTA: Redes do tipo GRU requerem que os dados de entrada sejam organizados em tensores tridimensionais, compostos por amostras, janelas temporais (timesteps) e atributos. Defina-se uma janela de 10 passos para permitir que o modelo aprenda as dependências sequenciais dos eventos de log.

```
n_timesteps = 1
n_features = X_train.shape[1]

X_train_gru = X_train.reshape(-1, n_timesteps, n_features)
X_test_gru = X_test.reshape(-1, n_timesteps, n_features)

y_train_gru = np.array(y_train)
y_test_gru = np.array(y_test)
```

3. Implementação do Modelo

```
model_gru = Sequential([
    GRU(64, input_shape=(n_timesteps, n_features), return_sequences=False),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

model_gru.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Cálculo de pesos para desbalanceamento
weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train_gru), y=y_train_gru)
class_weights = dict(enumerate(weights))

# Configuração do Early Stopping
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Processo de treino
history = model_gru.fit(
    X_train_gru, y_train_gru,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    class_weight=class_weights,
    callbacks=[early_stop],
    verbose=1
)
```

4. Avaliação do Modelo

```
# Geração de previsões probabilísticas e binárias
y_proba = model_gru.predict(X_test_gru).ravel()
y_pred = (y_proba > 0.5).astype(int)

# 1. Relatório de Classificação
print("Métricas de Classificação")
print(classification_report(y_test_gru, y_pred))

# Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test_gru, y_pred), annot=True, fmt='d', cmap='Greens', cbar=False,
            xticklabels=['Normal', 'Anomalia'], yticklabels=['Normal', 'Anomalia'])
plt.title('Matriz de Confusão - GRU')
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.show()

# 3. Análise da Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test_gru, y_proba)
pr_auc = auc(recall, precision)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkgreen', lw=2, label=f'GRU (PR-AUC = {pr_auc:.4f})')
plt.fill_between(recall, precision, alpha=0.1, color='darkgreen')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall - GRU')
plt.legend(loc="lower left")
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()

print(f"ROC-AUC: {roc_auc_score(y_test_gru, y_proba):.4f}")
```

APÊNDICE D3

GRU Otimizado – Código Fonte

4: Preparação para GRU (reshape 3D)

```
print("--- CÉLULA 4: Preparação para GRU ---")

# Converter para float32
X_train = X_train.to_numpy(dtype=np.float32)
X_test = X_test.to_numpy(dtype=np.float32)
y_train = y_train.to_numpy(dtype=np.float32)
y_test = y_test.to_numpy(dtype=np.float32)

# Ajustar para 3D (samples, timesteps, features)
n_timesteps = 1
X_train_gru = X_train.reshape((X_train.shape[0], n_timesteps, X_train.shape[1]))
X_test_gru = X_test.reshape((X_test.shape[0], n_timesteps, X_test.shape[1]))

print(f"Formato X_train_gru: {X_train_gru.shape}, X_test_gru: {X_test_gru.shape}")
```

5: Construção e Treino do Modelo GRU Otimizado (Duas Camadas)

```
print("--- CÉLULA 5: Construção e Treino do Modelo GRU Otimizado ---")

# Construção do modelo otimizado
model_gru_opt = Sequential()
model_gru_opt.add(GRU(units=64, return_sequences=True, input_shape=(n_timesteps, X_train.shape[1])))
model_gru_opt.add(Dropout(0.3))
model_gru_opt.add(GRU(units=32))
model_gru_opt.add(Dropout(0.3))
model_gru_opt.add(Dense(1, activation='sigmoid'))

model_gru_opt.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model_gru_opt.summary())

# Pesos de classe
neg_count = np.count_nonzero(y_train == 0)
pos_count = np.count_nonzero(y_train == 1)
total_count = neg_count + pos_count
class_weights_opt = {0: total_count / (2.0 * neg_count), 1: total_count / (2.0 * pos_count)}
print(f"Pesos de classe: {class_weights_opt}")

# EarlyStopping
early_stopping_opt = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)

# Treino
history_gru_opt = model_gru_opt.fit(
    X_train_gru, y_train,
    epochs=1000,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping_opt],
    verbose=1,
    class_weight=class_weights_opt
)
```

6: Avaliação do Modelo GRU Otimizado

```
print("--- CÉLULA 6: Avaliação do Modelo GRU Otimizado ---")

# Previsões
y_pred_proba_opt = model_gru_opt.predict(X_test_gru, verbose=0).ravel()
y_pred_opt = np.where(y_pred_proba_opt > 0.5, 1, 0)

# Relatório
print("\n--- Relatório de Classificação ---")
print(classification_report(y_test, y_pred_opt))

# Matriz de confusão
cm_opt = confusion_matrix(y_test, y_pred_opt)
plt.figure(figsize=(6,5))
sns.heatmap(cm_opt, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão (GRU Otimizado)')
plt.show()

# ROC-AUC
roc_auc_opt = roc_auc_score(y_test, y_pred_proba_opt)
print(f"ROC-AUC Score: {roc_auc_opt:.4f}")

# PR-AUC
precision_opt, recall_opt, _ = precision_recall_curve(y_test, y_pred_proba_opt)
pr_auc_opt = auc(recall_opt, precision_opt)
print(f"PR-AUC Score: {pr_auc_opt:.4f}")
```

```
plt.figure(figsize=(8,6))
plt.plot(recall_opt, precision_opt, label=f'PR Curve (area={pr_auc_opt:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (GRU Otimizeado)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```

APÊNDICE E

Modelos Deep Learning Não Supervisionado – Códigos Fonte

APÊNDICE E1

VAE – Código Fonte

3. Construção do Modelo Variational Autoencoder (VAE)

```
from tensorflow.keras.layers import Dense, Lambda, Layer
from tensorflow.keras.models import Model
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras import backend as K
import numpy as np
import tensorflow as tf

print("--- CÉLULA 14.1: Construção do Modelo Variational Autoencoder (VAE) ---")

if 'X_train' not in locals():
    print("ERRO: A variável X_train não está definida. Por favor, execute as Células 5.1 e 5.2.")
    raise NameError("Variáveis ausentes.")

# Otimização: Converter os dados para numpy (float32) para maior estabilidade do TF
if hasattr(X_train, 'values'):
    X_train_np = X_train.values.astype('float32')
else:
    X_train_np = np.asarray(X_train, dtype='float32')

n_features = X_train_np.shape[1]

# --- ALTERAÇÕES AQUI: AUMENTAR A CAPACIDADE DO MODELO ---
latent_dim = 10
intermediate_dim = 128
# --- FIM DAS ALTERAÇÕES ---

# --- 1. Definir o VAE como uma Subclasse de Keras.Model (corrigida) ---
class VAE(tf.keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    self.total_loss_tracker = tf.keras.metrics.Mean(name="total_loss")
    self.reconstruction_loss_tracker = tf.keras.metrics.Mean(name="reconstruction_loss")
    self.kl_loss_tracker = tf.keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [self.total_loss_tracker, self.reconstruction_loss_tracker, self.kl_loss_tracker]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(tf.square(data - reconstruction), axis=1)
            kl_loss = -0.5 * tf.reduce_mean(tf.reduce_sum(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var), axis=1))
            total_loss = reconstruction_loss + kl_loss

        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))

        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)

        return {"loss": self.total_loss_tracker.result(), "reconstruction_loss": self.reconstruction_loss_tracker.result(), "kl_loss": self.kl_loss_tracker.result()}

    def call(self, inputs):
        z_mean, z_log_var, z = self.encoder(inputs)
        reconstruction = self.decoder(z)
        return reconstruction

# --- 2. Definir o Encoder e o Decoder ---
encoder_inputs = tf.keras.Input(shape=(n_features,), name="encoder_input")
x = Dense(intermediate_dim, activation='relu')(encoder_inputs)
x = Dense(intermediate_dim, activation='relu')(x)

z_mean = Dense(latent_dim, name="z_mean")(x)
z_log_var = Dense(latent_dim, name="z_log_var")(x)

class Sampling(Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = K.random_normal(shape=(batch, dim))
        return z_mean + K.exp(0.5 * z_log_var) * epsilon

z = Sampling(name="z")([z_mean, z_log_var])
encoder = Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")

decoder_inputs = tf.keras.Input(shape=(latent_dim,), name="z_sampling")
d = Dense(intermediate_dim, activation='relu')(decoder_inputs)
d = Dense(intermediate_dim, activation='relu')(d)
decoder_outputs = Dense(n_features, activation='linear', name="decoder_output")(d)
decoder = Model(decoder_inputs, decoder_outputs, name="decoder")

vae = VAE(encoder, decoder)
```

4. Compilação e Treino do VAE

```
### Célula 14.2: Compilação e Treino do VAE

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

print("--- CÉLULA 14.2: Compilação e Treino do VAE ---")

if 'vae' not in locals():
    print("ERRO: O modelo VAE não está definido. Por favor, execute a Célula 14.1 primeiro.")
    raise NameError("Modelo VAE ausente.")

# Preparar os dados de treino: usar apenas a classe 'Normal'
X_train_normal = X_train_scaled[y_train == 0]

# Compilar o modelo com Adam, usando uma taxa de aprendizagem menor
vae.compile(optimizer=Adam(learning_rate=0.0005))

# Configurar EarlyStopping para monitorar a perda de treino
early_stop = EarlyStopping(
    monitor="loss",
    patience=10,
    restore_best_weights=True,
    verbose=1
)

# Treinar o modelo
print("\nIniciando o treino do modelo VAE (treinando apenas com dados 'normais'...)")
history = vae.fit(
    X_train_normal,
    epochs=100,
    batch_size=256,

    callbacks=[early_stop],
    verbose=1
)

print("\nTreino do modelo VAE concluído com sucesso!")
```

5. Avaliação do Modelo VAE

```
### Célula 14.3: Avaliação do Modelo VAE

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import numpy as np

print("--- CÉLULA 14.3: Avaliação do Modelo VAE ---")

# Calcular erros de reconstrução no treino (apenas classe normal)
reconstructions_train = vae.predict(X_train_scaled[y_train == 0], verbose=0)
reconstruction_errors_train = np.mean(np.square(X_train_scaled[y_train == 0] - reconstructions_train), axis=1)

# Calcular erros de reconstrução no teste
reconstructions_test = vae.predict(X_test_scaled, verbose=0)
reconstruction_errors_test = np.mean(np.square(X_test_scaled - reconstructions_test), axis=1)

# Tratar NaN ou Inf nos erros de reconstrução
mask = ~np.isnan(reconstruction_errors_test) & ~np.isinf(reconstruction_errors_test)
y_test_clean = y_test[mask]
reconstruction_errors_test_clean = reconstruction_errors_test[mask]
print(f"Valores NaN/Inf removidos do conjunto de teste: {len(y_test) - len(y_test_clean)}")

# Definir threshold a partir do 99.9º percentil dos erros de treino
threshold = np.percentile(reconstruction_errors_train, 99.9)
print(f"\nThreshold definido no 99.9º percentil: {threshold:.4f}")

# Fazer previsões com o threshold
y_pred = (reconstruction_errors_test_clean > threshold).astype(int)
```

```

# Relatório de Classificação
print("\n--- Relatório de Classificação (VAE) ---")
print(classification_report(y_test_clean, y_pred, zero_division=0))

# Matriz de Confusão
cm = confusion_matrix(y_test_clean, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão (VAE)')
plt.show()

# Métricas ROC-AUC e PR-AUC
roc_auc_vae = roc_auc_score(y_test_clean, reconstruction_errors_test_clean)
precision_vae, recall_vae, _ = precision_recall_curve(y_test_clean, reconstruction_errors_test_clean)
pr_auc_vae = auc(recall_vae, precision_vae)

print(f"ROC-AUC Score (VAE): {roc_auc_vae:.4f}")
print(f"PR-AUC Score (VAE): {pr_auc_vae:.4f}")

# Curva Precision-Recall
plt.figure(figsize=(8, 6))
plt.plot(recall_vae, precision_vae, label=f'Curva PR (área={pr_auc_vae:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (VAE)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()

# Avaliação com múltiplos thresholds
thresholds_to_test = [95, 98, 99, 99.5, 99.9]
for t in thresholds_to_test:
    threshold_t = np.percentile(reconstruction_errors_train, t)
    y_pred_t = (reconstruction_errors_test_clean > threshold_t).astype(int)
    report_t = classification_report(y_test_clean, y_pred_t, zero_division=0)
    print(f"\n--- Resultados para o Threshold no {t}% Percentil ({threshold_t:.4f}) ---")
    print(report_t)

```

APÊNDICE E2

DeepLog – Código Fonte

3. Amostragem Equilibrada para Treino

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

print("--- CÉLULA 5.3: Amostragem para Aumentar o Conjunto de Treino (100K Normais + Anomalias) ---")

if 'X_train' not in locals() or 'y_train' not in locals():
    print("ERRO: As variáveis X_train e/ou y_train não estão definidas. Por favor, execute as Células 5.1 e 5.2 primeiro.")
    raise NameError("Variáveis de treino ausentes.")

# 1. Separar a classe de anomalias da classe normal no conjunto de treino
X_train_normal = X_train[y_train == 0]
X_train_anomaly = X_train[y_train == 1]
y_train_normal = y_train[y_train == 0]
y_train_anomaly = y_train[y_train == 1]

# 2. Definir o novo tamanho do conjunto de dados
num_anomalies = len(X_train_anomaly)
# NOVO VALOR: Tentar 100.000 amostras normais
num_normal_samples = 100000
if len(X_train_normal) < num_normal_samples:
    num_normal_samples = len(X_train_normal)

# 3. Amostrar o novo conjunto de dados
X_train_normal_sampled = X_train_normal.sample(n=num_normal_samples, random_state=42)
y_train_normal_sampled = y_train_normal.sample(n=num_normal_samples, random_state=42)

# Juntar os dados de anomalia e a amostra de dados normais
X_train_balanced = pd.concat([X_train_normal_sampled, X_train_anomaly])
y_train_balanced = pd.concat([y_train_normal_sampled, y_train_anomaly])

# Misturar o novo conjunto de dados
shuffled_indices = np.random.permutation(X_train_balanced.index)
X_train_final = X_train_balanced.loc[shuffled_indices].reset_index(drop=True)
y_train_final = y_train_balanced.loc[shuffled_indices].reset_index(drop=True)

# Atualizar as variáveis de treino
X_train = X_train_final
y_train = y_train_final

print(f"\nNovo conjunto de treino (Amostragem: {num_normal_samples} Normais + {num_anomalies} Anomalias): {X_train.shape[0]} amostras.")
print(f"  Anomalias: {len(y_train[y_train==1])}")
print(f"  Normais: {len(y_train[y_train==0])}")
print(f"  Rácio Aproximado: 1 Anomalia para {round(len(y_train[y_train==0])/len(y_train[y_train==1]))} Normais")
```

4. Preparação de Dados para Deeplog (Reshape para Sequências)

```
4]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

print("--- CÉLULA 16.1: Preparação de Dados para Deeplog (Reshape para Sequências - Usando Amostra Total) ---")

if 'X_train' not in locals() or 'y_train' not in locals():
    print("ERRO: As variáveis de treino (X_train, y_train) não estão definidas. Por favor, execute as Células 5.1 e 5.2.")
    raise NameError("Variáveis de treino ausentes.")

# == PATCH: sane dtypes para TensorFlow ==
def _to_float32_matrix(X):
    if isinstance(X, pd.DataFrame):
        X = X.replace([True: 1.0, False: 0.0]).apply(pd.to_numeric, errors='coerce').fillna(0.0).to_numpy(dtype=np.float32)
    else:
        X = np.array(X, dtype=np.float32)
    if X.ndim == 1:
        X = X.reshape(-1, 1)
    return X

X_train_np = _to_float32_matrix(X_train)
X_test_np = _to_float32_matrix(X_test)
y_train_np = np.asarray(y_train, dtype=np.float32)
y_test_np = np.asarray(y_test, dtype=np.float32)

# == FIM PATCH ==

# Ajuste para 3D (n_amostras, n_timesteps, n_features)
n_timesteps = 10
n_features = X_train_np.shape[1]
```

```

# Para X_train (usa o total de 50.190 amostras)
train_size_adj = (X_train_np.shape[0] // n_timesteps) * n_timesteps
X_train_np_full = X_train_np[:train_size_adj]
y_train_np_full = y_train_np[:train_size_adj]

X_train_deeplog = X_train_np_full.reshape((-1, n_timesteps, n_features))
y_train_deeplog_target = y_train_np_full.reshape((-1, n_timesteps))

# Para X_test (usa o total de 450.060 amostras)
test_size_adj = (X_test_np.shape[0] // n_timesteps) * n_timesteps
X_test_np_full = X_test_np[:test_size_adj]
y_test_np_full = y_test_np[:test_size_adj]

X_test_deeplog = X_test_np_full.reshape((-1, n_timesteps, n_features))
y_test_deeplog_target = y_test_np_full.reshape((-1, n_timesteps))

# Variável plana do y_test (para métricas)
y_test_deeplog_flat = y_test_np_full[:,X_test_np_full.shape[0]].reshape(-1)

print(f"Total de amostras de TREINO (Após Amostragem 5.3): {X_train_np_full.shape[0]} amostras. ({X_train_deeplog.shape[0]} sequências)")
print(f"Total de amostras de TESTE (Total Original 20%): {X_test_np_full.shape[0]} amostras. ({X_test_deeplog.shape[0]} sequências)")
print(f"Formato de X_train para Deeplog: {X_train_deeplog.shape}, dtype={X_train_deeplog.dtype}")
print(f"Formato de X_test para Deeplog: {X_test_deeplog.shape}, dtype={X_test_deeplog.dtype}")

```

5. Construção do Modelo Deeplog

```

import tensorflow as tf
from tensorflow.keras.layers import GRU, Dense, Dropout, Attention, TimeDistributed
from tensorflow.keras.models import Model
from tensorflow.keras import Input

print("--- CÉLULA 16.2: Construção do Modelo Deeplog ---")

if 'X_train_deeplog' not in locals():
    print("ERRO: O DataFrame 'X_train_deeplog' não está definido. Por favor, execute a Célula 16.1 primeiro.")
    raise NameError("Variáveis ausentes.")

# 1. Obter o formato de entrada para o modelo
input_shape = (X_train_deeplog.shape[1], X_train_deeplog.shape[2])
n_features = input_shape[1]

# 2. Construir o Modelo Deeplog (com GRU)
inputs = Input(shape=input_shape)

# Camada GRU (o coração do modelo recorrente), deve retornar sequências
gru_out = GRU(units=128, return_sequences=True, name="gru_layer")(inputs)

# Adicionar uma segunda camada GRU para aumentar a profundidade do modelo
gru_out = GRU(units=64, return_sequences=True, name="gru_layer_2")(gru_out)

# A camada de atenção pode ser usada para focar, mas a sua saída tem que voltar a ser uma sequência para a reconstrução
# Uma maneira é usar a atenção para ponderar os 'timesteps' e depois usar uma camada densa para reconstruir cada timestep.
# Para manter a arquitetura de autoencoder, a camada de saída deve ter a mesma forma que a entrada.
# Usamos TimeDistributed para aplicar a camada Dense a cada 'timestep' da sequência
outputs = TimeDistributed(Dense(units=n_features, activation='linear', name="output_layer"))(gru_out)

model_deeplog = Model(inputs, outputs)

```

6. Compilação e Treino do Modelo Deeplog

```

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
import tensorflow as tf

print("--- CÉLULA 16.3: Compilação e Treino do Modelo Deeplog ---")

if 'model_deeplog' not in locals():
    print("ERRO: O modelo Deeplog não está definido. Por favor, execute a Célula 16.2 primeiro.")
    raise NameError("Modelo não construído.")

# 1. Compilar o modelo
model_deeplog.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse' # Usamos Mean Squared Error para otimizar a reconstrução
)

print("\nModelo Deeplog compilado com sucesso!")

# 2. Configurar EarlyStopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=5, restore_best_weights=True, verbose=1
)

# 3. Treinar o modelo
# Agora o target é o próprio input, e o modelo irá reconstruir cada passo de tempo na sequência
print("\nIniciando o treino do modelo Deeplog (o treino na GPU pode ser muito rápido)...")
history_deeplog = model_deeplog.fit(
    X_train_deeplog,
    X_train_deeplog, # O target é o próprio input (autoencoder)
    epochs=100,
    batch_size=256,
    validation_split=0.2,

    callbacks=[early_stop],
    verbose=1
)

```

7. Avaliação do Modelo Deeplog

```
import tensorflow as tf
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

print("--- CÉLULA 16.4: Avaliação do Modelo Deeplog ---")

if 'model_deeplog' not in locals() or 'X_test_deeplog' not in locals():
    print("ERRO: O modelo Deeplog ou os dados de teste não estão definidos. Por favor, execute as células anteriores.")
    raise NameError("Variáveis ausentes.")

# 1. Calcular o erro de reconstrução no conjunto de teste
print("\nCalculando o erro de reconstrução no conjunto de teste...")
reconstructions_test = model_deeplog.predict(X_test_deeplog, verbose=0)

# Aplanar os arrays para calcular o erro
n_features = X_test_deeplog.shape[2]
X_test_deeplog_flat = X_test_deeplog.reshape(-1, n_features)
reconstructions_test_flat = reconstructions_test.reshape(-1, n_features)

# O erro de reconstrução é a distância quadrática entre o original e o reconstruído
reconstruction_error = np.mean(np.square(X_test_deeplog_flat - reconstructions_test_flat), axis=1)

# 2. Definir um limiar de anomalia com base nos erros de treino
# Treinar o modelo em 100% dos dados e depois obter os erros de reconstrução
X_train_reconstructed = model_deeplog.predict(X_train_deeplog, verbose=0)
X_train_deeplog_flat = X_train_deeplog.reshape(-1, n_features)
X_train_reconstructed_flat = X_train_reconstructed.reshape(-1, n_features)
reconstruction_error_train = np.mean(np.square(X_train_deeplog_flat - X_train_reconstructed_flat), axis=1)

# Usamos um percentil para definir o limiar (pode ser ajustado)
threshold = np.percentile(reconstruction_error_train, 99.8)
print(f"Limiar de anomalia (erro de reconstrução): {threshold:.4f}")

# 3. Classificar como Anomalia (1) se o erro for maior que o limiar
y_pred_deeplog = (reconstruction_error > threshold).astype(int)

# 4. Usar a variável y_test_deeplog_flat (ajustada na Célula 16.1)
y_test_deeplog_flat_final = y_test_deeplog_flat[:len(y_pred_deeplog)]

# 5. Avaliar o Desempenho do Modelo
print("\n--- Relatório de Classificação do Modelo Deeplog ---")
print(classification_report(y_test_deeplog_flat_final, y_pred_deeplog, zero_division=0))

print("\n--- Matriz de Confusão do Modelo Deeplog ---")
cm_deeplog = confusion_matrix(y_test_deeplog_flat_final, y_pred_deeplog)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_deeplog, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão (Deeplog)')
plt.show()

# 6. ROC-AUC Score e PR-AUC
roc_auc_deeplog = roc_auc_score(y_test_deeplog_flat_final, reconstruction_error)
print(f"\nROC-AUC Score (Deeplog): {roc_auc_deeplog:.4f}")

precision_deeplog, recall_deeplog, _ = precision_recall_curve(y_test_deeplog_flat_final, reconstruction_error)
pr_auc_deeplog = auc(recall_deeplog, precision_deeplog)
print(f"PR-AUC Score (Deeplog): {pr_auc_deeplog:.4f}")

plt.figure(figsize=(8, 6))
plt.plot(recall_deeplog, precision_deeplog, label=f'PR Curve Deeplog (area = {pr_auc_deeplog:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (Deeplog)')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()
```

APÊNDICE F

Simulação de Cenários de Ataque – Código Fonte

APÊNDICE F1

Simulação de DDoS – Código Fonte

Simulação DDoS

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from datetime import datetime, timedelta

print("--- Cenário 1: Simulação de Ataque DDoS/DoS (v2 - Corrigido) ---")
print("\n--- PASSO 1: Preparação do Dataset ---")

# Carregar o dataset preparado
try:
    df_model = pd.read_csv('final_preprocessed_logs_target_severity.csv', low_memory=False)
    print("Dataset 'final_preprocessed_logs_target_severity.csv' carregado com sucesso.")
except FileNotFoundError:
    print("ERRO: O arquivo 'final_preprocessed_logs_target_severity.csv' não foi encontrado. Certifique-se de que está no mesmo diretório.")
    raise

# Separar features (X) e target (y)
target_column_name = 'Is_Anomaly_Severity'
if target_column_name in df_model.columns:
    X_full = df_model.drop(columns=[target_column_name])
    y_full = df_model[target_column_name]
    print(f"\nFeatures (X_full) e Target (y_full) separadas. Formato de X_full: {X_full.shape}")
else:
    print(f"ERRO: A coluna alvo '{target_column_name}' não foi encontrada.")
    raise KeyError

# Adicionar colunas necessárias para a simulação, se não existirem
np.random.seed(42)
if 'Destination IP' not in X_full.columns:
    print("AVISO: A coluna 'Destination IP' não foi encontrada. Criando uma coluna simulada para a simulação.")
    num_ips = 20
    X_full['Destination IP'] = [f'10.0.0.{i % num_ips}' for i in range(len(X_full))]

if 'Source IP' not in X_full.columns:
    print("AVISO: A coluna 'Source IP' não foi encontrada. Criando uma coluna simulada para a simulação.")
    X_full['Source IP'] = [f'192.168.{i % 10}.{i % 200}' for i in range(len(X_full))]

if 'timestamp' not in X_full.columns:
    print("AVISO: A coluna 'timestamp' não foi encontrada. Criando uma coluna simulada para a simulação.")
    start_time = datetime(2025, 1, 1, 0, 0, 0)
    time_deltas = [timedelta(seconds=i) for i in range(len(X_full))]
    X_full['timestamp'] = [start_time + delta for delta in time_deltas]

# Dividir em conjuntos de treino e teste (80/20) mantendo a estratificação
X_train, X_test, y_train, y_test = train_test_split(X_full, y_full, test_size=0.20, random_state=42, stratify=y_full)
print("Dados divididos em treino e teste (20% para teste).")

# Escalar as features numéricas
numerical_cols = X_train.select_dtypes(include=[float64, 'int64']).columns
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
scaler.fit(X_train_scaled[numerical_cols])
X_train_scaled[numerical_cols] = scaler.transform(X_train_scaled[numerical_cols])
X_test_scaled[numerical_cols] = scaler.transform(X_test_scaled[numerical_cols])
print("\nFeatures numéricas escaladas.")

# Remover as colunas de IP e timestamp dos conjuntos que vão para o modelo
X_train_model = X_train_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp'])
X_test_model = X_test_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp'])
print("Colunas de IP e timestamp removidas dos conjuntos para treino e predição do modelo.")
print("--- Preparação do dataset concluída. ---")

print("\n--- PASSO 2: Treino e Avaliação do Modelo Otimizado ---")
# Calcular o scale_pos_weight para o desbalanceamento de classes
neg_count = y_train.value_counts()[0]
pos_count = y_train.value_counts()[1]
scale_pos_weight_value = neg_count / pos_count
best_params = {'subsample': 1.0, 'n_estimators': 500, 'min_child_weight': 1, 'max_depth': 7, 'learning_rate': 0.1, 'gamma': 0.1, 'colsample_bytree': 0.6}
best_xgb_model = xgb.XGBClassifier(
    objective='binary:logistic',
    scale_pos_weight=scale_pos_weight_value,
    random_state=42,
    n_jobs=-1,
    eval_metric='logloss',
    **best_params
)
print("Iniciando o treino do modelo otimizado...")
best_xgb_model.fit(X_train_model, y_train)
print("Treino concluído com sucesso!")

print("\n--- PASSO 3: Simulação de Ataque DDoS/DoS e Análise ---")
print("Analisando o desempenho do modelo otimizado com dados injetados.")

# --- Definir a função de simulação ---
def create_ddos_attack_events(X_test, n_attack_events, target_ip, random_state):
    """Cria eventos de ataque sintéticos com base em uma amostra aleatória do X_test."""
    np.random.seed(random_state)
    attack_events_df = X_test.sample(n=n_attack_events, replace=True, random_state=random_state)

```

```

# Modificar os campos para simular o ataque DDoS
numerical_cols = X_test.select_dtypes(include=['float64', 'int64']).columns
attack_events_df[numerical_cols] = attack_events_df[numerical_cols].apply(
    lambda x: x * np.random.uniform(50, 100) if pd.api.types.is_numeric_dtype(x) else x)

# Simular o padrão DDoS: vários IPs de origem contra um IP de destino
attack_events_df['Destination IP'] = target_ip

# Adicionar uma coluna de timestamp concentrada em um curto intervalo
if 'timestamp' in attack_events_df.columns:
    start_time_attack = attack_events_df['timestamp'].iloc[0]
    time_deltas_attack = [timedelta(milliseconds=np.random.randint(0, 50)) for _ in range(n_attack_events)]
    attack_events_df['timestamp'] = [start_time_attack + delta for delta in time_deltas_attack]

return attack_events_df

def simulate_and_evaluate_ddos(X_test, y_test, best_xgb_model, scaler, numerical_cols, intensity_factor, random_state):
    """Executa a simulação e avalia o modelo."""
    n_inj = int(len(X_test) * intensity_factor)

    target_ip = X_test['Destination IP'].sample(1, random_state=random_state).values[0]

    attack_events = create_ddos_attack_events(X_test, n_inj, target_ip, random_state)

    y_attack = pd.Series([1] * len(attack_events), index=attack_events.index)

    X_simulated = pd.concat([X_test, attack_events], ignore_index=True)
    y_simulated = pd.concat([y_test, y_attack], ignore_index=True)

    X_simulated_scaled = X_simulated.copy()
    X_simulated_scaled[numerical_cols] = scaler.transform(X_simulated_scaled[numerical_cols])
    X_simulated_features = X_simulated_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp'], errors='ignore')

    y_pred_sim = best_xgb_model.predict(X_simulated_features)
    y_pred_proba_sim = best_xgb_model.predict_proba(X_simulated_features)[:, 1]

    cm = confusion_matrix(y_simulated, y_pred_sim)
    report = classification_report(y_simulated, y_pred_sim, output_dict=True, zero_division=0)
    roc_auc = roc_auc_score(y_simulated, y_pred_proba_sim)
    pr_auc = auc(precision_recall_curve(y_simulated, y_pred_proba_sim)[1],
                precision_recall_curve(y_simulated, y_pred_proba_sim)[0])

    return cm, report, roc_auc, pr_auc, n_inj, y_simulated.sum(), y_pred_sim, y_simulated, y_pred_proba_sim

# --- Bloco principal para executar a simulação em 3 intensidades ---
intensities = {'low': 0.001, 'medium': 0.005, 'high': 0.01}
results_ddos = {}
numerical_cols = X_test.select_dtypes(include=['float64', 'int64']).columns

for level, factor in intensities.items():
    print(f"\n-> Simulação com intensidade: {level.upper()} (Fator: {factor})")

    cm, report, roc_auc, pr_auc, n_inj, total_anomalies, y_pred_sim, y_simulated, y_pred_proba_sim = simulate_and_evaluate_ddos(
        X_test, y_test, best_xgb_model, scaler, numerical_cols, factor, 42)

    results_ddos[level] = {
        'confusion_matrix': cm,
        'report': report,
        'roc_auc': roc_auc,
        'pr_auc': pr_auc,
        'anomalias_injetadas': n_inj
    }

    print(f"Total de eventos injetados: {n_inj}")
    print(f"Total de anomalias reais (original + injetadas): {total_anomalies}")
    print("\n--- Relatório de Classificação ---")
    print(classification_report(y_simulated, y_pred_sim, zero_division=0))

# Visualizar a matriz de confusão
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title(f'Matriz de Confusão DDoS - {level.upper()}')
plt.show()

# Visualizar a curva PR-AUC
precision, recall, _ = precision_recall_curve(y_simulated, y_pred_proba_sim)
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'Curva PR DDoS {level.upper()} (area = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title(f'Curva Precision-Recall DDoS - {level.upper()}')
plt.legend(locs='lower left')
plt.grid(True)
plt.show()

# Apresentar o resumo dos resultados em uma tabela
results_df_ddos = pd.DataFrame([
    {
        'Intensidade': level,
        'Precision': results_ddos[level]['report']['1']['precision'],
        'Recall': results_ddos[level]['report']['1']['recall'],
        'F1-score': results_ddos[level]['report']['1']['f1-score'],
        'FP': results_ddos[level]['confusion_matrix'][0, 1],
        'FN': results_ddos[level]['confusion_matrix'][1, 0],
        'ROC-AUC': results_ddos[level]['roc_auc'],
        'PR-AUC': results_ddos[level]['pr_auc']
    } for level in intensities.keys()
])

```

APÊNDICE F2

Simulação de Exfiltração de Dados – Código Fonte

Simulação de Exfiltração de Dados

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from datetime import datetime, timedelta

print("--- Cenário 3: Simulação de Ataque de Exfiltração de Dados ---")
print("\n--- PASSO 1: Preparação do Dataset ---")

# Carregar o dataset preparado
try:
    df_model = pd.read_csv('final_preprocessed_logs_target_severity.csv', low_memory=False)
    print("Dataset 'final_preprocessed_logs_target_severity.csv' carregado com sucesso.")
except FileNotFoundError:
    print("ERRO: O arquivo 'final_preprocessed_logs_target_severity.csv' não foi encontrado. Certifique-se de que está no mesmo diretório.")
    raise

# Separar features (X) e target (y)
target_column_name = 'Is_Anomaly_Severity'
if target_column_name in df_model.columns:
    X_full = df_model.drop(columns=[target_column_name])
    y_full = df_model[target_column_name]
    print(f"\nFeatures (X_full) e Target (y_full) separadas. Formato de X_full: {X_full.shape}")
else:
    print(f"ERRO: A coluna alvo '{target_column_name}' não foi encontrada.")
    raise KeyError

# Adicionar colunas necessárias para a simulação, se não existirem
np.random.seed(42)
if 'Destination IP' not in X_full.columns:
    print("AVISO: A coluna 'Destination IP' não foi encontrada. Criando uma coluna simulada para a simulação.")
    num_ips = 20
    X_full['Destination IP'] = [f'10.0.0.{i % num_ips}' for i in range(len(X_full))]

if 'Source IP' not in X_full.columns:
    print("AVISO: A coluna 'Source IP' não foi encontrada. Criando uma coluna simulada para a simulação.")
    X_full['Source IP'] = [f'192.168.{i % 10}.{i % 200}' for i in range(len(X_full))]

if 'timestamp' not in X_full.columns:
    print("AVISO: A coluna 'timestamp' não foi encontrada. Criando uma coluna simulada para a simulação.")
    start_time = datetime(2025, 1, 1, 0, 0, 0)
    time_deltas = [timedelta(seconds=i) for i in range(len(X_full))]
    X_full['timestamp'] = [start_time + delta for delta in time_deltas]

# Dividir em conjuntos de treino e teste (80/20) mantendo a estratificação
X_train, X_test, y_train, y_test = train_test_split(X_full, y_full, test_size=0.20, random_state=42, stratify=y_full)
print("Dados divididos em treino e teste (20% para teste).")

# Escalar as features numéricas
numerical_cols = X_train.select_dtypes(include=[float64, 'int64']).columns
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
scaler.fit(X_train_scaled[numerical_cols])
X_train_scaled[numerical_cols] = scaler.transform(X_train_scaled[numerical_cols])
X_test_scaled[numerical_cols] = scaler.transform(X_test_scaled[numerical_cols])
print("\nFeatures numéricas escaladas.")

# Remover as colunas de IP e timestamp dos conjuntos que vão para o modelo
X_train_model = X_train_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp'])
X_test_model = X_test_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp'])
print("Colunas de IP e timestamp removidas dos conjuntos para treino e predição do modelo.")
print("--- Preparação do dataset concluída. ---")

print("\n--- PASSO 2: Treino e Avaliação do Modelo Otimizado ---")
# Calcular o scale_pos_weight para o desbalanceamento de classes
neg_count = y_train.value_counts()[0]
pos_count = y_train.value_counts()[1]
scale_pos_weight_value = neg_count / pos_count
best_params = {'subsample': 1.0, 'n_estimators': 500, 'min_child_weight': 1, 'max_depth': 7, 'learning_rate': 0.1, 'gamma': 0.1, 'colsample_bytree': 0.6}
best_xgb_model = xgb.XGBClassifier(
    objective='binary:logistic',
    scale_pos_weight=scale_pos_weight_value,
    random_states=42,
    n_jobs=-1,
    eval_metric='logloss',
    **best_params
)
print("Iniciando o treino do modelo otimizado...")
best_xgb_model.fit(X_train_model, y_train)
print("Treino concluído com sucesso!")

print("\n--- PASSO 3: Simulação de Ataque de Exfiltração de Dados e Análise ---")
print("Analisando o desempenho do modelo otimizado com dados injetados.")

# --- Definir a função de simulação de exfiltração de dados ---
def simulate_exfiltration_attack(X_test, y_test, scaler, numerical_cols, intensity_factor, random_state=42):
    """
    Simula um ataque de exfiltração de dados.
    """
```

```

X_sim_base = X_test.copy()
y_sim_base = y_test.copy()

n_attack_events = int(len(X_sim_base) * intensity_factor)
np.random.seed(random_state)
attack_events_df = X_sim_base.sample(n=n_attack_events, replace=True, random_state=random_state)

# Aumentar drasticamente os valores de bytes para simular a exfiltração
for col in ['Server Outbound Bytes', 'Sent Bytes']:
    if col in attack_events_df.columns:
        attack_events_df[col] = attack_events_df[col].apply(
            lambda x: x * np.random.uniform(100, 1000) if pd.api.types.is_numeric_dtype(x) else x
        )

y_attack = pd.Series([1] * len(attack_events_df), index=attack_events_df.index)

X_simulated = pd.concat([X_test, attack_events_df], ignore_index=True)
y_simulated = pd.concat([y_test, y_attack], ignore_index=True)

X_simulated_scaled = X_simulated.copy()
X_simulated_scaled[numerical_cols] = scaler.transform(X_simulated_scaled[numerical_cols])
X_simulated_features = X_simulated_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp'], errors='ignore')

return X_simulated_features, y_simulated, n_attack_events

# --- Bloco principal para executar a simulação em 3 intensidades ---
intensities = {'low': 0.001, 'medium': 0.005, 'high': 0.01}
results_exfiltration = {}
numerical_cols = X_test.select_dtypes(include=['float64', 'int64']).columns

for level, factor in intensities.items():
    print(f"\n-> Simulação com intensidade: {level.upper()} (Fator: {factor})")

    X_sim_exfiltration, y_sim_exfiltration, n_inj = simulate_exfiltration_attack(
        X_test, y_test, scaler, numerical_cols, factor, 42)

    y_pred_sim = best_xgb_model.predict(X_sim_exfiltration)
    y_pred_proba_sim = best_xgb_model.predict_proba(X_sim_exfiltration)[:, 1]

    cm = confusion_matrix(y_sim_exfiltration, y_pred_sim)
    report = classification_report(y_sim_exfiltration, y_pred_sim, output_dict=True, zero_division=0)
    roc_auc = roc_auc_score(y_sim_exfiltration, y_pred_proba_sim)
    pr_auc = auc(precision_recall_curve(y_sim_exfiltration, y_pred_proba_sim)[1],
                precision_recall_curve(y_sim_exfiltration, y_pred_proba_sim)[0])

    results_exfiltration[level] = {
        'confusion_matrix': cm,
        'report': report,
        'roc_auc': roc_auc,
        'pr_auc': pr_auc,
        'anomalias_injetadas': n_inj
    }

    print(f"\nTotal de eventos injetados: {n_inj}")
    print(f"\nTotal de anomalias reais (original + injetadas): {y_sim_exfiltration.sum()}")
    print(f"\n--- Relatório de Classificação ---")
    print(classification_report(y_sim_exfiltration, y_pred_sim, zero_division=0))

    # Visualizar a matriz de confusão
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
    plt.xlabel('Previsão')
    plt.ylabel('Real')
    plt.title(f'Matriz de Confusão Exfiltração de Dados - {level.upper()}')
    plt.show()

    # Visualizar a curva PR-AUC
    precision, recall, _ = precision_recall_curve(y_sim_exfiltration, y_pred_proba_sim)
    plt.figure(figsize=(8, 6))

    plt.plot(recall, precision, label=f'Curva PR Exfiltração de Dados {level.upper()} (area = {pr_auc:.2f})')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Curva Precision-Recall Exfiltração de Dados - {level.upper()}')
    plt.legend(loc='lower left')
    plt.grid(True)
    plt.show()

# Apresentar o resumo dos resultados em uma tabela
results_df_exfiltration = pd.DataFrame([
    {
        'Intensidade': level,
        'Precision': results_exfiltration[level]['report']['1']['precision'],
        'Recall': results_exfiltration[level]['report']['1']['recall'],
        'F1-score': results_exfiltration[level]['report']['1']['f1-score'],
        'FP': results_exfiltration[level]['confusion_matrix'][0, 1],
        'FN': results_exfiltration[level]['confusion_matrix'][1, 0],
        'ROC-AUC': results_exfiltration[level]['roc_auc'],
        'PR-AUC': results_exfiltration[level]['pr_auc']
    } for level in intensities.keys()
])

```

APÊNDICE F3

Simulação de SQL Injection - Código Fonte

Simulação de SQL Injection (Web Exploits)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, precision_recall_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from datetime import datetime, timedelta

print("---- Cenário 4: Simulação de Ataque de Malware ----")
print("\n--- PASSO 1: Preparação do Dataset ---")

# Carregar o dataset preparado
try:
    df_model = pd.read_csv('final_preprocessed_logs_target_severity.csv', low_memory=False)
    print("Dataset 'final_preprocessed_logs_target_severity.csv' carregado com sucesso.")
except FileNotFoundError:
    print("ERRO: O arquivo 'final_preprocessed_logs_target_severity.csv' não foi encontrado. Certifique-se de que está no mesmo diretório.")
    raise

# Separar features (X) e target (y)
target_column_name = 'Is_Anomaly_Severity'
if target_column_name in df_model.columns:
    X_full = df_model.drop(columns=[target_column_name])
    y_full = df_model[target_column_name]
    print(f"Features (X_full) e Target (y_full) separadas. Formato de X_full: {X_full.shape}")
else:
    print(f"ERRO: A coluna alvo '{target_column_name}' não foi encontrada.")
    raise KeyError

# Adicionar colunas necessárias para a simulação, se não existirem
np.random.seed(42)
if 'Destination IP' not in X_full.columns:
    print("AVISO: A coluna 'Destination IP' não foi encontrada. Criando uma coluna simulada para a simulação.")
    num_ips = 20
    X_full['Destination IP'] = [f'10.0.0.{i % num_ips}' for i in range(len(X_full))]

if 'Source IP' not in X_full.columns:
    print("AVISO: A coluna 'Source IP' não foi encontrada. Criando uma coluna simulada para a simulação.")
    X_full['Source IP'] = [f'192.168.{i % 10}.{i % 200}' for i in range(len(X_full))]

if 'timestamp' not in X_full.columns:
    print("AVISO: A coluna 'timestamp' não foi encontrada. Criando uma coluna simulada para a simulação.")
    start_time = datetime(2025, 1, 1, 0, 0, 0)
    time_deltas = [timedelta(seconds=i) for i in range(len(X_full))]
    X_full['timestamp'] = [start_time + delta for delta in time_deltas]

# Adicionar colunas categóricas necessárias para a simulação de malware/web exploits, se não existirem
if 'App Protocol' not in X_full.columns:
    X_full['App Protocol'] = np.random.choice(['HTTP', 'HTTPS', 'DNS'], size=len(X_full))

if 'Service' not in X_full.columns:
    X_full['Service'] = np.random.choice(['Web', 'Email', 'File'], size=len(X_full))

# Dividir em conjuntos de treino e teste (80/20) mantendo a estratificação
X_train, X_test, y_train, y_test = train_test_split(X_full, y_full, test_size=0.20, random_state=42, stratify=y_full)
print("Dados divididos em treino e teste (20% para teste).")

# Escalar as features numéricas
numerical_cols = X_train.select_dtypes(include=[float64, 'int64']).columns
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

scaler.fit(X_train_scaled[numerical_cols])
X_train_scaled[numerical_cols] = scaler.transform(X_train_scaled[numerical_cols])
X_test_scaled[numerical_cols] = scaler.transform(X_test_scaled[numerical_cols])
print("\nFeatures numéricas escaladas.")

# Remover as colunas de IP e timestamp dos conjuntos que vão para o modelo
# NOTA: Aqui removemos também as colunas de Serviço/Protocolo se existirem no modelo, pois o modelo original não foi treinado com elas.
X_train_model = X_train_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp', 'App Protocol', 'Service'], errors='ignore')
X_test_model = X_test_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp', 'App Protocol', 'Service'], errors='ignore')
print("Colunas de IP, timestamp e serviço/protocolo removidas dos conjuntos para treino e predição do modelo.")
print("---- Preparação do dataset concluída. ----")

print("\n--- PASSO 2: Treino e Avaliação do Modelo Otimizado ---")
# Calcular o scale_pos_weight para o desbalanceamento de classes
neg_count = y_train.value_counts()[0]
pos_count = y_train.value_counts()[1]
scale_pos_weight_value = neg_count / pos_count
best_params = {'subsample': 1.0, 'n_estimators': 500, 'min_child_weight': 1, 'max_depth': 7, 'learning_rate': 0.1, 'gamma': 0.1, 'colsample_bytree': 0.6}
best_xgb_model = xgb.XGBClassifier(
    objectives='binary:logistic',
    scale_pos_weight=scale_pos_weight_value,
    random_state=42,
    n_jobs=-1,
    eval_metric='logloss',
    **best_params
)
print("Iniciando o treino do modelo otimizado...")
best_xgb_model.fit(X_train_model, y_train)
print("Treino concluído com sucesso!\n")
```

```

print("--- PASSO 3: Simulação de Ataque de Malware e Análise ---")
print("Analisando o desempenho do modelo otimizado com dados injetados.")

# --- Definir a função de simulação de malware ---
def simulate_malware_attack(X_test, y_test, scaler, numerical_cols, intensity_factor, random_state=42):
    """
    Simula um ataque de malware.
    """
    X_sim_base = X_test.copy()
    y_sim_base = y_test.copy()

    n_attack_events = int(len(X_sim_base) * intensity_factor)
    np.random.seed(random_state)
    attack_events_df = X_sim_base.sample(n=n_attack_events, replace=True, random_state=random_state)

    # Simular o padrão de malware: valores anômalos em colunas que o modelo vê
    # NOTA: Aqui estamos a assumir que o modelo original NÃO foi treinado com estas colunas
    # A sua tese deve focar-se na falha do modelo em detetar estas anomalias.

    # Aumentar drasticamente os valores de bytes para simular a exfiltração
    for col in ['Server Outbound Bytes', 'Sent Bytes']:
        if col in attack_events_df.columns:
            attack_events_df[col] = attack_events_df[col].apply(
                lambda x: x * np.random.uniform(100, 1000) if pd.api.types.is_numeric_dtype(x) else x)

    y_attack = pd.Series([1] * len(attack_events_df), index=attack_events_df.index)

    X_simulated = pd.concat([X_test, attack_events_df], ignore_index=True)
    y_simulated = pd.concat([y_test, y_attack], ignore_index=True)

    X_simulated_scaled = X_simulated.copy()
    X_simulated_scaled[numerical_cols] = scaler.transform(X_simulated_scaled[numerical_cols])
    X_simulated_features = X_simulated_scaled.drop(columns=['Destination IP', 'Source IP', 'timestamp', 'App Protocol', 'Service'], errors='ignore')

    return X_simulated_features, y_simulated, n_attack_events

# --- Bloco principal para executar a simulação em 3 intensidades ---
intensities = {'low': 0.001, 'medium': 0.005, 'high': 0.01}
results_malware = {}
numerical_cols = X_test.select_dtypes(include=['float64', 'int64']).columns

for level, factor in intensities.items():
    print(f"\n-> Simulação com intensidade: {level.upper()} (Fator: {factor})")

    X_sim, y_sim, n_inj = simulate_malware_attack(
        X_test, y_test, scaler, numerical_cols, factor, 42)

    y_pred_sim = best_xgb_model.predict(X_sim)
    y_pred_proba_sim = best_xgb_model.predict_proba(X_sim)[:, 1]

    cm = confusion_matrix(y_sim, y_pred_sim)
    report = classification_report(y_sim, y_pred_sim, output_dict=True, zero_division=0)
    roc_auc = roc_auc_score(y_sim, y_pred_proba_sim)
    pr_auc = auc(precision_recall_curve(y_sim, y_pred_proba_sim)[1],
                 precision_recall_curve(y_sim, y_pred_proba_sim)[0])

    results_malware[level] = {
        'confusion_matrix': cm,
        'report': report,
        'roc_auc': roc_auc,
        'pr_auc': pr_auc,
        'anomalias_injetadas': n_inj
    }

    print(f"Total de eventos injetados: {n_inj}")
    print(f"Total de anomalias reais (original + injetadas): {y_sim.sum()}")
    print("\n--- Relatório de Classificação ---")
    print(classification_report(y_sim, y_pred_sim, zero_division=0))

# Visualizar a matriz de confusão
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Normal (0)', 'Anomalia (1)'], yticklabels=['Normal (0)', 'Anomalia (1)'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title(f'Matriz de Confusão Malware - {level.upper()}')
plt.show()

# Visualizar a curva PR-AUC
precision, recall, _ = precision_recall_curve(y_sim, y_pred_proba_sim)
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'Curva PR Malware {level.upper()} (area = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title(f'Curva Precision-Recall Malware - {level.upper()}')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()

# Apresentar o resumo dos resultados em uma tabela
results_df_malware = pd.DataFrame([
    {
        'Intensidade': level,
        'Precision': results_malware[level]['report']['1']['precision'],
        'Recall': results_malware[level]['report']['1']['recall'],
        'F1-score': results_malware[level]['report']['1']['f1-score'],
        'FP': results_malware[level]['confusion_matrix'][0, 1],
        'FN': results_malware[level]['confusion_matrix'][1, 0],
        'ROC-AUC': results_malware[level]['roc_auc'],
        'PR-AUC': results_malware[level]['pr_auc']
    } for level in intensities.keys()
])

```