Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Detecção de novidade para sistemas de detecção de intrusão

Júlio César Moura de Oliveira

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof. Dr. Luís Paulo Faina Garcia

Brasília
2026

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

Moura de Oliveira, Júlio César
M929d Detecção de novidade para sistemas de detecção de intrusão / Júlio César Moura de Oliveira; orientador Luís Paulo Faina Garcia. Brasília, 2026.
88 p.

Dissertação(Mestrado em Informática) Universidade de Brasília, 2026.

1. Aprendizado de Máquina. 2. Detecção de Intrusão. 3. Fluxos Contínuos de Dados. 4. Detecção de Novidade. I. Garcia, Luís Paulo Faina , orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Detecção de novidade para sistemas de detecção de intrusão

Júlio César Moura de Oliveira

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof. Dr. Luís Paulo Faina Garcia (Orientador)
CIC/UnB

Profa. Dra. Elaine Ribeiro de Faria Paiva
FACOM/UFU

Prof. Dr. João José Costa Gondim
ENE/UnB

Profa. Dra. Cláudia Nalon
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 24 de mar de 2026

Dedicatória

Dedico este trabalho aos meus pais, que acreditaram na capacidade transformadora da educação e sempre me apoiaram em todos os objetivos que busquei ao longo da vida.

Agradecimentos

Agradeço, primeiramente, a Deus, pela força recebida para superar os momentos mais difíceis desta jornada.

À minha família, em especial aos meus pais, Regina e Geraldo, agradeço pelo apoio incondicional que sempre recebi. À minha esposa e ao meu filho, agradeço pela paciência e compreensão nos momentos de ausência.

Expresso minha gratidão ao meu orientador, Prof. Luís Paulo Faina Garcia, por todo o conhecimento compartilhado, por sua paciência e dedicação, sem os quais a realização desta pesquisa não seria possível. O apoio recebido durante a orientação foi fundamental em um período marcado por diversos desafios pessoais e profissionais. À Universidade de Brasília, agradeço pela oportunidade concedida, permitindo que eu me tornasse um profissional mais completo.

Por fim, agradeço à Força Aérea Brasileira e aos meus colegas de farda pelo incentivo e suporte ao longo desta caminhada.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

No contexto do aumento da conectividade das redes de computadores e da crescente sofisticação dos ataques cibernéticos, a detecção de intrusão tornou-se um desafio cada vez mais crítico para a segurança das informações armazenadas e transmitidas. O tráfego dessas redes pode ser modelado como um fluxo contínuo de dados, onde os exemplos chegam de forma constante e a distribuição subjacente pode evoluir ao longo do tempo. Em tais fluxos, podem ocorrer fenômenos como mudanças de conceito em classes existentes ou o surgimento de novas classes. Este trabalho investiga técnicas desenvolvidas para a tarefa de Detecção de Novidade, que busca identificar padrões não explicados pelo modelo que representa o comportamento conhecido, aplicadas ao problema de detecção de intrusão. O objetivo da pesquisa é avaliar a capacidade de alguns algoritmos de identificar classes de ataques não presentes na fase de treinamento, enquanto buscam manter a precisão na classificação de classes já conhecidas. Para isso, foi realizada uma análise experimental com diferentes técnicas de Detecção de Novidades: ECSMiner, MINAS, SENCForest, KNNENS e Trident. Além da capacidade preditiva, também foram avaliados aspectos relacionados à eficiência computacional dos algoritmos. Os resultados mostraram que o SENCForest apresentou a maior agilidade na detecção inicial de novas classes de ataques e obteve o menor consumo de memória principal. Por outro lado, MINAS e Trident demonstraram maior viabilidade para operação em tempo real em redes de alto tráfego devido às suas taxa de processamento de amostras.

Palavras-chave: Fluxo de dados, Detecção de novidade, detecção de intrusão, surgimento de novas classes

Abstract

In the context of growing computer network connectivity and the increasing sophistication of cyberattacks, intrusion detection has become an ever more critical challenge for the security of stored and transmitted information. Network traffic can be modeled as a continuous data stream, where examples arrive constantly and the underlying distribution may evolve over time. In such streams, phenomena such as concept drift in existing classes or the emergence of new classes may occur. This study investigates techniques developed for the Novelty Detection ND task, which aims to identify patterns not explained by the model representing known behavior, applied to the problem of intrusion detection. The research objective is to evaluate the ability of certain algorithms to identify attack classes that were not present during the training phase, while striving to maintain accuracy in the classification of previously known classes. To achieve this, an experimental analysis was conducted using different Novelty Detection techniques: ECSMiner, MINAS, SENCForest, KNNENS, and Trident. In addition to predictive performance, aspects related to the computational efficiency of these algorithms were also evaluated. Results showed that SENCForest exhibited the highest agility in the initial detection of new attack classes and achieved the lowest main memory consumption. On the other hand, MINAS and Trident demonstrated greater feasibility for real-time operation in high-traffic networks due to their sample processing rates.

Keywords: Data streams, Novelty Detection, intrusion detection, emerging new classes

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AD Árvore de Decisão.

AM Aprendizado de Máquina.

APT Advanced Persistent Threat.

DN Detecção de Novidade.

FCD Fluxo Contínuo de Dados.

IDS Intrusion Detection System.

KNN K-Nearest Neighbors.

RNA Redes Neurais Artificiais.

SVM Support Vector Machine.

TTD Time-To-Detection.

Capítulo 1

Introdução

No atual cenário de crescente expansão de conectividade, os ataques cibernéticos se tornam cada vez mais sofisticados e frequentes. De acordo com o CrowdStrike 2025 Global Threat Report [1], os atores maliciosos que atuam no espaço cibernético tem evoluído cada vez mais rápido, utilizando técnicas e ferramentas inovadoras e buscando soluções criativas para burlar as soluções de defesa. O relatório informa que observou-se um aumento significativo na velocidade da ação dos atacantes. Os dados apresentados evidenciam que os adversários, uma vez infiltrados, agem em questão de segundos, impondo um ritmo operacional que desafia os métodos de defesa convencionais. O estudo concluiu que este aumento de velocidade de ação reforça a necessidade de uma capacidade de detecção de ameaças em tempo real, capaz de identificar e parar intrusões antes que estas possam ganhar maior dimensão.

A detecção de ataques cibernéticos exige a análise contínua do tráfego de rede em tempo real, de modo a caracterizar comportamentos normais e identificar desvios que possam indicar atividades maliciosas. Dados de tráfego podem ser registrados de muitas formas, como pacotes contendo endereços IP de origem e destino e *flows* que mostram o número de pacotes enviados, hora de início, hora de término e protocolos de rede utilizados [2]. Neste cenário, os *Intrusion Detection System* (IDS) desempenham um relevante papel na proteção das redes de computadores. Esses sistemas são úteis para monitorar o tráfego de rede em tempo real, identificando e alertando sobre possíveis atividades maliciosas que possam comprometer a integridade, confidencialidade ou disponibilidade dos dados [3].

Os IDS geralmente operam com base em assinaturas, anomalias ou ambos [4]. Nas implementações baseadas em assinaturas, os IDS comparam os dados recebidos com os padrões de ataque em seu banco de dados. Se uma correspondência for encontrada, um alarme será gerado. Se um ataque não constar na base de assinaturas do IDS, o ataque não será detectado. Em sistemas baseados em anomalias ou comportamentos, os padrões de interesse precisam ser aprendidos [5, 6]. Nesse caso, os IDS monitoram os eventos na

rede e um alarme é gerado sempre que houver correspondência com algum padrão modelado. Os IDS baseados em anomalias mais amplamente utilizados se valem de técnicas de Aprendizado de Máquina (AM) [7]. No contexto de detecção de intrusão, AM é utilizado como uma ferramenta para aprender características relevantes do tráfego de rede e prever atividades normais e anormais com base nos padrões aprendidos. Diversas pesquisas vem sendo realizadas nos últimos anos para avaliar a eficácia de IDS que utilizam AM, apresentando resultados bastante efetivos na detecção de intrusão em redes de computadores [6, 5, 8].

Muitos dos trabalhos de IDS baseados em AM utilizam fontes de dados estáticas [9], com um número pré-determinado de tipos de ataques. O tráfego de dados gerado por uma rede de computadores, no entanto, deve ser modelado como um fluxo contínuo de dados (FCD) [10]. FCD tem características desafiadoras para sua análise: são gerados continuamente, eventualmente em grandes quantidades, possuem tamanho infinito ou desconhecido e podem mudar ao longo do tempo [11]. Entre essas possíveis mudanças, destacam-se o surgimento de novos conceitos ou novas classes. Esses novos conceitos são representados por amostras que diferem consideravelmente das demais já conhecidas, mas que se assemelham entre si [10]. No tráfego gerado por uma rede de computadores, os novos conceitos podem representar novas técnicas de ataques percebidas ao longo do tempo. Assim, em uma rede monitorada por um IDS que utilize AM, essas novas classes denotam padrões não encontrados durante o treinamento do modelo de classificação, mas que surgem ao longo do tempo. Uma vez que o classificador foi treinado apenas com dados históricos, ele não conseguirá identificar um novo tipo de ataque.

O desafio de identificar novos conceitos em um FCD exige, portanto, o desenvolvimento de abordagens que possam se adaptar continuamente às mudanças. Além de manter a capacidade de predição das classes previamente conhecidas, este tipo de técnica deve ser capaz de detectar o surgimento de novas classes e se atualizar de forma automática, com recursos computacionais limitados e com acesso a pouco ou nenhum dado rotulado ao longo do fluxo [12]. Diversos trabalhos abordando detecção de novas classes em FCD vem sendo produzidos nos últimos anos [13, 10, 14, 15, 16, 17, 18, 12, 19].

Em geral, técnicas de Detecção de Novidades (DN) em FCD tem uma etapa inicial de aprendizado supervisionado, onde um modelo é inferido. Esse modelo é utilizado para a classificação das amostras ao longo do fluxo. Desta forma, se as amostras avaliadas aparentam pertencer a uma das classes existentes, seu rótulo de classe é predito pelo modelo. Caso contrário, elas são marcadas como potenciais amostras de classe emergentes e armazenadas em um *buffer*. Esse *buffer* é examinado periodicamente, onde são aplicados critérios para detecção de uma nova classe. Quando uma nova classe é identificada, o modelo inicial é atualizado, de modo a adicionar este novo conceito às demais classes

conhecidas [20].

As técnicas de DN apresentadas nos trabalhos mais recentes demonstram resultados promissores, com experimentos aplicados a problemas de diversas áreas de conhecimento. Dentre as aplicações está a detecção de intrusão, como em [13, 10, 14, 18, 19].

1.1 Justificativa

Dada a importância da DN em FCD, pesquisadores vem se dedicando ao desenvolvimento de técnicas para lidar com este desafio [21, 13, 10, 22, 23, 14, 15, 16, 17, 18, 12, 24, 25]. Além da capacidade de detecção de novas classes, é importante que os algoritmos de DN sejam capazes de manter o desempenho durante a evolução do fluxo. Igualmente importante é a capacidade desses algoritmos de processar sequencialmente grandes quantidades de amostras de dados em tempo real sem sobrecarregar os recursos computacionais, uma vez que os FCD são potencialmente infinitos e variáveis em volume.

A maior parte dos trabalhos desenvolvidos neste tema busca demostrar a eficácia dos métodos propostos de forma mais geral, sem se aprofundar em domínios mais específicos, como a detecção de intrusão. Em geral, os experimentos com algoritmos de DN utilizam diferentes *datasets* que não tem correlação entre si ou que são mais adequados para problemas de *batch* [26]. Dentre os trabalhos com abordagem multiclasse mais recentes, apenas em [13, 27, 14, 28, 18, 19, 29] há experimentos de detecção de novas classes com um *dataset* de detecção de intrusão. O *dataset* utilizado nesses trabalhos é, na maioria das vezes, o KDD99 [30], um *benchmark* relevante para experimentos com IDS, mas que possui limitações, como ataques pouco relevantes nos dias atuais. Além disso, aspectos relativos à eficiência computacional dos algoritmos foram pouco discutidos, embora sejam relevantes para se decidir pela utilidade de uma determinada técnica. A aplicabilidade de técnicas de DN para detecção de intrusão depende, no entanto, de estudos mais detalhados sobre o desempenho de cada algoritmo no cenário de interesse da sua aplicação.

Dado esse contexto, este trabalho se propôs a investigar, de forma criteriosa, a aplicabilidade de algoritmos de DN a problemas de detecção de intrusão. Portanto, este trabalho buscou apresentar uma contribuição para esta área de pesquisa por meio de uma análise experimental detalhada, utilizando *datasets* adequados ao problema e explorando aspectos relevantes para esse tipo de aplicação: a capacidade preditiva do modelo ao longo do FCD e a eficiência computacional na avaliação das amostras.

1.2 Objetivos

O principal objetivo deste trabalho foi avaliar o desempenho de técnicas de DN na tarefa de detecção de novas classes de intrusão em tráfego de redes de computadores, com ênfase em sua aplicabilidade em IDS. Para isso, foram realizados experimentos com algumas das mais relevantes técnicas de DN e alguns dos mais atuais *datasets* de detecção de intrusão.

Para alcançar o objetivo, o estudo avaliou a capacidade dos algoritmos de DN de identificar novas classes de intrusão em tráfego de redes de computadores, ou seja, que não estavam presentes durante a fase de treinamento. Para esta atividade, foram utilizadas métricas de avaliação específicas para medir a capacidade de identificação de novas classes. Além disso, o trabalho apresentou uma nova métrica que mostra de forma mais clara a capacidade dos algoritmos de diferenciar uma classe conhecida de uma nova classe.

Os experimentos também buscaram demonstrar a capacidade preditiva geral dos modelos induzidos pelos algoritmos de DN ao longo do fluxo, utilizando métricas tradicionais, como a acurácia. A avaliação da capacidade preditiva ao longo do fluxo é importante para verificar se a evolução contínua dos dados afetará o desempenho do classificador.

Por fim, esta pesquisa objetivou deixar como contribuição experimentos relativos à eficiência computacional dos algoritmos em dois aspectos relevantes para o contexto de FCD: o gerenciamento de memória RAM e a taxa de avaliação de amostras. Esses fatores são determinantes para a viabilidade de aplicação em ambientes de rede, onde grandes volumes de dados precisam ser processados com rapidez e eficiência, sem sobrecarregar os recursos de hardware disponíveis.

1.3 Hipóteses

Este trabalho procurou validar as seguintes hipóteses:

1. As técnicas de DN avaliadas são capazes de classificar corretamente amostras de classes conhecidas e identificar novas classes de intrusão em tráfego de redes de computadores, sem ter sua capacidade preditiva degradada ao longo do fluxo.
2. As técnicas de DN avaliadas são capazes de processar sequencialmente as amostras de tráfego no mínimo na mesma taxa de transferência do FCD e com consumo de memória estável, garantindo a capacidade de análise em tempo real.

1.4 Organização do Documento

O Capítulo 2 deste trabalho apresenta a revisão bibliográfica realizada para dar subsídios a esta pesquisa, abordando aspectos como detecção de intrusão, fluxo de dados e abordando

o surgimento de novas classes em fluxos de dados. O Capítulo 3 resume resultados obtidos por alguns dos mais relevantes trabalhos na área. O Capítulo 4 apresenta a metodologia utilizada para este estudo. Os resultados obtidos estão no Capítulo 5 e o Capítulo 6 apresenta as conclusões e os próximos passos no desenvolvimento da pesquisa.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta uma revisão bibliográfica dos conceitos utilizados neste trabalho, com o objetivo de fornecer o embasamento teórico necessário para abordar o tema em questão. Este capítulo é dividido da seguinte forma: na Seção 2.1 são introduzidos os conceitos de detecção de intrusão; na Seção 2.2 são abordados aspectos importantes de FCD; na Seção 2.3 são apresentados o conceito do surgimento de novas classes em FCD e algumas das mais relevantes técnicas de DN desenvolvidas para lidar com este tipo de problema. Por fim, a Seção 2.4 apresenta métricas utilizadas para avaliar o desempenho de algoritmos que buscam identificar novas classes em FCD.

2.1 Detecção de Intrusão

As redes de computadores são essenciais para a comunicação e troca de informações, mas expõem os sistemas a ameaças externas que precisam ser continuamente monitoradas e detectadas. Sistemas de detecção de intrusão são ferramentas utilizadas para monitorar e analisar o tráfego de rede em busca de sinais de comprometimento [31]. Quando esse tipo de sistema atua de modo a bloquear o ataque detectado, recebe o nome de *Intrusion Prevention System* (IPS); caso este seja utilizado apenas para a detecção de eventos, sem realizar bloqueios, chamamos de IDS [32]. A partir deste ponto, soluções de IDS ou IPS serão genericamente referenciadas como IDS, visto que o aspecto mais importante desse tipo de sistema para esta pesquisa é a sua capacidade de detecção de tráfego malicioso, independente das ações realizadas após a detecção.

Os IDS podem ser classificados também quanto ao escopo de monitoramento. Um *Host-based Intrusion Detection System* (HIDS) opera localmente em um dispositivo, monitorando eventos do sistema operacional, chamadas de sistema, registros de log e atividades de processos. Já um *Network-based Intrusion Detection System* (NIDS) atua sobre o tráfego de rede, analisando pacotes ou fluxos que transitam entre dispositivos [33]. Neste

trabalho, considera-se implicitamente o contexto de NIDS, visto que os *datasets* utilizados são compostos por dados de tráfego de rede e as técnicas avaliadas operam sobre atributos extraídos de fluxos de comunicação entre hosts.

É possível que as atividades anômalas percebidas por um IDS sejam ocasionadas por ataques cibernéticos. Os ataques cibernéticos, em geral, exploram vulnerabilidades e se aproveitam delas para gerar algum resultado não previsto pelo desenvolvedor da aplicação afetada [32]. Desta forma, considerando que há usos incomuns de serviços de tecnologia da informação, alterações no perfil de comunicações podem ser percebidas. Dentre estas alterações, é possível, por exemplo, identificar parâmetros como endereços de IP considerados maliciosos, clientes HTTP desconhecidos ou reconhecidamente maliciosos, requisições HTTP que buscam acessar informações restritas no servidor web, entre outras informações contidas nos pacotes trafegados que diferem do previsto. Além disso, uma distribuição incomum de serviços de rede ao longo do tempo pode ser indício de algum ataque cibernético em curso [34]. Para detectar estes comportamentos possivelmente maliciosos, um IDS pode atuar em nível de pacote, examinando cabeçalhos e, quando disponível, o conteúdo das mensagens [32].

Em relação à estratégia de detecção, os IDS costumam ser classificados em detecção de anomalias e detecção baseada em assinaturas [31]. A detecção baseada em assinatura compara diretamente eventos capturados com ataques previamente documentados, sendo mais precisa e gerando menos falsos positivos. Como essa abordagem depende de uma correspondência exata com a assinatura, ela não é eficaz para detectar novos ataques. Já a detecção de anomalias modela o comportamento normal do usuário, observando-o ao longo do tempo e gerando alarmes quando há desvios. Os IDS baseados em anomalias mais amplamente utilizados se valem de técnicas de AM [7].

A Figura 2.1 ilustra a aplicação de AM para incrementar a capacidade de detecção de um IDS. Neste contexto, algoritmos de AM são usados em tarefas de classificação, de modo a categorizar as amostras de tráfego de dados dentro de um conjunto de classes previamente definidas, incluindo o tráfego considerado normal. O processo de AM é iniciado com a escolha de *datasets* adequados para a atividade preditiva desejada. Um algoritmo de aprendizado é executado em uma porção de dados, onde será induzido um modelo a ser usado posteriormente nas predições. Este processo de indução do modelo é conhecido como treinamento. O algoritmo de AM buscará capturar as características dos dados legítimos e maliciosos utilizados durante o treinamento para identificar ataques presentes no tráfego durante sua fase de predição [31].

As bases de dados utilizadas para o treinamento geralmente passam por uma etapa de pré-processamento, que inclui tarefas como limpeza e transformação de dados. De acordo com Maharana *et al.* (2022) [35], dados reais são frequentemente vulneráveis a

ruído, corrupção, valores ausentes e inconsistências, o que pode ocultar padrões relevantes e comprometer a qualidade das análises e o desempenho dos modelos. Assim, a limpeza e a transformação dos dados contribuem para tornar a extração de conhecimento mais confiável, permitindo que os modelos aprendam padrões mais precisos a partir de dados de entrada com melhor qualidade.

De acordo com [36], os *datasets* de IDS normalmente reúnem diferentes tipos de dados, desde pacotes brutos (PCAP) e *payloads* até fluxos agregados (NetFlow/IPFIX), *logs* de ativos de rede (*syslog*, eventos do sistema operacional), informações de chamadas de sistema e telemetria de aplicações, cada um oferecendo granularidades e custos de tratamento distintos. Dados em nível de fluxo, ou simplesmente *flows*, consistem em um conjunto de atributos extraídos da comunicação de rede entre hosts ou serviços, como contadores e estatísticas. O protocolo NetFlow, inicialmente introduzido por dispositivos Cisco, consolidou esse modelo ao representar um fluxo como uma sequência unidirecional de pacotes trocados entre dois pontos finais que compartilham endereços, portas e tipo de serviço. Os autores explicam que o uso de *flows* tem se tornado o mais popular porque combina escalabilidade operacional (menor custo de armazenamento e processamento que PCAPs), melhor privacidade (*payloads* não são necessários), compatibilidade com tráfego criptografado (detecção por padrões de fluxo em vez de inspeção de conteúdo) e suporte nativo em dispositivos de rede (coleta NetFlow/IPFIX), o que facilita a instrumentação em ambientes reais. Além disso, *flows* permitem extração eficiente de atributos temporais e estatísticos úteis, como taxas, durações, contagens de pacotes/bytes. Os autores também ressaltam uma limitação: ao perder a visibilidade do *payload*, o uso exclusivo de *flows* reduz a capacidade de detectar ataques que dependem de conteúdo fino, exigindo, quando necessário, estratégias híbridas ou amostragem seletiva de PCAPs para investigação aprofundada.

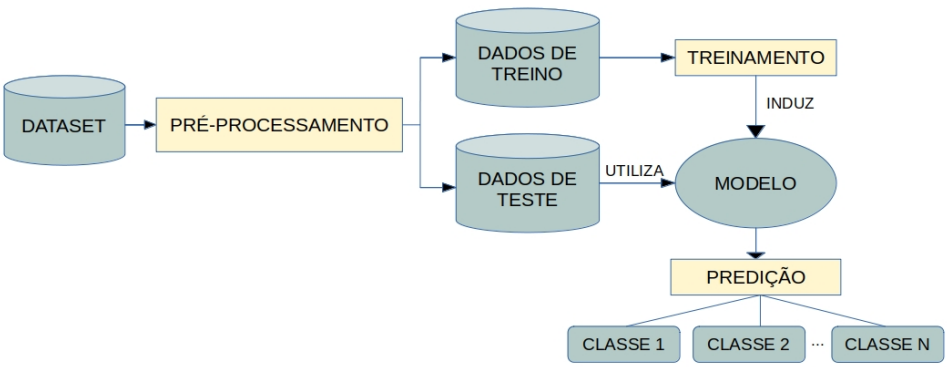

Figura 2.1: Aprendizado de Máquina em Detecção de Intrusão.

Em redes de computadores, o tráfego de rede é produzido continuamente e pode apresentar mudanças nos padrões de comportamento ao longo do tempo, seja pela evolução do uso legítimo dos serviços, seja pelo surgimento de novos tipos de ataque. Assim, a atividade de detecção de intrusão deve levar em consideração aspectos pertinentes aos FCD, tema abordado na próxima seção.

2.2 Fluxos Contínuos de Dados

FCD refere-se ao processo de transmissão constante e em tempo real de grandes quantidades de informações digitais entre diferentes sistemas ou dispositivos [11]. Esta é uma característica de diversas aplicações modernas, como tráfego de redes de computadores, dados do mercado financeiro e serviços de *streaming* de mídia. Aspectos relativos ao gerenciamento e análise de FCD têm sido pesquisados com frequência nos últimos anos [37, 38, 34, 39, 40, 41, 42].

Conforme apresentado em Din *et al.* (2021) [20], um FCD pode ser definido como uma sequência potencialmente infinita de amostras de dados que chegam continuamente em alta velocidade. Formalmente, o FCD pode ser definido como $S = \{(x_j, y_j)\}_1^\infty$, onde o par (x_j, y_j) é uma amostra que chegou em um tempo j, $x_j \in R^n$ é um vetor de atributos n-dimensional e $y_j \in Y = \{C_1, C_2, ..., C_k\}$ é um conjunto de rótulos de classe (se disponível) associado às amostras.

De acordo com Gama (2010) [11], algumas das principais características de FCD são as seguintes:

1. Ao contrário de conjuntos de dados estáticos que são carregados de uma só vez, FCD chegam continuamente em um sistema. A natureza contínua dos fluxos exige técnicas eficientes de processamento de dados.

2. O sistema que processa um FCD não tem controle sobre a ordem em que as amostras chegam.

3. Os FCD são potencialmente ilimitados em tamanho. Eles se caracterizam por um volume significativo de dados, geralmente gerados em alta velocidade e de forma contínua. Essa característica os diferenciam de conjuntos de dados estáticos, que possuem tamanho finito e estrutura definida.

4. Depois que uma amostra de um FCD é processada, ela é descartada ou arquivada. Ela não pode ser recuperada facilmente, a menos que seja explicitamente armazenada em memória, que é pequena em relação ao tamanho dos fluxos.

As técnicas mais tradicionais de AM geralmente funcionam em um modelo de aprendizado em lote ou aprendizado *offline*, em que um modelo é treinado por algum algoritmo de aprendizado a partir de um conjunto inteiro de dados de treinamento de uma só vez e, em seguida, o modelo é induzido sem realizar qualquer atualização posterior. Esse método de aprendizado sofre com altos custos de retreinamento ao lidar com novos dados de treinamento e, portanto, são pouco escaláveis para aplicações em ambientes mais dinâmicos [43].

Dessa forma, aprendizado nesse contexto requer o desenvolvimento de novos algoritmos e métodos que levem a modelos que se autoadaptem ao longo do tempo. Deve ser, portanto, um processo contínuo e com atualizações em tempo (quase) real do modelo [44]. Este tipo de abordagem é por vezes referenciado como aprendizado *online* [43].

Bifet *et al.* (2018) [45] resumem da seguinte forma os requisitos para um algoritmo que necessite realizar aprendizado *online*:

1. processar uma única amostra por vez;
2. processar cada amostra em um período de tempo limitado;
3. usar uma quantidade limitada de memória;
4. estar pronto para realizar predição a qualquer momento;
5. ser capaz de se adaptar a mudanças na distribuição de dados.

Os requisitos apresentados mostram, portanto, que FCDs exigem que algoritmos de AM sejam capazes de aprender de forma incremental, atualizando o conhecimento continuamente com novas observações. Algoritmos incrementais processam uma amostra de treinamento de cada vez, sem reprocessar amostras anteriores, mantendo apenas uma estrutura de conhecimento na memória e permitindo que o treinamento seja interrompido a qualquer momento para uma predição [10]. Esses algoritmos oferecem vantagens como economia de memória, eficiência no tempo de processamento e maior rapidez na resposta.

Bahri *et al.* (2021) [46] explicam que as abordagens incrementais para classificação em FCD são comumente divididas em duas categorias principais: incremental em *batch* ou incremental de amostra. Os métodos incrementais de amostra podem, de fato, aprender com cada amostra que surge. Na abordagem incremental em *batch*, um método tradicional de aprendizado em *batch* é treinado, de modo que cada novo lote de dados completo, de tamanho fixo w, é continuamente entregue ao algoritmo. Esta abordagem força a exclusão de modelos treinados para abrir espaço para novos. Conforme explicado por Molina-Coronado *et al.* (2020) [47], há estratégias de aprendizado em *batch* para fluxos onde um componente supervisor monitora o fluxo ou a resposta do sistema e decide se o retreinamento é necessário. Esta estratégia busca, portanto, mitigar uma limitação de

abordagens de *batch* tradicionais ao monitorar mudanças e ajustar o modelo conforme necessário.

Devido à dinâmica de sua natureza, um aspecto que pode ser observado em FCD é a mudança de conceito. Mudança de conceito refere-se à mudança nas distribuições de dados ao longo do tempo, o que pode levar a uma degradação do desempenho dos modelos de AM [48]. Essa mudança pode ocorrer devido a diversas razões, como alterações nas condições do ambiente, comportamento dos usuários ou evolução dos fenômenos monitorados. Em essência, o conceito implica que os padrões nos dados não são estáticos, de modo que a relação existente entre os atributos de entrada e saída do modelo é alterada [49]. Sendo assim, algoritmos de aprendizado utilizados em FCD devem ter mecanismos capazes de incorporar essas mudanças de conceito e adaptar o modelo de decisão ao estado recente do FCD [10].

Outro aspecto relevante em um FCD é que amostras podem diferir de forma significativa das demais classes de dados conhecidas. Muitas destas amostras podem ser consideradas *outliers*. Há várias definições na literatura para este tipo de amostra. De acordo com Al *et al.* (2021) [38], *outliers* podem ser definidos como pontos de dados considerados fora do comum. De acordo com Din *et al.* (2021) [20], *outliers* podem ser gerados por falha humana, mudanças no ambiente, mau funcionamento da instrumentação e eventos maliciosos. Os *outliers* se caracterizam por serem independentes e esparsos, sem representarem quaisquer novas classes.

Há situações, por outro lado, em que a quantidade de amostras desconhecidas ou inesperadas é considerável e há algumas semelhanças entre elas. Nesse caso, é possível que estas amostras representem um novo conceito, ainda não explicado pelo modelo de decisão atual.

2.3 Detecção de Novidade em Fluxos Contínuos de Dados

DN pode ser definida como uma tarefa de aprendizado que busca identificar perfis de dados que emergem do universo de amostras que não são explicadas pela representação atual do comportamento conhecido. Esses novos perfis podem representar novos conceitos, uma mudança nos conceitos conhecidos ou a presença de ruídos [11]. Dessa forma, em tarefas de DN os conceitos conhecidos podem ser compostos por diferentes classes e novas podem surgir ao longo do tempo.

A Figura 2.2 exemplifica esse cenário, ilustrando um espaço de atributos onde há duas classes conhecidas em um tempo t e amostras de uma nova classe estão presentes em um tempo $t + k$. É possível perceber também amostras consideradas *outliers*.

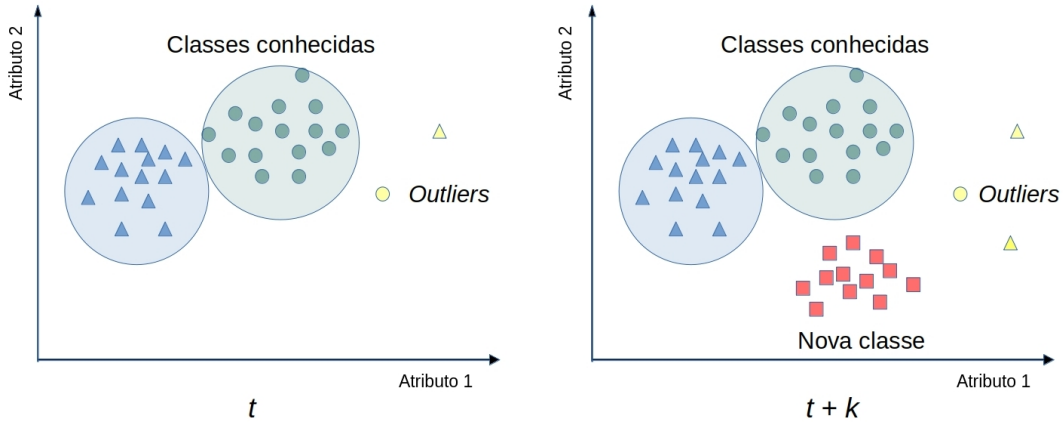

Figura 2.2: Classes conhecidas, nova classe e *outliers*.

Técnicas de DN geralmente têm uma etapa *offline* e uma etapa *online* [10]. A Figura 2.3 apresenta uma ideia geral utilizada por diversas abordagens de DN. Na etapa *offline* um modelo ($f : X \rightarrow Y$) é aprendido com um conjunto de dados de treinamento inicial $X = D_{init} = \{(x_i, y_i)\}_1^m$. Na fase *online*, caso uma amostra x_j do fluxo possa ser explicada pelo atual modelo de decisão, o algoritmo realizará a classificação. Em caso negativo, esta amostra será adicionada a um *buffer* B, que funciona como uma memória temporária. Esse *buffer* é analisado oportunamente e critérios específicos de cada algoritmo podem ser aplicados às amostras nele contidas. Caso esses critérios sejam atendidos em alguma porção dessas amostras, o modelo de decisão é atualizado incluindo a nova classe identificada [10]. Também é possível tomar uma decisão sobre a amostra antes de adicioná-la ao *buffer* e, sob certas condições, utilizar o conteúdo armazenado para a atualização do modelo.

Alguns algoritmos implementam também um mecanismo de esquecimento para remover conceitos desatualizados do modelo que foram aprendidos anteriormente, com o objetivo de tornar o modelo alinhado com as tendências mais recentes nos dados [20]. Em técnicas que utilizam *ensemble* de classificadores, novos classificadores podem ser treinados e substituir antigos. Quando o algoritmo é baseado em agrupamentos, grupos que não recebem novas amostras por um longo tempo podem ser removidos [10].

Diversas estratégias são adotadas para lidar com o desafio de detectar novas classes em FCD. Em algumas abordagens, o modelo é construído a partir de um único conceito, tipicamente o comportamento considerado normal, e a tarefa consiste em decidir se uma nova amostra é compatível com esse conceito ou se deve ser tratada como anômala. Nesse cenário, o modelo não busca diferenciar subclasses do comportamento normal; ele apenas separa amostras normais de *outliers* ou anômalos. Neste caso, diz-se que a abordagem é *one-class*, *single-class* ou *learning from positive-only* [11]. Exemplos destes algoritmos

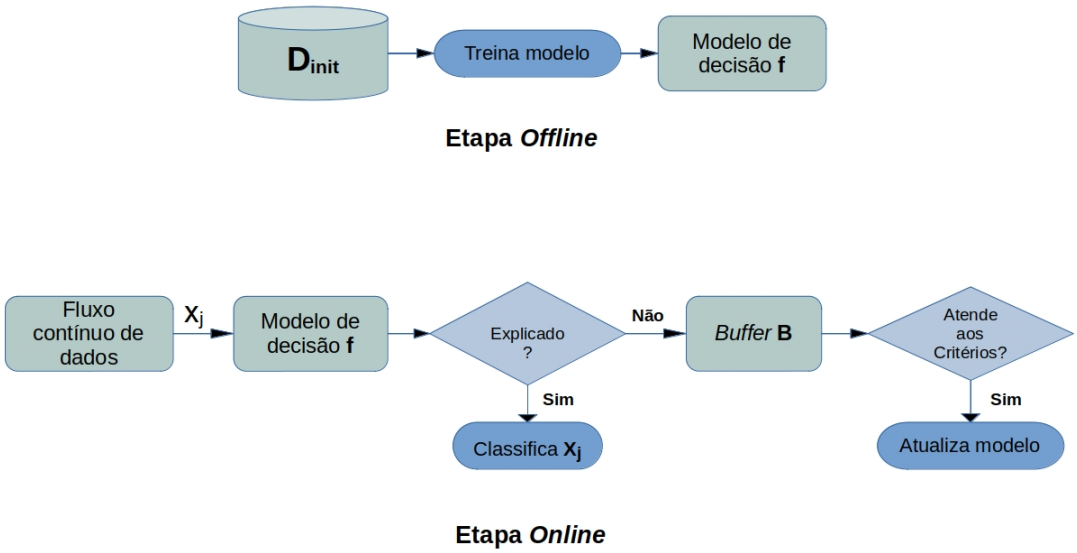

Figura 2.3: Abordagem geral de técnicas de DN.

são o *One-Class Support Vector Machines* OCSVM [50] e o *Support Vector Mapping Convergence* (SVMC) [51].

Mais recentemente, estratégias de DN foram estendidas ao cenário multiclasse, onde o algoritmo não só é capaz de identificar múltiplas classes conhecidas, mas também múltiplas classes novas [26]. IDS em geral buscam diferenciar os vários tipos de ações maliciosas detectadas. Desta forma, uma abordagem multiclasse é mais indicada para DN em detecção de intrusão.

As abordagens de DN multiclasse em FCD podem ser categorizadas em quatro grandes grupos fundamentais, baseados na forma como os modelos de classificação e detecção são construídos e atualizados incrementalmente. Não é uma classificação inflexível, mas uma forma de entender as estratégias mais comuns.

O primeiro grupo compreende as técnicas baseadas em agrupamento ou *clustering*, que são amplamente consolidadas na literatura devido à sua eficiência em representar modelos compactos. Segundo Gaudreault e Branco (2024) [26], esses métodos utilizam o conceito de micro-grupos para armazenar resumos estatísticos das classes conhecidas, definindo fronteiras de decisão em torno dessas regiões. Quando novas amostras surgem fora dessas fronteiras, elas são mantidas em um *buffer* temporário. A detecção de uma nova classe ocorre quando os dados nesse *buffer* apresentam alta coesão interna e estão suficientemente distantes dos agrupamentos existentes, permitindo que o sistema identifique múltiplas novas classes de forma autônoma conforme o fluxo evolui.

Um segundo grupo relevante é composto por técnicas baseadas em árvores de decisão (AD). Segundo Agrahari *et al.* (2024) [52], essas abordagens exploram a estrutura hierár-

quica para particionar o espaço de atributos e isolar padrões desconhecidos em sub-regiões de baixa densidade. O funcionamento geral baseia-se no monitoramento dos nós folha; se um novo dado percorre um caminho na árvore que leva a uma folha com características estatísticas divergentes ou se a amostra exige um isolamento muito rápido (como em florestas de isolamento), ela é marcada como candidata a novidade. Em cenários multiclasse, essas estruturas permitem a expansão de novos nós ou o retreinamento parcial de ramos específicos para acomodar novas classes sem comprometer o conhecimento previamente adquirido sobre as classes normais e de ataque já conhecidas.

As abordagens fundamentadas em Redes Neurais Artificiais (RNA) constituem o terceiro grupo, ganhando destaque pela capacidade de processar dados complexos e de alta dimensionalidade. Conforme discutido por Din *et al.* (2021) [20], uma estratégia comum nestes modelos é a utilização do erro de reconstrução, frequentemente implementada através de autoencoders. A rede é treinada para comprimir e reconstruir amostras das classes conhecidas com alta precisão; logo, amostras pertencentes a classes inéditas tendem a gerar um erro de reconstrução significativamente elevado, funcionando como um gatilho para a DN. Além disso, arquiteturas dinâmicas permitem que a rede adicione novos neurônios ou camadas de saída de forma incremental para representar as novas classes identificadas, mitigando problemas de esquecimento catastrófico inerentes ao aprendizado em fluxo.

Por fim, o quarto grupo abrange outras abordagens que incluem métodos estatísticos, probabilísticos e modelos híbridos. De acordo com Agrahari *et al.* (2024) [52], técnicas baseadas em modelos de mistura de gaussianas ou estimativas de densidade de kernel buscam modelar a distribuição de probabilidade subjacente das classes. Uma amostra é classificada como novidade se a probabilidade de ela pertencer a qualquer uma das distribuições conhecidas for inferior a um limiar crítico. Outros métodos utilizam *Support Vector Machines* (SVM) adaptadas para cenários multiclasse, mantendo fronteiras de decisão otimizadas em espaços de alta dimensão. Esses sistemas frequentemente combinam diferentes métricas de dissimilaridade para distinguir entre mudanças de conceito em classes já existentes e a emergência real de novas categorias de dados, buscando a robustez do classificador em ambientes não estacionários.

As subseções a seguir explicam o funcionamento de algumas das mais recentes ou mais relevantes técnicas de ND multiclasse. Algumas dessas foram selecionadas para as atividades experimentais desta pesquisa.

2.3.1 ECSMiner

O *Enhanced Classifier for data Streams with novel class Miner* (ECSMiner) [13] lida com o fluxo de dados dividindo o *dataset* em blocos de igual tamanho. Cada bloco é usado para treinar um modelo de classificação. O algoritmo cria grupos por meio do algoritmo K-

Means. Depois de construir os grupos, o algoritmo armazena informações de cada grupo (centróide e frequências das amostras pertencentes a cada classe) em uma estrutura de dados chamada micro-grupo. Uma amostra de teste x_j é classificada buscando o micro-grupo cujo centróide está mais próximo de x_j e atribui a ele um rótulo de classe que possui a frequência mais alta nesse micro-grupo. Amostras que não estão dentro dos limites de nenhum grupo são tratadas como possíveis novas classes e colocadas em um *buffer*, que funciona como uma memória temporária. Essas amostras são chamadas de *Foutliers*. A partir daí, uma medida chamada *q-neighborhood silhouette coefficient* (q-NSC) é definida para decidir se as amostras armazenadas são novas classes. O q-NSC funciona como uma medida de coesão entre as amostras, medindo quão bem cada uma se encaixa em seu próprio grupo em comparação com os grupos vizinhos mais próximos. A atualização do modelo é realizada avaliando o erro de cada um dos modelos existentes no último bloco rotulado e descartando aquele com maior erro.

O Algoritmo 1 ilustra uma visão geral do ECSMiner. Neste algoritmo, considere B o *buffer* temporário, U o *buffer* de dados não rotulados e D_{init} o *buffer* de dados rotulados. Cada amostra x_j inserida no FCD é classificada pelo *ensemble* f. Caso U exceda um limite de tempo T_l, a amostra mais antiga x_k recebe seu rótulo e é adicionada ao *buffer* rotulado D_{init}, visto que já que T_l unidades de tempo decorreram desde que x_k chegou ao fluxo. Isso ocorre porque o ECSMiner assume que o rótulo de classe de cada amostra estará disponível após o tempo T_l. Quando D_{init} atinge uma quantidade predeterminada S, a função $TreinaSalvaFronteiraDecisao$ atualiza o modelo com os dados rotulados em D_{init}.

O Algoritmo 2 detalha a técnica de classificação referenciada no Algoritmo 1. Nessa abordagem, se a amostra x_j não for identificada como *outlier*, ela é classificada diretamente pelo modelo f. Caso contrário, é armazenada em B. Quando B acumula mais que q elementos e a última verificação $ultima\_verificacao$ em B para detectar novas classes foi executada pelo menos q unidades de tempo antes, $DetectaNovaClasse$ avalia a possibilidade da existência de uma nova classe. Se confirmada, a função $RemoveNovos$ remove as amostras identificadas como nova classe de B e o modelo é atualizado para acomodar a nova classe.

2.3.2 MINAS

O *MultI-class learnNing Algorithm for data Streams* (MINAS) [10], implementa aprendizado supervisionado em uma fase de treinamento *offline*, onde um modelo de decisão é criado. Nessa fase, a porção de treino do *dataset* é dividida em subconjuntos, cada um deles com as amostras de uma mesma classe. Para cada subconjunto, é gerado um conjunto de micro-grupos. Na etapa *online*, se uma amostra é identificada pelo modelo,

Algorithm 1 ECSMiner - Visão Geral

$f \leftarrow ensemble - inicial()$
$B \leftarrow \emptyset$; ▷ *buffer* temporário
$U \leftarrow \emptyset$; ▷ *buffer* de dados não rotulados
$D_{init} \leftarrow \emptyset$; ▷ *buffer* de dados rotulados $//x_j$: amostra de teste
while *true* **do**
  Classifica(f, x_j, B); ▷ Detalhado no Algoritmo 2
  $U \leftarrow U \cup x_j$;
  if $(\|U\| > T_l)$ **then** ▷ tempo para rótular a amostra mais antiga
    $x_k \Leftarrow U$; ▷ Desempilha x_k de U
    ▷ Rotula a amostra mais antiga e adiciona ao *buffer* de dados rotulados:
    $D_{init} \leftarrow D_{init} \cup < x_k, y_k >$;
    if $(\|D_{init}\| = S)$ **then** ▷ *buffer* cheio
      $f' \leftarrow TreinaSalvaFronteiraDecisao(D_{init})$;
      $f \leftarrow Atualize(f, f', D_{init})$;
      $D_{init} \leftarrow \emptyset$;
    end if
  end if
end while

Algorithm 2 ECSMiner - Classifica

Entrada: f, x_j, B
▷ f: *ensemble* dos atuais melhores M classificadores
▷ x_j: amostra de teste
▷ B: *buffer* que armazena amostras para futura avaliação
$fout \leftarrow true$
if $(Foutlier(f, x_j) = false)$ **then**
  $y'_i \leftarrow VotacaoMajoritaria(f, x_j)$; ▷ classifique imediatamente
  $fout \leftarrow false$
end if
Filter(B)
if $(fout = true)$ **then**
  $B \leftarrow x_j$; ▷ empilha x_j
  if $(B.tamanho > q)$ e $(ultima\_verificacao + q \leq t_i)$ **then**
    $ultima\_verificacao \leftarrow t_i$
    $novaclasse \leftarrow DetectaNovaClasse(f, B)$
    if $(novaclasse = true)$ **then**
      $RemoveNovos(B)$
    end if
  end if
end if

ela é classificada em uma das classes conhecidas. Caso contrário, é armazenada em um *buffer* para análise futura. Quando esse *buffer* alcança um determinado tamanho, um algoritmo de agrupamento é aplicado nas amostras armazenadas, criando novos micro-grupos. A partir daí, é aplicada uma medida de coesão para decidir se um determinado micro-grupo pode ser classificado como uma extensão de uma classe conhecida ou se pode ser considerado uma nova classe. Em caso positivo, os micro-grupos são adicionados ao modelo de decisão atual. Os micro-grupos que não atenderem aos critérios estabelecidos são descartados, sendo considerados ruídos ou *outliers*.

O Algoritmo 3 ilustra a etapa *offline*. Essa etapa gera k micro-grupos para cada classe, usando $alg\_agrupamento$ para dividir D_{init} em subconjuntos de uma única classe. Cada micro-grupo criado contém informações sobre o centróide, o raio, o rótulo de classe e o *timestamp* da última amostra incluída no micro-grupo. A fronteira de decisão de cada classe é estabelecida pela união dos micro-grupos. O modelo de decisão inicial f é composto pela união dos k micro-grupos obtidos para cada classe.

Algorithm 3 MINAS - Fase de Treinamento (*offline*)

Entrada: k, $alg\_agrupamento$, D_{init}
$f \leftarrow \emptyset$;
for $classeC_i \in D_{init}$ **do**
  $f\_Tmp \leftarrow Agrupamento(D_{init\_ClasseCi}, k, alg\_agrupamento)$;
  for $classeC_i \in D_{init}$ **do**
    $micro.rotulo \leftarrow C_i$
  end for
  $f \leftarrow f \cup f\_Tmp$;
end for
retorna f

A fase *online* é ilustrada pelo Algoritmo 4. Nessa fase, para cada amostra não rotulada que chega, o algoritmo verifica se ela pode ser explicada pelo modelo de decisão atual. Para isso, a distância euclidiana entre a amostra x_j e os micro-grupos em f é calculada. Se x_j estiver dentro do raio de um micro-grupo, é classificada com o rótulo desse grupo. Caso contrário, x_j é adicionada ao *buffer* B, e, se B atingir um número mínimo de amostras, $DeteccaoNovidade$ verifica a existência de uma nova classe. Grupos inativos são movidos para $MemSleep$ para otimizar o uso de memória.

2.3.3 DT Novel Class

A técnica apresentada em Farid & Rahman (2012) [27], aqui referenciada por DT Novel Class, é baseada em um classificador de AD. Depois de criar a AD a partir do conjunto de dados de treinamento, são criados grupos baseados na similaridade de atributos para

Algorithm 4 MINAS - Fase de Aplicação (*online*)

Entrada: $f, T, NumMinAmostras, ts, P$
$B \leftarrow \emptyset$;
$MemSleep \leftarrow \emptyset$;
while *true* **do**
  $(Dist, micro) \leftarrow MicroMaisProximo(x_j, f)$;
  if $(Dist < raio(micro))$ **then**
    $x_j.classe \leftarrow micro.rotulo$;
    $AtualizarMicro(micro, x_j)$;
  else
    $x_j.classe \leftarrow desconhecido$;
    $B \leftarrow B \cup x_j$;
    if $(\|B\| \geq NumMinAmostras)$ **then**
      $(f) \leftarrow DeteccaoNovidade(f, B, T)$;
    end if
  end if
  $TempoAtual \leftarrow x_j.T$;
  if $(TempoAtual \mod TamJanela == 0)$ **then**
    $(f) \leftarrow MoverMicroMemSleep(f, MemSleep, P)$;
    $(B) \leftarrow RemoverAmostrasAntigas(B, ts)$;
  end if
end while

as amostras de cada nó folha na árvore. Um *threshold* é calculado como uma razão entre o número total de amostras no nó folha e o número total de amostras no conjunto de treinamento. Para cada amostra de teste no fluxo, se sua adição ao nó folha aumentar o *threshold* e se esta amostra não pertencer a nenhum grupo do nó folha, ela será considerada uma amostra de nova classe. Em seguida, são realizados procedimentos de atualização do modelo.

2.3.4 SENCForest

O SENCForest [14] é um algoritmo baseado no detector de anomalia não supervisionado Isolation Forest [53], que são *ensembles* de um tipo de AD chamada de completamente aleatória [54]. Depois de construir uma iForest, o comprimento do caminho (número médio de arestas percorridas pela amostra do nó raiz para um nó folha) é usado como uma métrica para diferenciar o espaço de dados em duas regiões: região normal e regiões de anomalia. Amostras cujo comprimento do caminho é menor do que um *threshold* escolhido definem a região normal. A região de anomalia, por sua vez, é subdividida em duas subregiões: a subregião de anomalias de classes conhecidas e a subregião de *outliers*. As amostras de teste que se enquadram na região de *outliers* são consideradas exemplares de novas classes e são armazenadas em um *buffer*.

Após a construção do detector de novas classes, os rótulos das classes conhecidas são registrados na região normal ou na região de anomalias das classes conhecidas. Cada amostra de teste que se enquadra nessas sub-regiões é classificada assumindo rótulos da classe majoritária daquela região. O processo de atualização do modelo começa quando o *buffer* está cheio. Usando amostras do *buffer*, o mesmo processo de crescimento da árvore é então aplicado a cada folha de cada árvore existente até que o critério de parada seja satisfeito. Vale ressaltar que todas as amostras avaliadas são classificadas como novidade ou dentre alguma das classes conhecidas. Desta forma, não há amostras classificadas como "desconhecidas", como visto em outras abordagens. Os Algoritmos 5 e 6 resumem o funcionamento do SENCForest.

Na etapa *offline*, o algoritmo constrói uma floresta de árvores f para DN. Cada árvore $SENCTree$ é treinada com uma subamostra X_{sub} do *dataset* D. O parâmetro z define o número de árvores, enquanto ψ representa o tamanho de cada subamostra. Ao final, a floresta é retornada e usada na fase *online* para detectar novas classes.

Algorithm 5 SENCForest - Fase de Treinamento (*offline*)

Entrada: D - *dataset*, z - numero de árvores, ψ - tamanho da subamostra
Saida: SENCForest
$f \leftarrow \emptyset$;
for $sub = 1, 2, \ldots, z$ **do**
    $X_{sub} \leftarrow subamostra(D; \psi)$;
    $f \leftarrow f \cup SENCTree(X_{sub})$;
end for
retorna f

Conforme ilustrado no Algoritmo 6, cada amostra é classificada por f. Se for identificada como uma nova classe, é armazenada no *buffer* B. Quando B atinge o limite s, as amostras são usadas para atualizar a floresta, o *buffer* é esvaziado e o número m de classes conhecidas até o momento é atualizado.

2.3.5 Higia

Em Garcia *et al.* (2019) [48], os autores propuseram uma técnica denominada Higia. Assim como nos algoritmos já citados, o algoritmo tem uma fase *offline* onde é realizado um treinamento com amostras das classes conhecidas. As amostras de cada classe conhecida são agrupadas em micro-grupos criados usando o algoritmo CluStream [55] e cada um deles só possui amostras de uma classe em particular, de modo semelhante ao MINAS. Durante a etapa *online*, o modelo classifica cada amostra em conhecida, extensão ou desconhecida. As amostras definidas como extensões de um micro-grupo são consideradas pela técnica proposta como uma mudança de conceitos. Amostras definidas como

Algorithm 6 SENCForest - Fase de Aplicação (*Online*)

Entrada: f, B - *buffer* de tamanho s
Saida: Classe y_j de cada amostra x_j do fluxo
while *true* **do**
  $y_j \leftarrow f(x_j)$
  if $(y_j = novaclasse)$ **then**
    $B \leftarrow B \cup \{x_j\}$;
    if $(\|B\| \geq s)$ **then** ▷ *buffer* cheio
      $Atualize(f(B))$;
      $B \leftarrow \emptyset$;
      $m \leftarrow m + 1$;
    end if
  end if
  return y_j
end while

desconhecidas são armazenadas em um *buffer* temporário. As amostras do *buffer* que fazem parte de um conjunto coeso e representativo são consideradas amostras de uma nova classe. Quando uma nova classe é detectada, são criados novos micro-grupos para representá-la.

2.3.6 KNNENS

O *k-Nearest Neighbor Ensemble-Based* (KNNENS) [12] é um método que utiliza um *ensemble* de classificadores baseados em KNN para detecção multiclasse de novas classes em FCD. A abordagem explora informações de vizinhança local através de hiperesferas construídas a partir de vizinhos mais próximos da mesma classe. O método opera sem necessidade de rótulos verdadeiros durante a fase *online*, realizando simultaneamente classificação de classes conhecidas e detecção de novas classes. O modelo mantém tamanho fixo através de um mecanismo de substituição que preserva apenas uma versão da nova classe detectada, eliminando versões anteriores durante atualizações subsequentes. Assim, como no SENCForest, todas as amostras avaliadas são classificadas como novidade ou dentre alguma das classes conhecidas. Desta forma, não há amostras classificadas como "desconhecidas", como visto em abordagens como o MINAS.

O Algoritmo 7 ilustra a etapa *offline*. Nessa fase, o algoritmo realiza b amostragens *bootstrap*, selecionando a amostras sem reposição de cada classe conhecida em cada iteração. Para cada subconjunto D_i gerado, constrói-se um *ensemble* de hiperesferas H_i, onde cada hiperesfera é centrada em uma amostra $c \in D_i$ e possui raio $r(c)$ calculado como a média das distâncias aos L vizinhos mais próximos da mesma classe. O modelo inicial H

é composto pela união de todos os *sub-ensembles* H_i, totalizando $K \times a \times b$ hiperesferas. Cada *sub-ensemble* H_i contém informações de todas as K classes conhecidas.

Algorithm 7 KNNENS - Fase de Treinamento (*offline*)

Entrada: a, b, L, D_{init} com K classes
$H \leftarrow \emptyset$; ▷ *Ensemble* de hiperesferas
for $i = 1$ **to** b **do**
  $D_i \leftarrow \emptyset$;
  for $k = 1$ **to** K **do**
    $D_i^{(k)} \leftarrow SortearSemReposicao(D_{init\_ClasseC_k}, a)$;
    $D_i \leftarrow D_i \cup D_i^{(k)}$;
  end for
  $H_i \leftarrow ConstruirHiperesferas(D_i, L)$;
  $H \leftarrow H \cup H_i$;
end for
retorna H

A fase *online* é ilustrada pelo Algoritmo 8. Para cada amostra x_j, o algoritmo calcula a razão entre a distância ao centro de cada hiperesfera e seu respectivo raio. Se essa razão exceder o limiar t para todas as hiperesferas de um *sub-ensemble* H_i, o sub-modelo vota como nova classe; caso contrário, classifica x_j via vizinho mais próximo usando o rótulo da hiperesfera de menor distância. A predição final $f(x_j)$ é obtida por votação majoritária entre os b *sub-ensembles*. Amostras classificadas como nova classe são adicionadas ao *buffer* B. Quando $|B|$ atinge o limite pré-definido, o módulo de atualização é acionado: para cada *sub-ensemble* H_i, hiperesferas antigas rotuladas como "NC"(*new class*) são removidas, novas hiperesferas são construídas usando a amostras sorteadas do *buffer*, e o *buffer* é esvaziado. Este mecanismo de substituição garante que o modelo mantenha tamanho fixo de $(K + 1) \times a \times b$ hiperesferas após a primeira detecção de nova classe.

2.3.7 LC-INC

O *Learning to Classify with Incremental New Class* (LC-INC) [56] propõe uma solução unificada para detecção de novas classes emergentes e expansão de modelo em ambientes de aprendizado incremental. O método se inicia com um conjunto de classes conhecidas cujas amostras são rotuladas. Conforme novas amostras chegam no fluxo, a rede de estrutura (*structure network*) compara essas amostras com centros prototípicos das classes conhecidas para detectar possíveis novos conceitos. Notavelmente, a rede integra informação de protótipo (centros das classes conhecidas) com a predição do classificador para decidir se uma amostra pertence a uma classe conhecida ou representa uma nova classe emergente. Quando suficientes amostras suspeitas de representar uma nova classe são acumuladas, LC-INC utiliza esse conjunto para expandir o modelo, criando uma nova

Algorithm 8 KNNENS - Fase de Aplicação (*online*)

Entrada: H com b *sub-ensembles* H_i, t, $TamBuffer$
$B \leftarrow \emptyset$; ▷ *Buffer* de desconhecidos
while *true* **do**
  for $i = 1$ **to** b **do**
    $votos \leftarrow \emptyset$;
    for $hiperesfera\, h(c, r(c)) \in H_i$ **do**
      if $(\|x_j - c\|/r(c) > t)$ **then**
        $votos \leftarrow votos \cup \{NC\}$;
      else
        $votos \leftarrow votos \cup \{rotulo(h)\}$;
      end if
    end for
    $f'_i(x_j) \leftarrow$ *VotoMaisFrequente*$(votos)$;
  end for
  $f(x_j) \leftarrow$ *VotacaoMajoritaria*$(\{f'_1(x_j), ..., f'_b(x_j)\})$;
  if $(f(x_j) == NC)$ **then**
    $B \leftarrow B \cup x_j$;
    if $(\|B\| \geq TamBuffer)$ **then**
      $(H) \leftarrow$ *AtualizarModelo*(H, B, a, L);
      $B \leftarrow \emptyset$;
    end if
  end if
end while

classe com base nas representações aprendidas, sem requerer rótulos externos para essas amostras emergentes. O modelo também funciona como uma meta-rede que aprende a estrutura geral de expansão, de modo que se adapta com relativamente poucas amostras novas, reduzindo a sobrecarga de treinamento.

2.3.8 SNDProb

O *Streaming Novelty Detection based on Probabilistic approach* (SNDProb) [57] propõe um *framework* probabilístico para DN em FCD que modela o conjunto de classes conhecidas através de uma mistura de distribuições Gaussianas. As amostras são preditas com base na probabilidade de pertencerem a cada uma das classes modeladas, e aquelas para as quais o modelo não consegue fornecer predições confiáveis são inseridas em um *buffer* de tamanho fixo. Quando o *buffer* está completo, um algoritmo Expectation Maximization (EM) é executado para buscar novas classes emergentes e atualizar o modelo corrente. O algoritmo EM enfrenta o cenário desafiador onde tanto distribuições de probabilidade (classes conhecidas) quanto amostras (candidatas a novas classes) estão disponíveis simultaneamente, sendo necessário ponderar as distribuições existentes durante o processo de aprendizado. Para isso, os pesos são inferidos utilizando um modelo de meta-regressão pré-treinado, composto por uma hierarquia em cascata onde um classificador Random Forest diferencia entre cenários de *concept drift* e emergência de novas classes, seguido por um classificador KNN que seleciona o conjunto adequado de pesos para as componentes da mistura.

2.3.9 TRIDENT

O Trident é uma técnica que utiliza *learners one-class* (autoencoders, RNNs ou GNNs), onde cada *learner* é treinado exclusivamente com dados de uma única classe conhecida. Durante a fase *online*, amostras são classificadas com base no *reconstruction loss*: se o *loss* de pelo menos um *learner* estiver abaixo de seu limiar, a amostra recebe o rótulo da classe mais similar; caso contrário, vai para um *buffer* de novidades. Periodicamente, *clustering ensemble* é aplicado ao *buffer* para identificar e distinguir novas classes. A atualização incremental ocorre criando novos *learners* para as classes detectadas, sem necessidade de retreinar os existentes. No Trident todas as amostras avaliadas são classificadas como novidade ou dentre alguma das classes conhecidas. Desta forma, não há amostras classificadas como "desconhecidas".

O Algoritmo 9 ilustra a etapa *offline*. Para cada classe conhecida C_i, um *learner* one-class L_i é treinado exclusivamente com amostras dessa classe, utilizando $alg\_learner$ (AutoEncoder, RNN ou GNN). Cada *learner* aprende a reconstruir apenas amostras de

sua classe, gerando baixo *reconstruction loss* para amostras similares. Em seguida, o módulo tScissors determina o limiar θ_i para cada *learner*, usando EVT (*Extreme Value Theory*) ou método preset, definindo a fronteira de aceitação. O modelo inicial f é composto pelo conjunto de *learners* $\{L_1, L_2, ..., L_n\}$ e seus respectivos limiares $\{\theta_1, \theta_2, ..., \theta_n\}$.

Algorithm 9 Trident - Fase de Treinamento (*offline*)

Entrada: $alg\_learner$, $metodo\_limiar$, D_{init}
$L_p \leftarrow \emptyset$; ▷ Lista de *learners*
$L_t \leftarrow \emptyset$; ▷ Lista de limiares
for $classeC_i \in D_{init}$ **do**
    $L_i \leftarrow TreinarLearner(D_{init\_ClasseCi}, alg\_learner)$;
    $losses \leftarrow CalcularLosses(L_i, D_{init\_ClasseCi})$;
    $\theta_i \leftarrow DeterminarLimiar(losses, metodo\_limiar)$;
    $L_p \leftarrow L_p \cup \{L_i\}$;
    $L_t \leftarrow L_t \cup \{\theta_i\}$;
end for
retorna L_p, L_t

A fase *online* é ilustrada pelo Algoritmo 10. Para cada amostra x_j que chega, o algoritmo calcula o *reconstruction loss* usando todos os *learners* em L_p. Se pelo menos um *learner* produz $loss < \theta_i$, a amostra é classificada com o rótulo do *learner* de menor *loss* (conjunto candidato M_c). Caso contrário, x_j é adicionada ao *buffer* B. Quando B atinge tamanho suficiente, o módulo tMagnifier aplica *clustering ensemble* (combinando K-means, DBSCAN e DEC com múltiplos parâmetros) para atribuir rótulos *fine-grained* às novas classes. Para cada novo *cluster* detectado, um novo *learner one-class* é criado e adicionado ao modelo via atualização incremental.

2.4 Métricas de Avaliação

A avaliação de algoritmos em FCD deve envolver critérios complementares, pois esses métodos precisam aprender e realizar predições de forma sequencial, sob restrições do ambiente onde estão implementados. Em particular, além de medir a qualidade das predições ao longo do fluxo, é necessário caracterizar o consumo de recursos e a capacidade do algoritmo de acompanhar a taxa de chegada das amostras. Nesse sentido, Gama *et al.* (2008) [58] discutem que a análise em fluxos deve considerar, de forma conjunta, (i) o desempenho preditivo do modelo, (ii) o uso de memória ao longo do tempo e (iii) a velocidade de processamento das amostras, uma vez que limitações de memória e tempo são parte inerente do cenário de aprendizado em fluxo.

Em Gaudreault & Branco (2024) [26], os autores realizam uma revisão sistemática sobre métodos de DN em FCD, reforçando a necessidade de processar dados em alta ve-

Algorithm 10 Trident - Fase de Aplicação (*online*)

Entrada: $L_p, L_t, NumMinAmostras, Num$
$B \leftarrow \emptyset$; ▷ *Buffer* de desconhecidos
$Result \leftarrow \emptyset$;
while *true* **do**
  $L_r \leftarrow \emptyset$; ▷ Lista de *losses*
  for $learner_i \in L_p$ **do**
    $l \leftarrow ReconstructionLoss(learner_i, x_j)$;
    $L_r \leftarrow L_r \cup \{l\}$;
  end for
  $M_c \leftarrow \{i \mid L_r[i] < L_t[i]\}$; ▷ Candidatos aceitos
  if $(\|M_c\| == 0)$ **then**
    $B \leftarrow B \cup x_j$;
    if $(\|B\| \geq NumMinAmostras)$ **then**
      $R_u \leftarrow ClusteringEnsemble(B, Num)$;
      $Y \leftarrow RotulosDetectados(R_u)$;
      $(L_p, L_t) \leftarrow AtualizacaoIncremental(Y, B, L_p, L_t)$;
      $Result \leftarrow Result \cup R_u$;
    end if
  else
    $j^* \leftarrow \arg\min_{j \in M_c} L_r[j]$;
    $x_j.classe \leftarrow rotulo(learner_{j^*})$;
    $Result \leftarrow Result \cup x_j.classe$;
  end if
end while
retorna *Result*

locidade para evitar atrasos, adotar estratégias para otimizar o uso de memória e manter alta acurácia. Com base nesses aspectos, esta seção apresenta métricas consideradas relevantes para avaliação de algoritmos de DN em FCD, organizadas em capacidade preditiva, uso de memória e taxa de processamento de amostras.

2.4.1 Avaliação da Capacidade Preditiva

Diversas métricas são utilizadas para a avaliação da capacidade preditiva de algoritmos de classificação. Elas oferecem medidas quantitativas para avaliar a eficácia dos modelos, auxiliando os pesquisadores na avaliação, comparação e melhoria de algoritmos. Algumas delas são precisão, revocação, acurácia, F-measure, Kappa de Cohen e AUC-ROC [59, 60].

De uma maneira geral, em problemas de classificação multiclasse com N classes, a matriz de confusão é uma matriz quadrada NxN. A Figura 2.4 ilustra uma matriz de confusão de um problema de classificação com classes C_1, C_2 e C_3, onde cada coluna representa uma das classes preditas pelo algoritmo. Os elementos da diagonal principal, em destaque, representam as predições corretas.

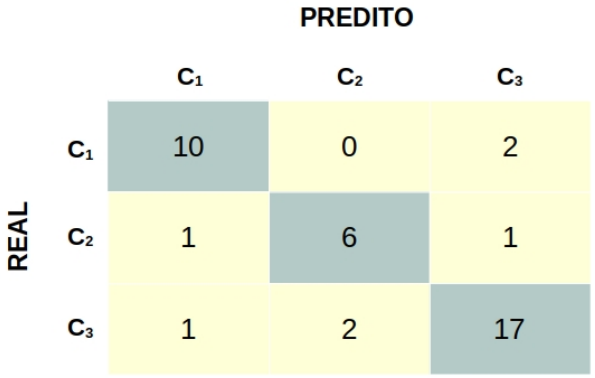

Figura 2.4: Matriz de Confusão Multiclasse.

Nos problemas de DN em FCD, a matriz de confusão adquire características dinâmicas e incrementais: à medida que o número de classes cresce, o número de linhas e colunas da matriz aumenta, o que torna sua forma não necessariamente quadrada. O surgimento de novas classes requer a adição de novas colunas (representando classes detectadas) e linhas (novas classes adicionadas ao modelo de decisão). É possível também incluir uma coluna para amostras classificadas como desconhecidas, representando aquelas que o algoritmo não conseguiu alocar em nenhuma classe conhecida, como na abordagem proposta por De Faria *et al.* (2015) [61].

A Figura 2.5 representa uma matriz de confusão em um problema de classificação multiclasse com surgimento de novas classes. As linhas $CNov_1$ até $CNov_q$ representam as

novas classes presentes no fluxo. Vale observar que essas classes são novidades sem rótulo, visto que o algoritmo detecta a existência de um novo conceito, mas não possui, *a priori*, informação sobre sua categoria real. Por sua vez, as colunas Nov_1 até Nov_p representam as novas classes identificadas pelo algoritmo. Deve ser observado que não necessariamente $p = q$, ou seja, o número de novas classes preditas pelo algoritmo não necessariamente será igual ao número de novas classes existentes no fluxo. Além disso, se o algoritmo de classificação em questão for capaz de identificar apenas uma nova classe por vez, então $p = 1$.

Adicionalmente, um aspecto fundamental na avaliação de algoritmos de DN multi-classe é o tratamento das amostras classificadas temporariamente como desconhecidas (*unknown*). Segundo De Faria *et al.* (2015) [61], a categoria *desconhecido* pode ser utilizada para representar amostras que não são explicadas pelo modelo atual, mas que podem ser futuramente utilizadas para modelar novos conceitos ou extensões de classes conhecidas. Conforme discutido na Seção 2.3, em diversas abordagens essas amostras são armazenadas em um *buffer* de espera até que uma massa crítica de dados similares permita a identificação de um novo padrão de novidade ou a atualização de uma fronteira existente

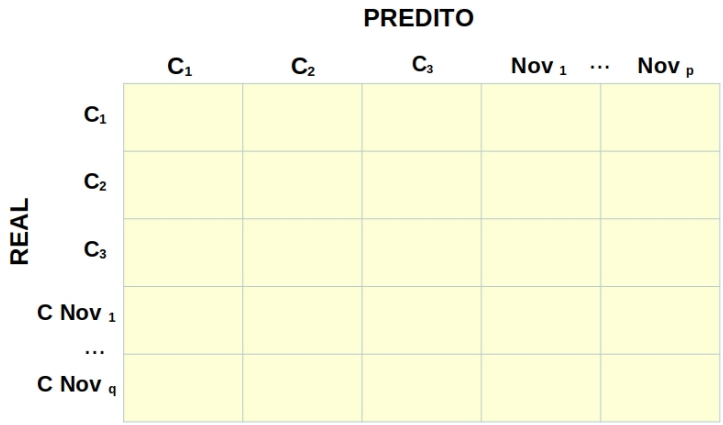

Figura 2.5: Matriz de confusão com surgimento de novas classes.

Outra observação que pode ser feita em relação a essa matriz é que em um cenário de surgimento de classes há outros possíveis erros de predição. É possível que uma amostra de uma nova classe seja erroneamente classificada como de uma das classes conhecidas. É possível também que o modelo empregado classifique amostras de classes conhecidas como novas. Percebe-se, portanto, que além das métricas tradicionais citadas, são necessárias métricas específicas para avaliar a eficácia desses algoritmos na tarefa de distinguir entre amostras de classes conhecidas e de novas classes. Algumas das métricas mais comuns em um cenário com surgimento de novas classes são o M_{new} e o F_{new} [20, 10, 13].

O M_{new}, apresentado na Equação (2.1), representa o percentual de novas classes erroneamente classificadas como uma das classes conhecidas. Nesta equação, F_n representa o número de amostras de classes novas que estão incorretamente classificadas como classes conhecidas e N_c é o total de amostras de novas classes.

$$M_{new} = \frac{F_n}{N_c} \tag{2.1}$$

O F_{new}, apresentado na Equação (2.2), mostra o percentual de classes conhecidas erroneamente classificadas como novas. Nesta equação, F_p indica o número de amostras de classes conhecidas que foram identificadas incorretamente como uma nova classe e N é o total de amostras no fluxo.

$$F_{new} = \frac{F_p}{N - N_c} \tag{2.2}$$

É válido observar que, de acordo com a definição das equações (2.1) e (2.2), não há a necessidade de que o algoritmo tenha a capacidade de diferenciação entre duas novas classes distintas para que um acerto seja computado. Esta característica pode ser considerada uma limitação desta métrica.

Seja F_e o total de amostras que são identificadas corretamente como sendo de classes conhecidas, mas com erro na identificação da classe. É possível computar o erro total de classificação do algoritmo por meio da Equação (2.3).

$$Erro = \frac{F_p + F_n + F_e}{N} \tag{2.3}$$

Outra métrica que pode ser registrada é a acurácia geral do modelo, apresentada na Equação (2.4), que considera as predições corretas das classes conhecidas e as identificações assertivas de novas classes. Nessa equação, A_c é o número de amostras de novas classes identificadas como sendo de novas classes, A_0 é o número de amostras de classes conhecidas classificadas corretamente na sua classe real e N é o total de amostras do fluxo.

$$Acurácia = \frac{A_c + A_0}{N} \tag{2.4}$$

2.4.2 Avaliação do Consumo de Memória Principal

Considerando as limitações de recursos de quaisquer ativos de rede, a capacidade de gerenciar o consumo de memória principal por parte do algoritmo é um fator determinante para a aplicabilidade das técnicas em dispositivos de detecção de intrusão. Em particular, em um FCD há uma sequência potencialmente infinita de dados que precisam ser pro-

cessados pelo dispositivo, o que inviabiliza o armazenamento integral do histórico e exige que o modelo opere com memória limitada e uso controlado ao longo do tempo.

2.4.3 Avaliação da Taxa de Avaliação de Amostras

Considerando também que em um FCD as amostras podem chegar em alta velocidade, a taxa de processamento destas amostras também é uma característica importante de algoritmos utilizados neste contexto. Esses algoritmos devem ter uma capacidade de realizar todos os passos previstos em seu método preditivo em um tempo aceitável para a aplicação. Se o algoritmo demorar muito para avaliar as amostras, ele não será capaz de acompanhar a taxa de chegada dos dados, resultando em perdas de informações importantes ou aumento do tempo de latência. Em aplicações de detecção de intrusão, uma baixa eficiência na tarefa de predição dos comportamentos suspeitos pode gerar um atraso na detecção de ataques cibernéticos.

2.5 Considerações Finais

Este capítulo apresentou os conceitos fundamentais que sustentam esta pesquisa. Foram introduzidos os IDS e suas abordagens baseadas em assinatura e em anomalia, com ênfase no uso de AM para a classificação de tráfego de rede. Em seguida, foram discutidos os desafios inerentes aos FCD, como o processamento de sequências potencialmente infinitas de dados sob restrições de memória e tempo. O capítulo também caracterizou o problema do surgimento de novas classes em FCD e apresentou as principais técnicas de DN desenvolvidas para tratá-lo, descrevendo suas fases offline e online e seus mecanismos de atualização de modelo. Por fim, foram introduzidas as métricas de avaliação adotadas nesta pesquisa, incluindo acurácia, M_{new} e F_{new} e as métricas de eficiência computacional.

O próximo capítulo apresenta os principais trabalhos relacionados a esta pesquisa, discutindo estudos que aplicam técnicas de DN em FCD ao problema de detecção de intrusão e identificando as lacunas que esta dissertação busca endereçar.

Capítulo 3

Trabalhos Relacionados

3.1 Revisão da Literatura

Técnicas de AM, como SVM, RNA, Naive Bayes e AD, são amplamente empregadas em IDS por sua capacidade de extrair padrões a partir de dados históricos e apoiar a discriminação entre tráfego benigno e categorias de ataques. Em geral, esses métodos são aplicados sobre conjuntos de atributos derivados do tráfego (por exemplo, estatísticas de fluxos ou contadores agregados), reduzindo a necessidade de regras explícitas e permitindo a construção de classificadores com bom desempenho em cenários controlados [3, 62, 63, 64, 65, 66].

Trabalhos representativos nessa linha comparam modelos de classificação treinados de forma supervisionada em bases de referência (por exemplo, NSL-KDD e ToN-IoT), frequentemente combinando técnicas de seleção/otimização de atributos (como ganho de informação ou correlação) para reduzir dimensionalidade e melhorar o desempenho [3, 64, 5, 8]. Em termos gerais, os resultados reportados nesses estudos indicam que abordagens como Random Forest, SVM, KNN e RNA podem atingir níveis elevados de acurácia quando avaliadas em partições estáticas de dados rotulados, reforçando o potencial do AM para IDS em cenários onde o conjunto de ataques é conhecido e bem representado na base de treinamento.

Alguns trabalhos buscam realizar a detecção de intrusão por meio de abordagens de classe única, onde os modelos são treinados com apenas uma classe de dados [67, 68, 69, 70, 71, 72]. No contexto de IDS, essa classe única normalmente representa o tráfego benigno. Assim, as categorias de tráfego que forem diferentes do que foi aprendido são consideradas tráfego malicioso. Os trabalhos citados utilizam majoritariamente os algoritmos OCSVM, *Isolation Forests* (iForests), *Local Outlier Factor* (LOF) e *autoencoders* para identificar o tráfego malicioso. No entanto, esta abordagem, de forma isolada, não tem a capacidade

de diferenciar os ataques entre si, visto que esses ataques serão considerados todos como anomalias da classe benigna [73].

Embora os resultados da aplicação de algoritmos de AM em IDS sejam promissores, as estratégias discutidas baseiam-se predominantemente no treinamento em lotes (*batch*), no qual o modelo é induzido a partir do processamento integral de bases de dados estáticas. Apesar de eficaz para o reconhecimento de comportamentos maliciosos previamente mapeados, essa abordagem limita a capacidade preditiva às categorias presentes no treinamento, tornando o sistema incapaz de identificar ameaças inéditas que emergem com a evolução das redes. Segundo Miani *et al.* (2025) [36], tal paradigma estático é insuficiente para gerenciar fenômenos críticos, como a mudança na distribuição do tráfego e a escassez de rótulos para atualizações oportunas. Nesse sentido, os autores enfatizam a necessidade de reestruturar o processo de descoberta de conhecimento para o cenário de FCD, distinguindo os IDS convencionais daqueles que empregam algoritmos incrementais capazes de se adaptar dinamicamente à chegada constante de novas amostras.

Essa mudança de paradigma exige que o sistema de detecção seja capaz não apenas de manter o desempenho frente a mudanças graduais, mas de identificar e incorporar novos conceitos de forma autônoma conforme o tráfego evolui. Nesse cenário, a capacidade de DN torna-se valiosa para a segurança cibernética, permitindo que o classificador detecte explicitamente ameaças inéditas sem a necessidade de intervenção manual constante.

Embora vários estudos investiguem DN multiclasse em FCD, a maioria tende a adotar uma abordagem de avaliação mais generalista, realizando experimentos em diversos domínios. Uma exploração aprofundada de campos de pesquisa específicos é menos comum. Além disso, muitos trabalhos avaliam apenas o desempenho preditivo, sem tratar adequadamente a eficiência computacional, como o consumo de memória e as taxas de avaliação de amostras. A ausência de *datasets* de IDS mais atuais e de avaliações mais abrangentes nos estudos dificulta a comparação eficaz entre algoritmos. Essas lacunas destacam a necessidade de pesquisas que integrem conjuntos de dados mais realistas, equilibrem o desempenho preditivo com restrições computacionais e forneçam entendimentos para a implementação prática de técnicas de DN.

Revisões abrangentes sobre DN em FCD reforçam esse diagnóstico. Faria *et al.* (2016)[74] apresentam uma visão geral da DN em FCD, organizando a literatura segundo aspectos como a separação entre fases *offline* e *online*, o número de classes consideradas em cada fase, o uso de *ensembles* em contraste com classificadores únicos, abordagens supervisionadas e não supervisionadas, mecanismos de esquecimento, tratamento de *concept drift*, distinção entre ruído, *outliers* e novidades e estratégias para lidar com amostras de rótulo desconhecido e classes recorrentes. Os autores também descrevem aplicações em diferentes domínios e destacam desafios em aberto, como a falta de consenso sobre

protocolos de avaliação e métricas capazes de capturar todos os aspectos relevantes da tarefa. Estudos mais recentes, como o de Gaudreault e Branco (2024)[26], aprofundam essa discussão ao propor um arcabouço de avaliação para DN em FCD e evidenciar lacunas recorrentes, como a ausência de métricas temporais padronizadas e a dificuldade de comparação justa entre algoritmos em cenários próximos ao uso prático, reforçando a necessidade de estudos mais sistemáticos em domínios específicos, como detecção de intrusão.

No contexto específico de IDS, Miani *et al.* (2025) [36] apresentam um levantamento sistemático dos principais trabalhos que tratam tráfego de rede como FCD e utilizam algoritmos incrementais ou adaptativos. Os autores organizam a literatura deste tipo de IDS segundo uma taxonomia baseada nas etapas de KDD, destacando: (i) as fontes de dados mais empregadas (pacotes, fluxos e sessões), (ii) estratégias de pré-processamento e redução de dados, como seleção de atributos, amostragem e janelas deslizantes, (iii) tipos de modelos utilizados (classificadores incrementais, *ensembles*, métodos de detecção de anomalias) e (iv) esquemas de atualização e realimentação, que variam desde feedback total até cenários com feedback parcial ou inexistente. Essa taxonomia evidencia que, embora haja um número crescente de estudos de IDS, a maioria ainda se concentra em manter o desempenho frente a mudanças graduais no tráfego, sem explorar de forma sistemática a detecção explícita de novas classes de ataques.

Ao organizar os estudos por *datasets*, Miani *et al.* (2025) [36] mostram que a maioria das pesquisas em IDS ainda é avaliada em KDD99 ou NSL-KDD, apesar de ambos serem considerados obsoletos para os dias de hoje, uma vez que derivam de tráfego coletado em 1998 e não refletem o perfil contemporâneo de ataques e de serviços de rede. Os autores recomendam explicitamente o uso de bases mais recentes e mantidas, como os *datasets* do UNB CIC, UNSW-NB15 e coleções da Stratosphere Laboratory, como princípio para avaliações mais realistas de IDS.

A Tabela 3.1 resume estudos que incorporam *datasets* de detecção de intrusão em DN multiclasse. A tabela apresenta as publicações, técnicas avaliadas, *datasets* utilizados, estratégia de avaliação de eficiência computacional (quando realizada), métricas de desempenho preditivo e informação sobre a realização de otimização de hiperparâmetros nos experimentos. A abordagem mais comum entre os trabalhos em questão foi usar um único *dataset* de IDS, normalmente o KDDCup 99. ECSMiner, SENCForest e MINAS são as técnicas mais estudadas, destacando a importância destes trabalhos nesta área de pesquisa. Além das métricas clássicas de avaliação de desempenho, como acurácia e F1-Score, métricas específicas para medir a capacidade de identificar amostras de novas classes, como M_{new} e F_{new}, são comumente usadas. Em relaçao à otimização de hiperparâmetros nos experimentos, o termo "parcial"indica que o trabalho implementou esta

tarefa apenas para alguns algoritmos do estudo, não para todos.

Tabela 3.1: Detecção de Intrusão em Detecção de Novidade Multiclasse

Autor	**Técnicas Avaliadas**	***Datasets***	**Eficiência Computacional**	**Desempenho Preditivo**	**Otimização de hiperparâmetros**
[13]	ECSMiner e OLINDDA	KDDCup 99	Taxa de avaliação de amostras	M_{new}, F_{new} e $Erro$	Sim
[75]	CLAM, ECSMiner, OLINDDA-WCE, SCANR	KDDCup 99	-	M_{new}, F_{new}, ERR	Sim
[61]	MINAS, OLINDDA, ECSMiner e CLAM	KDDCup 99	-	CER e $UnkR$	Não
[14]	SENCForest, ECSMiner e CLAM	KDDCup 99	-	Acurácia e F1-Score	Parcial
[76]	Mukers, ECSMiner, CLAM, LOCE e SvsClass	KDDCup 99	Taxa de avaliação de amostras	M_{new}, F_{new} e $Erro$	Parcial
[28]	ECSMiner, MINAS e AnyNovel	Kyoto 2006+	Consumo de memória e taxa de avaliação de amostras	Accuracy e F1-Score	Sim
[77]	FROST e MINAS	UNSW-NB15	-	M_{new}, F_{new} e $Erro$	Não
[18]	SACCOS, AHT, ECSMiner, SENC-MaS e SAND-D	KDDCup 99	-	M_{new}, F_{new} e Acurácia	Parcial
[56]	LC-INC, mCANDIES, ECSMiner, CLAM, SENCForest, SENNE, CPL e DNN-Based	KDDCup 99	-	Acurácia e F1-Score	Sim
[19])	Trident, SENCForest e mais 15 métodos	USTC-TFC2016, ISCXTor2016, CIC-IDS2017, CIC-IDS2018 e CrossNet2021	-	ACC, AMI, Precision, Recall, F1-Score e AUC	Parcial

No ECSMiner, Masud *et al.* (2010) propõem um *ensemble* que incorpora explicitamente um mecanismo de DN e o avaliam sobre o KDD Cup 99, medindo não apenas M_{new}, F_{new} e o erro total de classificação, mas também a taxa de avaliação de amostras, de forma a verificar se o algoritmo atende a restrições de tempo real impostas por fluxos de alta velocidade.

Al-Khateeb *et al.* (2012) [75] avaliaram o CLAM em *datasets* sintéticos (SynC10, SynC20, SynC40) e reais (KDD Cup 99 e Forest Cover Type), comparando-o com ECSMiner, OLINDDA-WCE e SCANR. Os resultados demonstraram que o CLAM obteve as menores taxas de erro geral (ERR) e de falsos positivos de novas classes (F_{new}) em todos os *datasets*, com desempenho particularmente superior em cenários com classes recorrentes. Entretanto, cabe observar a limitação de que o método requer supervisão na fase *online* para atualização do modelo.

De Faria (2015) [61], por sua vez, compara OLINDDA, MINAS, ECSMiner e CLAM também no KDD Cup 99, discutindo o compromisso entre taxa de erro em novas classes (UnkR) e erro de classificação global (CER). Os experimentos demonstraram que o MINAS, apesar de operar sem *feedback* externo durante a fase online, obteve valores de CER comparáveis aos métodos supervisionados (CLAM e ECSMiner) no *dataset* KDD.

Nos experimentos apresentados por Mu *et al.* (2017) [14], o SENCForest é comparado com ECSMiner e CLAM e obtém ganhos de acurácia e de capacidade de DN em diferentes *datasets*. O resultado obtido reforça o potencial de abordagens baseadas em *ensembles* de árvores para DN em FCD.

O Mukers [76] emprega uma abordagem baseada em múltiplos *kernels* combinados (estático e dinâmico) para classificação em fluxos não estacionários com surgimento de novas classes. A técnica mantém fronteiras de decisão no espaço de características mapeado, atualizando os *kernels* dinâmicos conforme novos dados chegam. Os experimentos no KDD99 mostraram desempenho superior ao ECSMiner e CLAM em termos de M_{new}, F_{new} e erro total. Diferentemente da maioria dos trabalhos na área, o estudo também avaliou a taxa de processamento de amostras, evidenciando preocupação com a eficiência computacional em cenários de FCD. Contudo, o método opera em regime totalmente supervisionado na fase *online*, assumindo que todos os rótulos verdadeiros estarão disponíveis após a classificação de cada *chunk*, uma premissa que pode ser irrealista em aplicações práticas de detecção de intrusão.

Entre os trabalhos que exploram aspectos arquiteturais, destaca-se a proposta IDSA-IoT [28], que combina recursos de borda e nuvem para detecção de intrusão em redes IoT. Embora não proponha um novo algoritmo de DN, o trabalho avalia experimentalmente MINAS, ECSMiner e AnyNovel no *dataset* Kyoto 2006+, evidenciando *trade-offs* entre acurácia e eficiência computacional em dispositivos com recursos restritos. De forma similar ao Mukers, o estudo avaliou aspectos de eficiência computacional frequentemente negligenciados na literatura, incluindo consumo de memória e taxa de processamento de amostras. Os resultados demonstraram a viabilidade de técnicas de DN em ambientes de alto dinamismo, porém a avaliação restringiu-se à classificação binária (normal versus ataque), deixando em aberto a análise de desempenho em cenários verdadeiramente multiclasse.

O algoritmo FROST [77], baseado em agrupamento *fuzzy*, propõe uma abordagem de DN que utiliza micro-grupos com representação SPFMiC (*Summarized Prototype - Fuzzy Micro-Cluster*) para classificação multiclasse de ataques conhecidos e detecção de padrões anômalos. Os experimentos realizados no *dataset* UNSW-NB15 demonstraram taxas de erro inferiores às obtidas pelo MINAS nas métricas M_{new} e F_{new}. Entretanto, o modelo não incorpora novos padrões de ataque detectados durante a fase *online*, limitando sua

capacidade de adaptação em cenários onde novas ameaças emergem continuamente.

Gao *et al.* (2020) propuseram o SACCOS, um arcabouço semi-supervisionado que integra a detecção de classes emergentes com a adaptação a mudanças de conceito. O processo de DN utiliza um conjunto de *clusters* para isolar *outliers*, os quais são filtrados conforme o nível de confiança de um classificador (como uma rede neural). Em avaliações experimentais com o *dataset* KDD Cup 99, o SACCOS demonstrou acurácia superior e menor taxa de erro em novas classes em comparação ao ECSMiner, utilizando significativamente menos dados rotulados. Entretanto, o estudo aponta que o uso de redes neurais pode tornar o sistema sensível a mudanças abruptas no fluxo devido ao excesso de confiança do modelo, além de sua eficácia prática estar condicionada à disponibilidade de rótulos externos para a fase de propagação.

Quanto ao algoritmo LC-INC (*Learning to Classify with Incremental New Class*), os experimentos conduzidos por Zhou *et al.* (2022) [56] envolveram um fluxo sintético bidimensional e diversas bases de dados amplamente utilizadas, incluindo KDD Cup 99, MNIST, Fashion-MNIST, HAR e o conjunto real de imagens YSneaker. A avaliação considerou acurácia geral, F-Measure e AUC, sendo esta última utilizada para medir a capacidade do modelo em distinguir amostras conhecidas das desconhecidas. Os autores informaram que o LC-INC alcançou desempenho consistentemente superior ao de diversos algoritmos comparativos, como iForest+KNN, ECSMiner e SENNE, em todos os fluxos testados. No conjunto MNIST, por exemplo, o método apresentou ganhos significativos tanto em acurácia quanto em F-Measure. Os autores também destacam a robustez do modelo frente à emergência simultânea de múltiplas novas classes e sua estabilidade diante de variações no tamanho do buffer, sendo um indicativo de sua eficácia mesmo sob restrições de amostragem.

O Trident [19] foi submetido a uma avaliação abrangente utilizando cinco *datasets* de tráfego de rede: USTC-TFC2016, ISCXTor2016, CIC-IDS2017, CIC-IDS2018 e CrossNet2021. O desempenho na descoberta de classes emergentes foi mensurado por meio da *clustering accuracy* (ACC) e da *adjusted mutual information* (AMI), enquanto métricas como precisão, revocação, F1-score e AUC foram aplicadas nas tarefas de classificação supervisionada e detecção binária de anomalias. Os resultados apresentados pelos autores indicaram que o Trident superou significativamente 16 métodos do estado da arte, apresentando F1-score consistentemente superior mesmo em cenários com classes desbalanceadas. Na DN multiclasse, os resultados informados mostram vantagem à medida que o número de classes desconhecidas aumenta, chegando a superar competidores em até 10% no F1-score. Além disso, o algoritmo demonstrou robustez frente a mudanças de conceito causadas por vieses temporais e de cenário, mantendo desempenho estável em diferentes distribuições de dados ao longo do tempo.

Um aspecto metodológico que vale observar nas técnicas de DN avaliadas é a capacidade de descoberta simultânea, que define se um modelo consegue identificar e separar múltiplas classes desconhecidas que emergem na mesma janela de atualização. Algoritmos como ECSMiner, CLAM, SENCForest, KNNENS e LC-INC tratam as novidades de forma agregada durante a fase de descoberta, agrupando todos os padrões não reconhecidos de um mesmo conjunto de dados sob um único rótulo genérico, o que restringe o sistema a descobrir apenas uma "nova classe"por vez. Em contraste, abordagens equipadas com mecanismos que superam essa limitação. Modelos como MINAS, Trident e SACCOS possuem a capacidade de múltiplas descobertas simultâneas, conseguindo isolar, distinguir e diferentes padrões de dados que ocorram concomitantemente antes de integrá-las ao modelo definitivo.

Em síntese, embora os trabalhos avaliados evidenciem que técnicas de DN como MINAS, ECSMiner, SENCForest são promissoras para detecção de intrusão em fluxos, a literatura existente tende a: (i) concentrar-se em um único *dataset*, frequentemente o KDD Cup 99; (ii) privilegiar métricas preditivas em detrimento de medidas de eficiência computacional, como consumo de memória e capacidade de acompanhar o *throughput* do fluxo; e (iii) não analisar de forma detalhada a evolução do desempenho ao longo do FCD. No contexto específico de IDS baseados em fluxo, Miani *et al.* (2025) [36] reforçam a necessidade de estudos que combinem *datasets* contemporâneos, métricas preditivas especializadas e medidas de eficiência computacional sob restrições de tempo real. Esta dissertação buscou endereçar essas lacunas ao comparar diferentes algoritmos de DN em múltiplos *datasets* mais atuais de IDS, avaliando simultaneamente desempenho preditivo e aspectos de eficiência computacional.

3.2 Considerações Finais

Este capítulo situou a presente pesquisa no contexto da literatura existente. Foram revisados trabalhos que aplicam AM em IDS em abordagens supervisionadas, incluindo estudos sobre aplicação de DN multiclasse neste domínio. O capítulo abordou algoritmos como ECSMiner, MINAS, SENCForest, KNNENS e Trident, avaliados em diferentes *datasets* e com distintos conjuntos de métricas. A análise revelou lacunas recorrentes na literatura: concentração em *datasets* desatualizados, avaliação restrita à capacidade preditiva com negligência da eficiência computacional ou ausência de métricas específicas de DN. O próximo capítulo descreve a metodologia adotada nesta pesquisa para endereçar essas lacunas, detalhando o protocolo experimental, os *datasets* utilizados, as técnicas avaliadas e as métricas de desempenho empregadas.

Capítulo 4

Avaliação de Técnicas de Detecção de Novidade em IDS

Este capítulo aborda o método adotado para a avaliação de algoritmos de DN aplicados a IDS. O objetivo foi avaliar o desempenho de um conjunto de algoritmos de DN na identificação de novas classes de ataques cibernéticos em redes de computadores.

O capítulo está dividido da seguinte maneira: o método adotado para avaliar algoritmos de DN em FCD está descrito na Seção 4.1. Os *datasets* utilizados são detalhados na Seção 4.2, as técnicas avaliadas na Seção 4.3 e os critérios e métricas de desempenho na Seção 4.5.

4.1 Método Proposto

Esta seção descreve o método adotado para avaliar algoritmos de DN em FCD, com foco na identificação de novas classes de ataques em tráfego de rede. O protocolo compreende quatro etapas: (i) pré-processamento dos *datasets*; (ii) particionamento sequencial dos *datasets*; (iii) ajuste de hiperparâmetros via *Grid Search* e (iv) avaliação dos modelos, com registro contínuo de métricas de desempenho.

4.1.1 Pré-processamento dos *datasets*

A primeira atividade realizada após a seleção dos *datasets* foi uma etapa de pré-processamento, com o objetivo de incrementar a qualidade e a consistência dos dados utilizados nos experimentos. Essa etapa é essencial para preparar os dados brutos para a análise, reduzir ruídos e eliminar informações irrelevantes que possam comprometer o desempenho performance dos algoritmos. As ações de pré-processamento realizadas foram as seguintes:

- **Limpeza de dados**: remoção de valores inválidos (ausentes, negativos, infinitos), remoção de amostras repetidas.
- **Eliminação de atributos**: remoção de atributos que não agregam informação útil para o aprendizado, tais como endereços IP, identificadores das conexões ou índices das amostras. A avaliação e remoção dos atributos foram realizadas manualmente, analisando cada um dos atributos dos *datasets*.
- **Codificação de atributos**: mapeamento dos atributos categóricos, como classes e protocolos, em valores inteiros;

4.1.2 Particionamento sequencial dos *datasets*

Cada *dataset* foi particionado de forma sequencial, preservando rigorosamente a ordem temporal das amostras, conforme recomendado em cenários de aprendizado em fluxo [78]. Diferentemente do aprendizado em lote, neste tipo de problema o embaralhamento das amostras pode eliminar dependências temporais relevantes, como a ordem de ocorrência de mudanças no tráfego e o surgimento de novas classes.

As partições foram definidas nas seguintes proporções:

- **Partição *offline* inicial (30%):** utilizada para o treinamento inicial do modelo e para estimar parâmetros de normalização.
- **Partição de validação (20%):** reservada exclusivamente para ajuste de hiperparâmetros por *Grid Search*.
- **Partição *online* (50%):** empregada para simular o FCD e realizar a avaliação final, sem uso em qualquer etapa de ajuste.

4.1.3 Ajuste de hiperparâmetros

O ajuste de hiperparâmetros foi conduzido via *Grid Search*, utilizando exclusivamente as partições *offline* inicial e de validação. Para cada combinação testada, o modelo foi treinado na partição *offline* inicial (30%) e avaliado na partição de validação (20%), respeitando a ordem temporal das amostras, buscando maximizar a métrica H_{new}. Desta forma, o objetivo desta etapa foi identificar um conjunto de hiperparâmetros que otimizasse a capacidade de DN do modelo.

Após a seleção do melhor conjunto de hiperparâmetros para cada técnica, o modelo foi treinado utilizando a união das partições *offline* inicial e de validação, totalizando 50% dos dados. Esse conjunto constitui a partição *offline* final, cujas classes presentes foram consideradas como classes conhecidas pelo modelo. Classes que surgiram apenas na

partição *online* (50% restante) foram tratadas como classes emergentes. Essa estratégia evita a seleção manual de quais classes devem ser conhecidas *a priori*, refletindo cenários realistas onde o surgimento de novas ameaças é imprevisível.

O treinamento com todos os dados disponíveis é uma prática recomendada por Goodfellow, Bengio e Courville (2016) [79] e Kuhn e Johnson (2013) [80], maximizando o aproveitamento da informação histórica. Os espaços de busca explorados, as justificativas de seleção e as configurações finais adotadas para cada algoritmo são detalhados na Seção 4.3.

Um aspecto crítico em experimentos com FCD refere-se ao tratamento de operações que dependem de estatísticas globais dos dados, como a normalização dos valores numéricos. Conforme observado por Gomes *et al.* (2019) [44], estatísticas como valores mínimos e máximos de atributos são, em princípio, desconhecidas *a priori* em fluxos contínuos reais, uma vez que novos dados podem apresentar valores fora dos intervalos observados historicamente.

Nesta pesquisa, as operações de ajuste (*fit*) e transformação (*transform*) foram realizadas respeitando o isolamento entre as etapas experimentais:

1. **Durante o *Grid Search*:** Os parâmetros de normalização foram estimados (*fit*) exclusivamente a partir da partição *offline* inicial (30%) e aplicados (*transform*) tanto à partição *offline* inicial quanto à partição de validação (20%).
2. **Após seleção dos hiperparâmetros:** Novos parâmetros de normalização foram estimados (*fit*) utilizando a partição *offline* final (união de *offline* inicial e validação, totalizando 50%).
3. **Na avaliação final:** Os parâmetros estimados na partição *offline* final foram aplicados (*transform*) à partição *online* (50%), sem qualquer reajuste baseado em estatísticas dessa partição.

Essa estratégia garante que: (i) durante o *Grid Search*, não há vazamento de informação da partição de validação para o treinamento; e (ii) durante a avaliação final, não há vazamento de informação da partição *online* para o modelo. As transformações utilizam apenas conhecimento disponível no momento do treinamento inicial de cada etapa. Conforme destacado por Gama (2014) [78], tal precaução metodológica é essencial para validar algoritmos em contextos de aprendizado contínuo, evitando que o modelo tenha acesso antecipado a características de dados futuros.

4.1.4 Avaliação dos modelos

A avaliação final foi realizada na partição *online* por meio de simulação de um FCD, processando as amostras sequencialmente e na ordem temporal original. Para cada amostra, o algoritmo produziu uma predição e, nas condições previstas por cada técnica, realizou atualização incremental do modelo. Os rótulos verdadeiros foram utilizados exclusivamente para o cálculo das métricas de desempenho.

A cada 1.000 amostras processadas, foram registrados o consumo de memória RAM, o tempo de processamento acumulado e as métricas de capacidade preditiva. Ao término do fluxo, foram computadas as métricas globais de desempenho, eficiência computacional e tempo de detecção (TTD), conforme descrito na Seção 4.5. A Figura 4.1 apresenta o método aqui descrito.

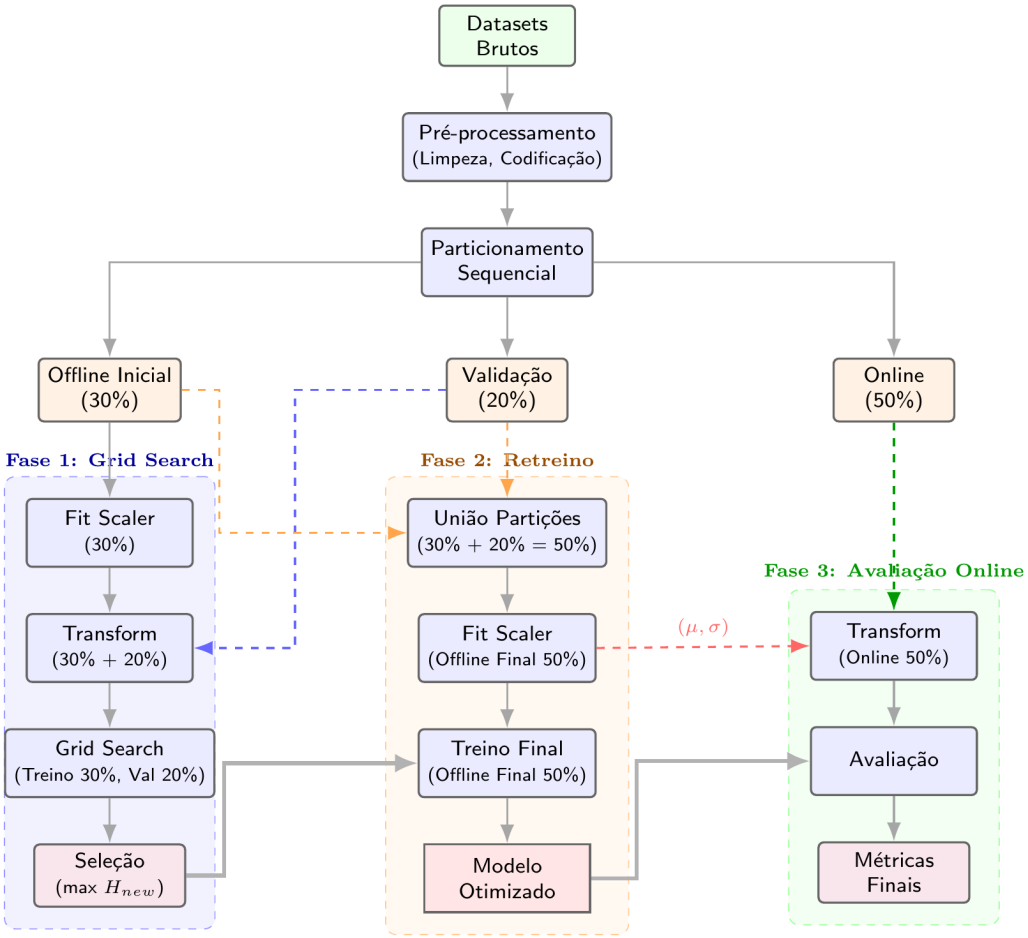

Figura 4.1: Metodo proposto.

4.2 *Datasets* Utilizados

Diversos critérios podem ser usados para escolher *datasets* adequados a pesquisas em IDS baseados em AM. De acordo com Ring *et al.* (2019) [81], é essencial considerar a atualidade dos ataques, a representatividade do tráfego real e a presença de dados rotulados de forma consistente. Já em [82] e [83], os autores enfatizam a importância da qualidade dos dados e diversidade das amostras, fundamentais para evitar viés de treinamento. Estudos recentes, como [84] e [85], propõem *frameworks* que avaliam o grau de cobertura de ataques modernos e a aderência às características operacionais de redes reais. Os trabalhos enfatizam a necessidade de *datasets* com grande variedade de ataques e continuidade temporal, pois esses fatores permitem simular fluxos nos quais novas ameaças surgem gradualmente, condição essencial para avaliar métodos de DN.

Há *datasets* de detecção de intrusão que, embora ainda sejam utilizados em trabalhos nesta área, podem não ser os mais adequados nos dias de hoje por estarem muito desatualizados. Dessa forma, um dos critérios de escolha de *datasets* para esta pesquisa foi a presença de ataques que ainda sejam relevantes nos dias atuais.

Com base nos critérios desejáveis apresentados, os *datasets* a seguir foram selecionados para os experimentos. Essas bases, em geral, foram obtidas por meio de emulação de ataques cibernéticos em ambiente de laboratório. Todas as bases são de classificação.

- **UNSW-NB15**
  O UNSW-NB15 [86] foi desenvolvido com o objetivo de superar limitações de *datasets* anteriores, como KDD99 e NSL-KDD, que já não refletiam adequadamente o ambiente de ameaças contemporâneo. Os dados foram coletados no *Cyber Range Lab* do *Australian Centre for Cyber Security* (ACCS) da *University of New South Wales*, utilizando a ferramenta IXIA PerfectStorm para gerar tráfego sintético de ataques mais modernos. O *dataset* é caracterizado por ser um híbrido de comportamentos normais reais e atividades de ataque sintéticas contemporâneas, com 49 atributos extraídos com ajuda de ferramentas como Argus e Bro-IDS. Os dados foram organizados em 9 categorias de ataques (*Fuzzers*, *Analysis*, *Backdoors*, DoS, *Exploits*, *Generic*, *Reconnaissance*, *Shellcode* e *Worms*), além do tráfego normal.

- **Ton-IoT-V2**
  O Ton IoT V2 [87] foi desenvolvido com o objetivo de representar tráfego de redes modernas que incluem dispositivos normalmente referenciados como IoT, tais como smartphones e smart TVs. Os dados foram coletados de uma rede realista e de grande escala projetada no *Cyber Range IoT Labs* da *University of New South Wales*. De acordo com os desenvolvedores, o *dataset* destaca-se por conter um grande número de protocolos e serviços e possuir eventos de segurança e cenários de *malware*

modernos. Os dados foram organizados em 9 classes de ataques, além do tráfego normal.

- **DAPT 2020**
  De acordo com Myneni *et al.* (2020) [88], o *dataset* DAPT 2020 foi criado para abordar a lacuna existente na pesquisa sobre *Advanced Persistent Threat* (APT), que carece de conjuntos de dados adequados para simular e estudar esses tipos de ataques. O DAPT 2020 foi desenvolvido a partir de um experimento que simulou comportamentos de APTs em uma rede em nuvem ao longo de cinco dias. Durante esse período, os pesquisadores coletaram dados que representavam diferentes fases de um ataque APT, incluindo a fase inicial, onde não havia ataques em execução, proporcionando assim uma linha de base para o tráfego benigno na rede. O *dataset* possui, além da classe benigna, 12 classes de ataques, distribuídas em 4 fases: reconhecimento, estabelecimento do acesso, movimento lateral e exfiltração de dados.

Os *datasets* escolhidos oferecem uma ampla diversidade de ataques de rede modernos, distribuídos em diferentes dias e cenários, o que favorece a simulação de fluxos temporais com surgimentos de novas classes. A Tabela 4.1 mostra as categorias de ataques presentes em cada *dataset*, assim como o número de amostras de cada classe.

Tabela 4.1: Categorias de ataques presentes nos *datasets* utilizados

Dataset	**Categoria de ataque**	**Nº de amostras**
UNSWNB15	Benigno	85722
	Analysis	2032
	Backdoor	1880
	DoS	5500
	Exploits	27434
	Fuzzers	20960
	Generic	7599
	Reconnaissance	9991
	Shellcode	1523
	Worms	171
ToN-IoT-V2	Benigno	41958
	Backdoor	18692
	DDoS	19972
	DoS	14669
	Injection	19962
	MITM	1039
	Passwords	19829
	Ransomware	14266
	Scanning	18615
	XSS	14514
DAPT 2020	Benigno	63611
	Account bruteforce	141
	Account discovery	2408
	Backdoor	20
	Command injection	12
	Csrf	7
	Data exfiltration	6
	Directory bruteforce	9970
	Malware download	2
	Network scan	7742
	Privilege escalation	13
	Sql injection	84
	Web vulnerability scan	2574

A Tabela 4.2 destaca os atributos eliminados em cada base na etapa de pré-processamento. A Tabela 4.3, por sua vez, sumariza alguns aspectos quantitativos dos *datasets* utilizados nos experimentos. Os dados apresentados referem-se aos números finais dos *datasets*, após as atividades de pré-processamento. Os quantitativos de classes conhecidas e novas classes foram identificados após a divisão dos *datasets* da forma apresentada.

Tabela 4.2: Atributos removidos

Dataset	**Atributos removidos**
UNSWNB15	Source IP Address, Destination IP Address
Ton-IoT-V2	Source IP Address, Destination IP Address
DAPT 2020	Flow ID, Source IP Address, Destination IP Address, Timestamp

Tabela 4.3: Informações Quantitativas dos *Datasets*

	UNSWNB15	**Ton-IoT-V2**	**DAPT 2020**
Amostras FCD	81.407	91.759	43.295
Classes Conhecidas	9	6	6
Novas Classes	1	4	7

4.3 Técnicas de DN Avaliadas

Nesta pesquisa, foram selecionados e avaliados cinco algoritmos de DN: ECSMiner [13], MINAS [10], SENCForest [14], KNNENS [12] e Trident [19]. Essas técnicas foram escolhidas por representarem diferentes abordagens de aprendizado com DN em FCD. ECSMiner, MINAS e SENCForest são três das mais relevantes técnicas de DN, sendo frequentemente citadas em trabalhos nesta área de pesquisa, enquanto KNNENS e Trident representam abordagens mais recentes.

Originalmente, o ECSMiner pressupõe que os rótulos de todas as amostras estarão disponíveis após determinado tempo. Conforme proposto por Paiva (2014) [10], para uma melhor comparação com os demais algoritmos, foi empregada neste trabalho uma versão do ECSMiner que não utiliza este *feedback*, ou seja, os rótulos na fase *online* não são utilizados pelo algoritmo.

As tabelas Tabela 4.5, Tabela 4.4, Tabela 4.6, Tabela 4.7 e Tabela 4.8 apresentam os hiperparâmetros utilizados nos experimentos. Parte dos hiperparâmetros foi escolhida a partir de atividades de *tuning*, enquanto outros foram escolhidos com base em trabalhos anteriores.

No ECSMiner, o parâmetro K define o número de *clusters* do algoritmo de agrupamento, modulando a granularidade da representação. O *min_examples_cluster* estabelece o limiar para declaração de novas classes, equilibrando a detecção precoce contra a

inclusão de ruídos ou muitas novas classes simultaneamente. O *ensemble_size* determina o número de classificadores no *ensemble*, afetando a robustez das predições.

Tabela 4.4: Hiperparâmetros ECSMiner

Parâmetro	**Descrição**	**Valor**
K	Parâmetro K do algoritmo de agrupamento	30, 60, 100, 140, 180
min_examples_cluster	Mínimo de amostras para declarar uma nova classe	10, 20, 50, 100, 200
ensemble_size	Número de classificadores a serem usados para criar o conjunto	5
init_algorithm	Algoritmo de agrupamento a ser usado para inicializar os grupos	Kmeans

No MINAS, o *window_size* controla o mecanismo de esquecimento, determinando o número de amostras após o qual micro-grupos inativos são movidos para a memória *sleep*. Este parâmetro é importante para a adaptação do modelo a extensões de conceitos conhecidos e para permitir que representações desatualizadas sejam gradualmente descartadas.

O *min_short_mem_trigger* estabelece o número mínimo de amostras na memória de curto prazo para acionar o processo de DN. Valores baixos aumentam a frequência de verificação, potencialmente detectando novidades mais rapidamente, mas com maior risco de falsos positivos. Valores elevados exigem maior acúmulo de evidências antes da detecção. O *min_examples_cluster* define o número mínimo de amostras necessárias para formar um novo micro-grupo válido, sendo essencial para distinguir entre ruído e padrões emergentes genuínos.

Tabela 4.5: Hiperparâmetros MINAS

Parâmetro	**Descrição**	**Valor**
kini	Número de K grupos para o algoritmo de agrupamento	15
cluster_algorithm	Algoritmo de agrupamento (Clustream ou KMeans)	CluStream
window_size	Número de amostras usadas pelo mecanismo de esquecimento	100, 200, 300, 500, 1000
min_short_mem_trigger	Mínimo de amostras na memória de curto prazo para acionar a detecção de novidades	100, 200, 300, 400, 500
min_examples_cluster	Mínimo de amostras para formar um micro-grupo	20, 40, 80, 100, 300

No SENCForest, o *num_tree* define o número de árvores na floresta, impactando a precisão das pontuações de anomalia. O *num_sub* controla o tamanho das subamostras utilizadas na construção de cada árvore. O $buffer\_size$ determina a capacidade de armazenamento de amostras que serão utilizadas na atualização do modelo, influenciando seu comportamento futuro.

Tabela 4.6: Hiperparâmetros SENCForest

Parâmetro	**Descrição**	**Valor**
num_tree	Número de árvores	32
num_sub	Tamanho de subamostragem	10, 20, 80, 100, 200
buffer_size	Tamanho do armazenamento de novas classes	200, 300, 600, 1000, 3000

No KNNENS, *num_bags* representa o número de *sub-ensembles* criados, *num_boot* é o número de amostras por classe em cada *sub-ensemble* e K é o número de vizinhos usados no KNN. O *threshold* regula a distância normalizada que determina se uma amostra é classificada como novidade, sendo importante para o equilíbrio entre F_{new} e M_{new}. O $buffer\_size$ define a capacidade de armazenamento de amostras que serão utilizadas na atualização do modelo, influenciando seu comportamento futuro.

Tabela 4.7: Hiperparâmetros KNNENS

Parâmetro	**Descrição**	**Valor**
num_boot	Amostras por classe no *bootstrap*	10
num_bags	Número de amostragens *bootstrap*	10
K	Número de vizinhos usados no KNN	4
threshold	Limiar de detecção de novidade	0.5, 0.6, 0.7, 0.8, 0.9
buffer_size	Tamanho do armazenamento de novas classes	30, 60, 100, 200, 300

No Trident, o $buffer\_size$ controla a quantidade de amostras acumuladas no *buffer* antes de acionar o processo de clusterização para identificação de novas classes. Valores maiores permitem maior acúmulo de evidências, impactando a qualidade da clusterização.

O *min_samples* define o número mínimo de amostras necessárias para que um *cluster* seja considerado válido durante a identificação de novas classes, filtrando agrupamentos espúrios causados por ruído.

Os parâmetros *threshold_q* e *threshold_level* são utilizados na determinação automática dos limiares de aceitação de cada aprendiz *one-class*. O *threshold_q* representa o parâmetro de risco utilizado na Teoria do Valor Extremo (EVT), enquanto *threshold_level* define o valor inicial para estimativa do limiar. Esses parâmetros controlam a sensibilidade de cada aprendiz na distinção entre amostras da própria classe e amostras de outras classes ou novidades.

Tabela 4.8: Hiperparâmetros Trident

Parâmetro	Descrição	Valor
threshold_q	Parâmetro de risco (EVT)	0.1
min_samples	Mínimo de amostras por *cluster* válido	30, 40, 50, 100, 300
threshold_level	Valor inicial para limiar	0.9, 0.95, 0.98, 1, 1.1
buffer_size	Capacidade do *buffer* de desconhecidos	60, 80, 100, 200, 300

A Tabela 4.9 apresenta os valores de hiperparâmetros selecionados pelo *Grid Search* para cada combinação de algoritmo e *dataset*, com base na maximização da métrica H_{new} na partição de validação.

Tabela 4.9: Hiperparâmetros selecionados após *Grid Search* para cada *dataset*

Algoritmo	Parâmetro	UNSW-NB15	ToN-IoT-V2	DAPT 2020
ECSMiner	K	140	30	180
	min_examples_cluster	10	200	10
MINAS	*window_size*	200	1000	300
	min_short_mem_trigger	100	500	100
	min_examples_cluster	20	300	40
SENCForest	*num_sub*	10	100	20
	buffer_size	3000	3000	300
KNNENS	*threshold*	0.5	0.8	0.9
	buffer_size	200	100	100
Trident	*min_samples*	40	40	30
	threshold_level	0.95	0.90	0.90
	buffer_size	100	100	100

4.4 Ambiente Experimental

Os experimentos foram realizados no ambiente Google Colab. O ambiente tinha à disposição GPU NVIDIA L4, CPU Intel(R) Xeon(R) @ 2.20GHz, 53 GB de RAM, sistema operacional Ubuntu 22.04 e Python 3.12.12.

As implementações do SENCForest e do KNNENS foram obtidas das versões originais disponibilizadas pelos pesquisadores, desenvolvidas em linguagem MATLAB. Para viabilizar a execução no ambiente utilizado, foi desenvolvido um código em Python que integra a ferramenta GNU Octave 8.4, permitindo a execução dos algoritmos originais sem modificações que pudessem comprometer a fidelidade aos métodos propostos.

Para os experimentos com MINAS e ECSMiner, foi utilizada a biblioteca StreamNDR [89] versão 0.1.6, que disponibiliza implementações desses algoritmos compatíveis com o

framework River para aprendizado em fluxo. A implementação do Trident foi obtida do repositório oficial disponibilizado pelos autores.

4.5 Critérios e Métricas de Desempenho

Nesta pesquisa, os algoritmos selecionados foram avaliados segundo três dimensões complementares: capacidade preditiva, eficiência computacional e tempo de detecção de novidades. Essas dimensões contemplam os aspectos considerados relevantes para DN em FCD no contexto de detecção de intrusão, conforme discutido na Seção 2.4.

4.5.1 Capacidade Preditiva

A avaliação da capacidade preditiva considerou tanto o desempenho geral de classificação quanto a habilidade específica de identificar novas classes. Para tanto, foram utilizadas métricas adequadas para cada aspecto.

A **acurácia** mede a proporção de predições corretas em relação ao total de amostras avaliadas. Para o cálculo dessa métrica, considerou-se neste trabalho que uma amostra de classe nova foi classificada corretamente se o algoritmo a identificou como pertencente a alguma novidade, independentemente do rótulo específico atribuído. Esta abordagem é necessária porque os algoritmos de DN atribuem rótulos internos às novas classes detectadas, que não necessariamente correspondem aos rótulos originais do *dataset*.

Para avaliar especificamente a capacidade de DN, foram utilizadas as métricas M_{new} e F_{new}, já apresentadas na Seção 2.4:

- M_{new}: proporção de amostras de classes novas incorretamente classificadas como pertencentes a classes conhecidas. Valores elevados indicam incapacidade de detectar novidades.
- F_{new}: proporção de amostras de classes conhecidas incorretamente classificadas como novidades. Valores elevados indicam excesso de falsos alarmes.

Adicionalmente, este trabalho propôs a métrica H_{new} para fornecer uma avaliação balanceada da capacidade de DN. A métrica combina M_{new} e F_{new} por meio de uma média harmônica, conforme a Equação 4.1:

$$H_{new} = \frac{2(1 - M_{new})(1 - F_{new})}{(1 - M_{new}) + (1 - F_{new})} \tag{4.1}$$

A escolha da média harmônica foi motivada por sua propriedade de penalizar desproporcionalidades de forma mais agressiva que a média aritmética. Um algoritmo com

$M_{new} = 0$ e $F_{new} = 0{,}8$, por exemplo, obteria $H_{new} = 0{,}18$, refletindo que um desempenho ruim em uma das métricas compromete significativamente o resultado global. O valor $H_{new} = 1$ indica desempenho perfeito em ambas as métricas, enquanto $H_{new} = 0$ indica falha completa em pelo menos uma delas.

4.5.2 Tempo de Detecção de Novidade (TTD)

Em [84] os autores citam algumas propriedades de interesse para um IDS. Entre elas, o trabalho apresenta o conceito de **acionabilidade**, definida como uma medição da utilidade dos alertas para um operador de segurança. Um dos aspectos mais relevantes para a acionabilidade é o fator tempo, conforme abordado em [90], [91] e [92]. Uma detecção rápida pode assegurar que a informação sobre um ataque chegue em tempo hábil para que as devidas ações de investigação e contenção possam ser tomadas. Uma detecção tardia, por outro lado, pode tornar o alerta irrelevante, pois o ataque já terá causado danos ou se dissipado.

Para incorporar esse aspecto à avaliação das técnicas de DN, este trabalho adotou uma métrica aqui chamada de *Time-To-Detection* (TTD). Na literatura há referências a conceitos com este objetivo, como em [26], onde é sugerido que o TTD seja medido em termos de número de amostras necessárias para a detecção de uma novidade. Trabalhos como [78] e [37] também discutem a importância deste tipo de métrica para avaliação de algoritmos em FCD, destacando que o tempo de adaptação a mudanças é um fator crítico para aplicações em tempo real.

Nesta pesquisa, o TTD foi definido como o intervalo de tempo entre o surgimento da primeira amostra de uma classe nova no fluxo e a primeira classificação assertiva dessa amostra como sendo uma novidade. Seja t_1 o instante de tempo em que a primeira amostra de uma nova classe c surge no FCD e t_3 o instante em que o algoritmo detecta corretamente uma amostra dessa classe como novidade. O TTD é calculado conforme a Equação 4.2:

$$TTD = t_3 - t_1 \tag{4.2}$$

No caso geral, uma detecção somente é considerada válida para fins de cálculo do TTD se três condições forem simultaneamente satisfeitas:

1. **Amostra genuinamente nova**: a amostra detectada deve pertencer efetivamente a uma classe não presente no treinamento inicial.
2. **Classificação como novidade**: o algoritmo deve ter classificado a amostra como pertencente a uma classe nova.

3. **Pós-atualização do modelo**: deve ter ocorrido ao menos uma atualização de modelo entre o surgimento da amostra de nova classe e sua detecção assertiva.

A terceira condição é importante para realizar uma estimativa mais adequada para técnicas que dependem de uma atualização de modelo para serem capazes de identificarem uma nova classe. Uma amostra de classe nova classificada como novidade *antes* de qualquer atualização do modelo representaria, nesse tipo de abordagem, uma detecção prematura, possivelmente decorrente de configuração excessivamente permissiva dos limiares de decisão, não de aprendizado efetivo.

Adicionalmente, contabilizar o tempo de detecção em termos do total de amostras avaliadas oferece uma perspectiva independente da infraestrutura computacional utilizada. Enquanto o TTD em segundos é uma medida sensível ao hardware, o TTD medido em amostras reflete diretamente a quantidade de evidências que o fluxo precisou apresentar antes que o algoritmo fosse capaz de detectar a novidade — uma medida intrinsecamente ligada ao mecanismo de cada técnica. Essa distinção é relevante pois, conforme apontado por Gaudreault e Branco (2025) [93], a dimensão temporal da DN é um aspecto crítico de avaliação frequentemente negligenciado, em especial em contextos como detecção de intrusão, onde a rapidez na identificação de novas ameaças é determinante para a acionabilidade do alerta.

Para o SENCForest, KNNENS e Trident a condição de pós-atualização de modelo não se aplica. Nestas técnicas, o *buffer* serve para acumular amostras detectadas como novidade para posterior atualização do modelo, mas a detecção em si já ocorreu. Assim, o TTD foi implementado como o intervalo entre o surgimento da primeira amostra de classe nova e sua primeira classificação como novidade, sem a exigência de atualização entre os dois momentos.

A Figura 4.2 ilustra os diferentes cenários de medição do TTD para algoritmos que requerem atualização do modelo (MINAS e ECSMiner). Para o SENCForest, KNNENS e Trident, apenas os cenários 1 e 3 são aplicáveis, sendo que no cenário 1 a detecção válida pode ocorrer antes ou após a atualização do modelo.

Para implementar este experimento, foram geradas cópias dos *datasets* mantendo apenas uma nova classe em cada um deles. Assim, com apenas uma nova classe em cada *dataset*, é possível obter uma estimativa um pouco mais precisa do tempo de detecção de cada algoritmo para a classe em questão. Para o UNSWNB15 foi escolhida a classe *Generic*, para o Ton Iot V2 a classe *Password* e para o DAPT2020 foi utilizada a classe *SQL Injection*. As classes escolhidas foram as que possuíam mais amostras no *dataset*.

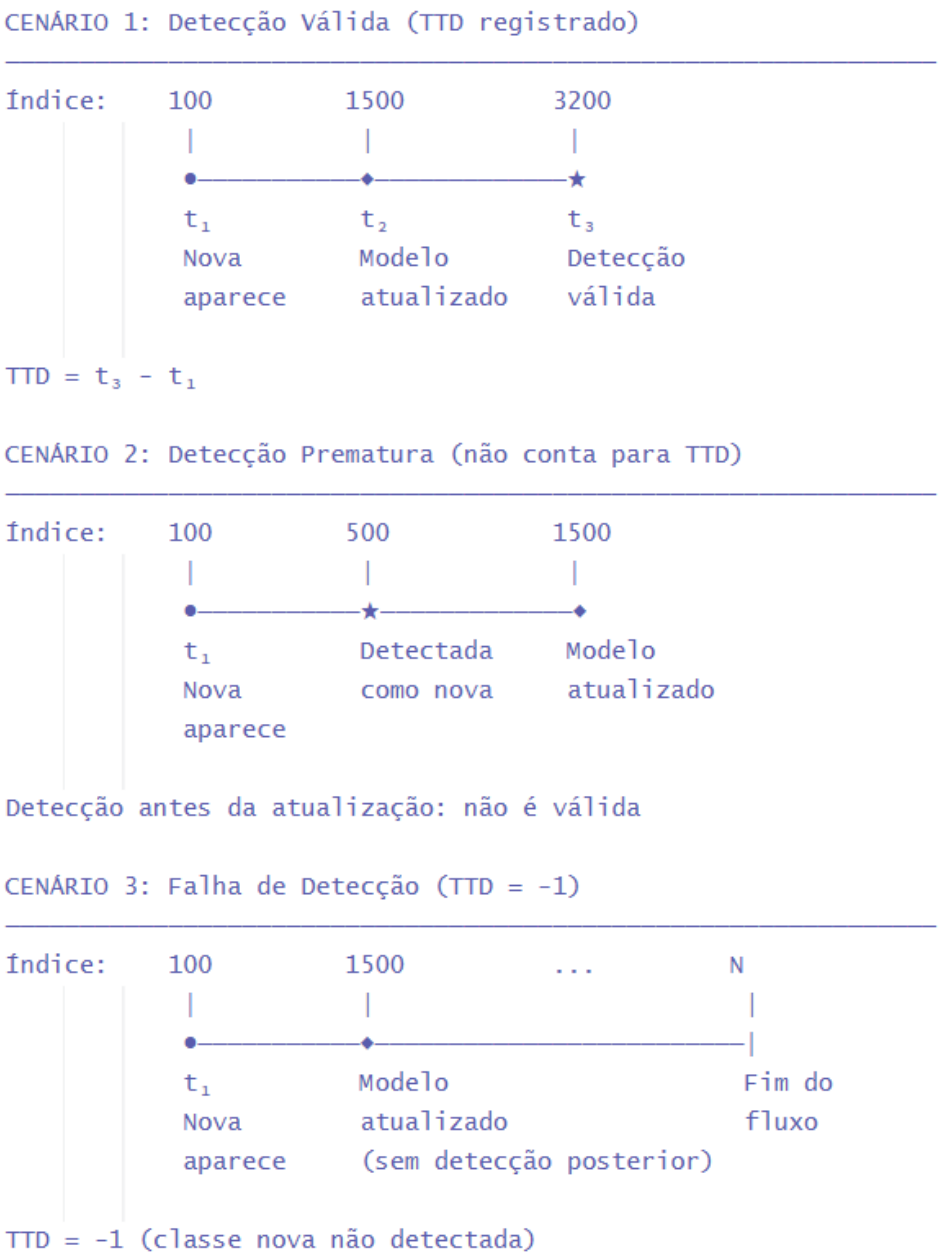

Figura 4.2: Medição do TTD.

4.5.3 Eficiência Computacional

O consumo de memória RAM foi monitorado para verificar como cada algoritmo gerencia os recursos alocados durante a evolução do fluxo. Os valores foram registrados a cada 1000 amostras. Esta métrica é particularmente relevante para FCD, onde o crescimento descontrolado do consumo de memória pode inviabilizar a operação em tempo real. Algoritmos que mantêm estável o consumo de memória ao longo do tempo demonstram uma capacidade adequada de gerenciamento de recursos em cenários de fluxos potencialmente

infinitos.

A taxa de avaliação mede o valor médio da quantidade de amostras processadas por segundo, permitindo verificar se os algoritmos são capazes de acompanhar a taxa de chegada dos dados do fluxo. Assim, para cada experimento, essa taxa foi calculada como a razão entre o quantidade de amostras do fluxo (etapa *online*) e o tempo total de avaliação destas amostas. Em um IDS em operação, essa taxa de processamento deve ser igual ou superior à taxa de geração de tráfego na rede monitorada (vazão ou *throughput*), para evitar acúmulo de dados não processados. Para cada *dataset* utilizado nesta pesquisa foi realizada uma estimativa de *throughput*. A estimativa do tempo total de coleta de tráfego, para fins de cálculo do *throughput*, foi feita com base nos *timestamps* e nas informações disponíveis na documentação dos *datasets*. Para esta medição do tempo, foram excluídos os intervalos de descontinuidade de tráfego presentes nos *datasets*.

4.6 Considerações Finais

Este capítulo descreveu o método adotado para a avaliação dos algoritmos de DN na tarefa de detecção de intrusão. O método compreende quatro etapas: pré-processamento dos dados, particionamento sequencial que preserva a ordem temporal, ajuste de hiperparâmetros via *Grid Search* com maximização do H_{new} e avaliação final na partição *online*. Foram detalhados os três *datasets* utilizados, cada um representando um perfil distinto de tráfego e padrões de surgimento de novas classes. Os cinco algoritmos avaliados foram apresentados juntamente com seus espaços de busca e configurações finais de hiperparâmetros. Por fim, foram definidas as métricas de avaliação, incluindo acurácia, M_{new}, F_{new}, a métrica proposta H_{new}, o TTD e as medidas de eficiência computacional. O próximo capítulo apresenta e discute os resultados obtidos nos experimentos conduzidos segundo este protocolo.

Capítulo 5

Resultados

Este capítulo apresenta os resultados obtidos nos experimentos de avaliação dos algoritmos de DN aplicados à detecção de intrusão. As análises estão organizadas em três dimensões complementares: capacidade preditiva (Seção 5.1), eficiência computacional quanto ao consumo de memória (Seção 5.2) e taxa de processamento de amostras (Seção 5.3). Os experimentos foram conduzidos nos *datasets* DAPT 2020, UNSW-NB15 e ToN-IoT-V2, descritos no Capítulo 4, utilizando os algoritmos ECSMiner, MINAS, SENCForest, KNNENS e Trident, apresentados no Capítulo 2.

5.1 Capacidade Preditiva

A capacidade preditiva dos algoritmos foi avaliada por meio de métricas que contemplam tanto o desempenho geral de classificação quanto a habilidade específica de DN. As Tabelas 5.1, 5.2 e 5.3 apresentam os valores finais obtidos para acurácia, M_{new}, F_{new} e H_{new} para os *datasets* UNSW-NB15, Ton-Iot-V2 e DAPT 2020, respectivamente.

5.1.1 UNSW-NB15

No *dataset* UNSW-NB15, o Trident apresentou a maior acurácia (0,569), demonstrando a melhor capacidade de generalização entre os modelos avaliados. O MINAS registrou uma acurácia de 0,202. Vale ressaltar que, nesse caso, uma parcela significativa das amostras é retida no *buffer* de desconhecidos ao longo da evolução do fluxo. Embora essa estratégia evite classificações precipitadas, ela pode penalizar a acurácia, visto que as amostras não classificadas não são computadas. O SENCForest teve o desempenho mais baixo neste cenário (0,104).

Quanto à capacidade de DN, o MINAS apresentou o melhor desempenho geral, atingindo o maior H_{new} (0,861). O algoritmo demonstrou um equilíbrio mais eficaz, combi-

nando baixa perda na detecção de novas classes ($M_{new} = 0,233$) com uma taxa de falsos positivos extremamente baixa ($F_{new} = 0,019$). O SENCForest, por outro lado, obteve o pior H_{new} (0,242), prejudicado por uma alta taxa de falsos novos ($F_{new} = 0,860$).

Tabela 5.1: Métricas de capacidade preditiva no *dataset* UNSW-NB15.

Algoritmo	**Acurácia**	M_{new}	F_{new}	H_{new}
ECSMiner	0,427	0,480	**0,004**	0,683
MINAS	0,202	0,233	0,019	**0,861**
SENCForest	0,104	**0,123**	0,860	0,242
KNNENS	0,470	0,425	0,026	0,723
Trident	**0,569**	0,288	0,071	0,806

5.1.2 ToN-IoT-V2

No ToN-IoT-V2, o SENCForest obteve a maior acurácia (0,738). O KNNENS registrou a menor acurácia (0,130), indicando dificuldades de adaptação do *ensemble* às características deste tráfego.

Em relação à capacidade de DN, o ECSMiner obteve o melhor resultado de detecção, com um H_{new} de 0,855. O algoritmo diferenciou-se pela ausência total de falsos novos ($F_{new} = 0,000$), ainda que tenha mantido um M_{new} de 0,254. O SENCForest apresentou o pior desempenho neste cenário, com H_{new} de apenas 0,264, influenciado pelo alto valor de F_{new}.

Tabela 5.2: Métricas de capacidade preditiva no *dataset* ToN-IoT-V2.

Algoritmo	**Acurácia**	M_{new}	F_{new}	H_{new}
ECSMiner	0,394	0,254	**0,000**	**0,855**
MINAS	0,443	0,066	0,474	0,673
SENCForest	**0,738**	**0,048**	0,847	0,264
KNNENS	0,130	0,823	0,004	0,300
Trident	0,420	0,550	0,637	0,402

5.1.3 DAPT 2020

Nos experimentos com o DAPT 2020, o ECSMiner apresentou a maior acurácia (0,689). Contudo, este resultado deve ser interpretado considerando o desbalanceamento de classes deste *dataset*: dada a predominância massiva da classe *Normal* e a postura conservadora do algoritmo (que pouco previu novidades), o ECSMiner maximizou os acertos nessa classe majoritária. Em contraste, o MINAS obteve a menor acurácia (0,095), devido

a uma classificação incorreta de grande parte do tráfego da classe *Normal* como sendo novidade, penalizando severamente sua pontuação global neste cenário específico.

Neste *dataset*, o MINAS registrou o maior H_{new} (0,578) entre as técnicas comparadas. Sua abordagem permitiu detectar a maioria das ameaças raras presentes no fluxo ($M_{new} = 0,132$), embora ao custo de elevar a taxa de falsos positivos para 0,567. Em contrapartida, o ECSMiner mostrou dificuldade neste perfil de tráfego com classes com poucas amostras, apresentando um M_{new} de 0,889. Ao classificar a quase totalidade dos novos ataques como classes conhecidas, o algoritmo obteve o pior H_{new} do cenário (0,200).

Importante observar que o ECSMiner (assim como o MINAS) classifica também amostras como "desconhecidas"enquanto estão no *buffer*, até o momento da atualização de modelo. Essas amostras recebem um rótulo específico e não são contabilizadas como erro ou acerto. No experimento do ECSMiner com o DAPT, por exemplo, 88,9% das amostras de novas classes foram erroneamente classificadas como classes conhecidas e as demais 11,1% permaneceram sem classificação. Em particular, neste experimento, o ECSMiner não identificou nenhuma amostra pertencente a novas classes.

Tabela 5.3: Métricas de capacidade preditiva no *dataset* DAPT 2020.

Algoritmo	**Acurácia**	M_{new}	F_{new}	H_{new}
ECSMiner	**0,689**	0,889	**0,011**	0,200
MINAS	0,095	**0,132**	0,567	**0,578**
SENCForest	0,163	0,285	0,683	0,440
KNNENS	0,500	0,882	0,172	0,207
Trident	0,608	0,745	0,116	0,396

Conforme descrito na Seção 4.2, o DAPT 2020 foi projetado para simular o ciclo de vida completo de um APT. Nesse cenário, os ataques não ocorrem de forma isolada, mas seguem uma sequência lógica de estágios. De acordo com [88], um ataque APT pode ser organizado em cinco fases: (1) Reconhecimento, (2) Manutenção de acesso, (3) Movimento lateral, (4) Exfiltração de dados e (5) Pós-exfiltração. As 4 primeiras fases estão contempladas no DAPT 2020.

Dada esta característica realística deste *dataset*, foi realizada uma análise da acurácia segmentada por fase do ataque, indicando a capacidade preditiva de cada algoritmo em cada estágio da ação maliciosa. A Tabela 5.4 detalha a composição do FCD representado pela porção de teste deste *dataset*, mapeando cada classe de ataque à sua respectiva fase e indicando também se a classe aparece no fluxo como conhecida ou nova.

A Tabela 5.5 informa a acurácia obtida por cada algoritmo em cada fase. A acurácia nas amostras da classe *Normal* também é apresentada.

A análise por fases revela comportamentos distintos que não são evidentes apenas nas métricas globais:

Tabela 5.4: Mapeamento das fases de ataque do DAPT 2020.

Fase do Ataque	Classes de Ataque	Status	Amostras
Tráfego normal	Normal	Conhecida	32.173
Reconhecimento	Account Discovery, Network Scan	Conhecida	2.420
	Account Bruteforce, Directory Bruteforce	Conhecida	8.558
Manutenção de acesso	Backdoor	Nova	20
	CSRF	Nova	7
	Command Injection	Nova	12
	Malware Download	Nova	2
	SQL Injection	Nova	84
Movimento lateral	Privilege Escalation	Nova	13
Exfiltração de dados	Exfiltração de dados	Nova	6

Tabela 5.5: Acurácia dos algoritmos por fase do ataque no DAPT 2020.

Fase	ECSMiner	MINAS	SENCForest	KNNENS	Trident
Tráfego normal	**0,9269**	0,0689	0,2124	0,5478	0,7867
Reconhecimento	0,0000	0,1707	0,0105	**0,3496**	0,0849
Manutenção de acesso	0,0000	0,2080	**0,6960**	0,0720	0,2640
Movimento lateral	0,0000	0,6154	**0,9231**	0,5385	0,4615
Exfiltração de dados	0,0000	0,5000	**0,6667**	0,1667	0,1667

1. Tráfego Normal e Reconhecimento: O ECSMiner apresentou excelente classificação do tráfego normal (92,69%). A identificação de um comportamento malicioso ainda na fase de *Reconhecimento*, por sua vez, é bastante importante para identificar o início de um ataque. Nesta etapa, o KNNENS obteve o melhor resultado e os demais algoritmos apresentaram desempenho bastante reduzido.

2. Manutenção de acesso: Esta fase indica que a intrusão já ocorreu e que o ator malicioso está se estabelecendo no ambiente. O SENCForest demonstrou superioridade na classificação, com acurácia de 69,6% nas amostras desta etapa. O ECSMiner, por sua vez, apresentou acurácia nula (0,00%).

3. Movimento Lateral e Exfiltração: Estas fases contêm o menor número de amostras (13 e 6, respectivamente), tornando a detecção extremamente desafiadora. O SENCForest manteve o melhor desempenho, detectando 92% do movimento lateral e 66% de exfiltração. Em contraste, o ECSMiner falhou novamente em classificar corretamente qualquer amostra.

Em síntese, o SENCForest apresentou a melhor capacidade de classificação dos ataques propriamente ditos, com melhor acurácia em 3 das 4 fases da ação maliciosa representada pelo DAPT 2020. Os resultados desta análise revelaram que o ECSMiner, embora excelente para identificar o tráfego normal, obteve um desempenho nulo no desafio de identificação dos ataques.

5.1.4 Capacidade preditiva ao longo do FCD

A análise da evolução da capacidade preditiva ao longo do FCD permite identificar como os algoritmos respondem às mudanças na composição do tráfego. Importa destacar que os três *datasets* utilizados representam cenários distintos de tráfego de rede. Esta configuração experimental permitiu analisar como cada algoritmo se comporta frente a diferentes padrões de surgimento de novas classes.

As Figuras 5.1, 5.2 e 5.3 apresentam a evolução da acurácia e do H_{new} para cada um dos três *datasets*. Para cada um dos gráficos, o eixo y do gráfico à esquerda apresenta os valores da acurácia, o eixo y do gráfico à direita apresenta os valores do H_{new} e o eixo x de ambos indica as amostras processadas no fluxo.

No UNSW-NB15, a evolução das métricas apresenta correlação com a estrutura do FCD representado pelo *dataset*, onde os algoritmos encontraram um cenário com uma única classe nova. A classe *Generic* surge a partir da amostra 10.001, representa aproximadamente 9% do fluxo de teste e está presente de forma intercalada com as classes conhecidas até aproximadamente a amostra 67.000. A presença de uma classe nova com volume representativo permitiu que a maioria dos algoritmos obtivesse bom desempenho em DN, com exceção do SENCForest. O Trident destacou-se por combinar a melhor acurácia com H_{new} elevado, sugerindo equilíbrio entre a classificação correta de classes conhecidas e a detecção das novidades.

A acurácia do Trident apresenta crescimento até atingir pico de aproximadamente 0,68 na região de 50.000 a 55.000 amostras. Esse pico coincide com um bloco de tráfego exclusivamente *Normal*, onde não há amostras da classe nova para serem erroneamente classificadas. Após a amostra 55.000, quando o tráfego misto retorna com presença da classe *Generic*, observa-se queda da acurácia para aproximadamente 0,57. O ECSMiner apresenta padrão similar, com pico de acurácia (aproximadamente 0,45) na mesma região, seguido de queda.

O comportamento do H_{new} revela aspectos importantes da dinâmica de detecção. Nas primeiras 10.000 amostras, quando ainda não há classes novas no fluxo, em geral os algoritmos apresentam H_{new} elevado, à exceção do SENCForest que permaneceu com H_{new} baixo (entre 0,17 e 0,24) ao longo de todo o fluxo. O KNNENS, MINAS e Trident iniciam com valores próximos de 1,0, refletindo o fato de que, na ausência de amostras de classes novas, o M_{new} é zero e o H_{new} depende apenas do F_{new}, que é baixo para esses algoritmos nesta região.

Com o surgimento da classe *Generic* (amostra 10.001), o comportamento dos algoritmos diverge. O KNNENS apresenta queda abrupta de H_{new} de aproximadamente 1,0 para 0,3. Essa queda indica que o algoritmo demonstrou baixa capacidade de detectar as primeiras amostras da classe nova, o que se pode confirmar em uma análise mais detalhada

das predições nesta região. Após esse impacto inicial, o KNNENS apresenta recuperação gradual do H_{new} até estabilizar em torno de 0,72, passando a detectar a classe *Generic* após processar um volume maior de amostras.

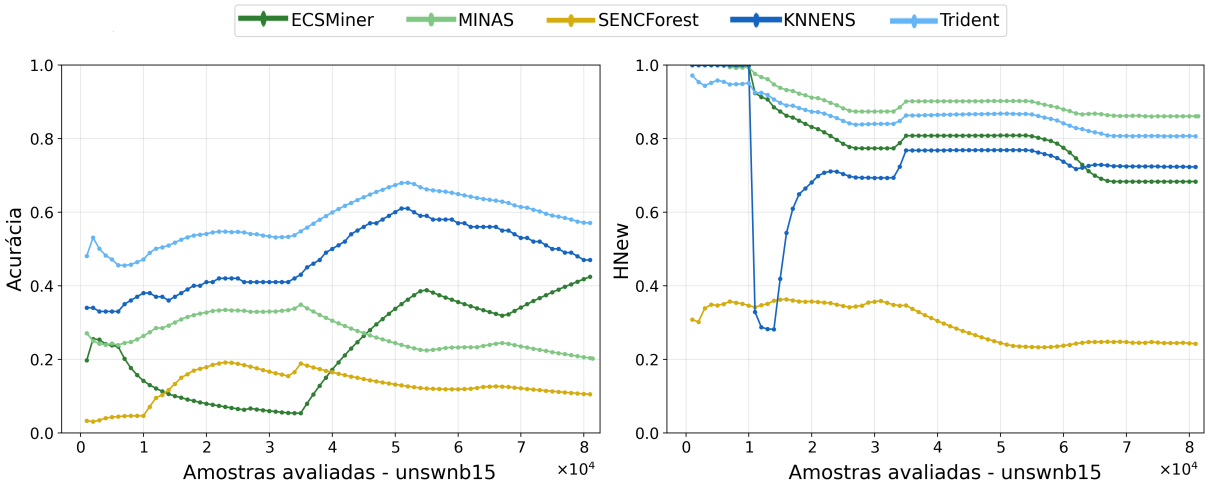

Figura 5.1: Evolução temporal da acurácia e H_{new} no *dataset* UNSW-NB15.

No ToN-IoT-V2, os algoritmos enfrentaram uma situação menos comum: cada classe aparece em um bloco contíguo, sem sobreposição temporal e nenhuma das demais classes conhecidas do treinamento reaparece no teste. O fluxo de teste inicia pela classe *Normal* (amostras 0 a 24.534), seguido pelo surgimento sequencial de quatro classes de ataque inéditas *Password* (24.535 a 44.363), *Ransomware* (44.364 a 58.629), *Scanning* (58.630 a 77.244) e *XSS* (77.245 a 91.758), que juntas compõem 73% do fluxo.

A acurácia de MINAS, SENCForest e Trident apresenta um padrão de queda semelhante nas primeiras 25.000 amostras aproximadamente, demonstrando dificuldade de predizer adequadamente a classe *Normal*. O ECSMiner manteve acurácia nula durante todo o bloco inicial da classe *Normal*. Uma análise mais detalhada das predições revela que o algoritmo classificou a maioria das amostras como desconhecidas e confundiu o restante com outras classes conhecidas do treinamento, sem contudo consolidar a detecção desta classe nesse período.

Durante o bloco inicial de tráfego *Normal* (amostras 0 a 24.534), não há classes novas para detectar. Nessa região, o KNNENS apresenta H_{new} próximo de 1,0, refletindo um baixo F_{new}. MINAS, Trident, ECSMiner e SENCForest apresentam queda gradual do H_{new} nessa mesma região, refletindo um aumento do F_{new} à medida que classificam erroneamente mais amostras da classe *Normal* como novidades.

Na transição para o bloco de classes novas (amostra 24.535), o KNNENS apresentou queda abrupta de H_{new} de aproximadamente 1,0 para zero, mostrando particular dificuldade em detectar amostras da classe *Password* e *Ransomware*. A recuperação do H_{new}

inicia-se de forma gradual durante o bloco de *Ransomware* e acelera significativamente durante o bloco de *Scanning*. Também foi possível perceber um comportamento bastante semelhante no Trident e ECSMiner, embora menos acentuado.

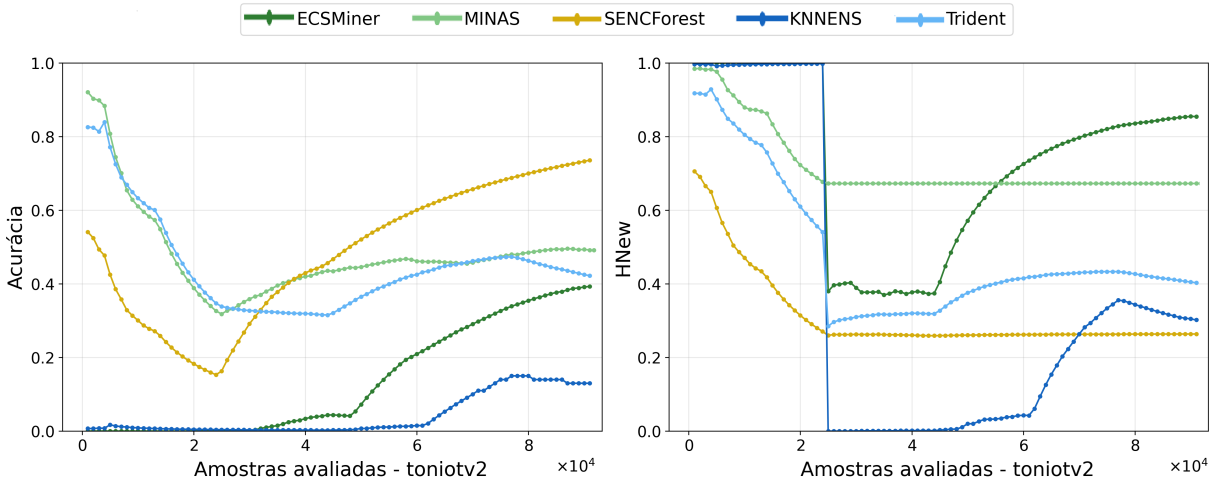

Figura 5.2: Evolução temporal da acurácia e H_{new} no *dataset* ToN-IoT-V2.

Em um cenário de APT, representado pelo DAPT 2020, os algoritmos enfrentaram o desafio de detectar ataques extremamente raros. Neste *dataset*, sete classes de ataque surgem ao longo do fluxo, porém representam apenas 0,3% do total de amostras. As classes novas emergem gradualmente ao longo do tempo, refletindo uma progressão temporal compatível com um ataque cibernético.

No DAPT 2020, observa-se um comportamento geral de queda da acurácia aproximadamente nas primeiras 10.000 amostras. O ECSMiner, por exemplo, inicia com acurácia próxima de 0,9 e atinge um mínimo de aproximadamente 0,2 nessa região. A análise da composição do fluxo revela que a classe *Directory Bruteforce* (conhecida) passa a dominar progressivamente nessa região. representando cerca de 79% das amostras na região 0-10.000. Esse comportamento de queda na acurácia nessa região pode ser explicado pela baixa representatividade dessa classe no conjunto de treinamento: *Directory Bruteforce* correspondia a apenas 3,5% das amostras de treino.

A partir da amostra 10.000, inicia-se uma estabilização/recuperação gradual da acurácia para ECSMiner, Trident, SENCForest e KNNENS. Essa região coincide com o aumento da proporção de tráfego da classe *Normal*, atingindo cerca de 90% do total de amostras da porção *online*. O ECSMiner apresenta a recuperação mais expressiva, atingindo acurácia final de 0,689. O MINAS manteve acurácia baixa ao longo de todo o fluxo (abaixo de 0,1), refletindo os efeitos de uma configuração com maior tendência a classificar amostras conhecidas como novidades (F_{new} elevado).

Quanto ao H_{new}, o SENCForest manteve valores razoavelmente estáveis, entre 0,35 e 0,45, enquanto o MINAS atingiu um pico de 0,578. O ECSMiner e o KNNENS mantiveram os piores desempenhos de DN ao longo de todo o fluxo, falhando em identificar os ataques raros.

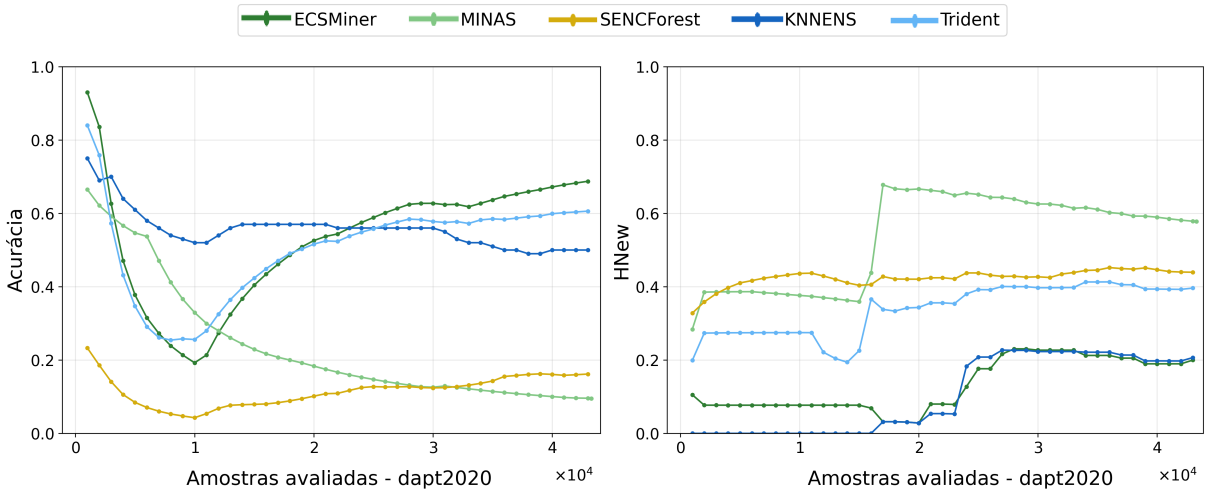

Figura 5.3: Evolução temporal da acurácia e H_{new} no *dataset* DAPT 2020.

5.1.5 Tempo de Detecção de Novidade

A métrica TTD proposta nesta pesquisa mede o intervalo de tempo entre o surgimento da primeira amostra de uma classe nova no fluxo e sua primeira classificação correta como novidade, conforme definido na Seção 4.5.2. Complementarmente ao TTD medido em segundos, foi registrado o intervalo em número de amostras processadas entre o surgimento da primeira amostra de nova classe e sua primeira detecção assertiva. A Tabela 5.6 apresenta os valores obtidos para cada algoritmo e *dataset*. Os valores faltantes referem-se a experimentos onde não houve identificação da nova classe em questão.

O SENCForest apresentou os menores TTD em todos os *datasets* (0,00s, 0,00s e 0,28s), demonstrando capacidade de detecção quase imediata de novidades. Este desempenho é sustentado pelo fato de que o SENCForest detecta novidades diretamente pelo mecanismo de *path length* da *Isolation Forest*, sem necessidade de acumular amostras em *buffer* antes da detecção. Vale ressaltar também que, nas configurações dos experimentos realizados, o SENCForest apresentou alta tendência em classificar amostras como novidades, com baixo M_{new} e alto F_{new}, colaborando para a detecção quase imediata apresentada.

A estimativa do TTD é um valor relevante para aplicações de IDS, visto que a detecção rápida de novos tipos de ataque torna o alerta uma informação mais acionável, favorecendo a resposta a incidentes. Algoritmos com TTD elevado podem permitir que ataques

Tabela 5.6: TTD em segundos e em amostras para cada algoritmo e *dataset*.

Algoritmo	Medida	UNSW-NB15	ToN-IoT-V2	DAPT 2020
ECSMiner	Segundos	–	0,34	634,76
	Amostras	–	9	19387
MINAS	Segundos	0,02	9,19	43,99
	Amostras	88	579	15441
SENCForest	Segundos	**0,00**	**0,00**	**0,28**
	Amostras	**1**	**1**	**3**
KNNENS	Segundos	72,77	0,26	–
	Amostras	9070	62	–
Trident	Segundos	3,41	–	153,42
	Amostras	146	–	19387

permaneçam não detectados por períodos significativos, comprometendo a segurança do ambiente monitorado.

5.2 Consumo de Memória

O consumo de memória RAM foi monitorado ao longo do processamento de cada *dataset*, com medições a cada 1.000 amostras processadas. As Figuras 5.4, 5.5 e 5.6 apresentam o comportamento de cada algoritmo nos três *datasets*. Para cada um dos gráficos, o eixo y apresenta os valores da memória alocada e o eixo x as amostras do fluxo.

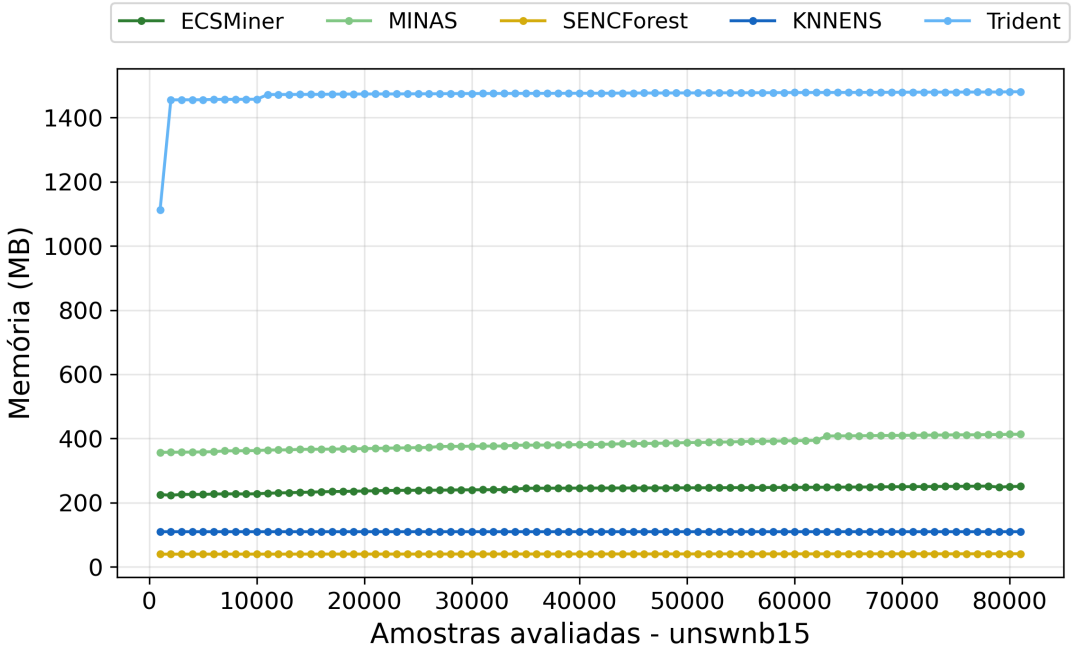

Figura 5.4: Consumo de memória ao longo do processamento do UNSW-NB15.

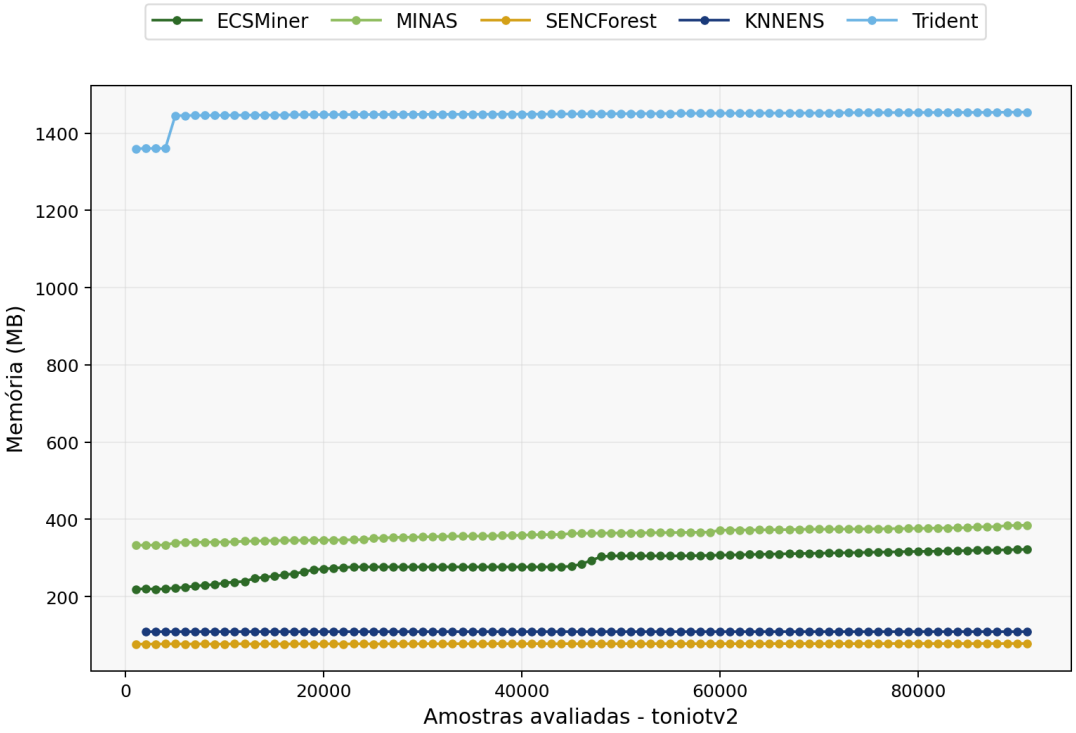

Figura 5.5: Consumo de memória ao longo do processamento do ToN-IoT-V2.

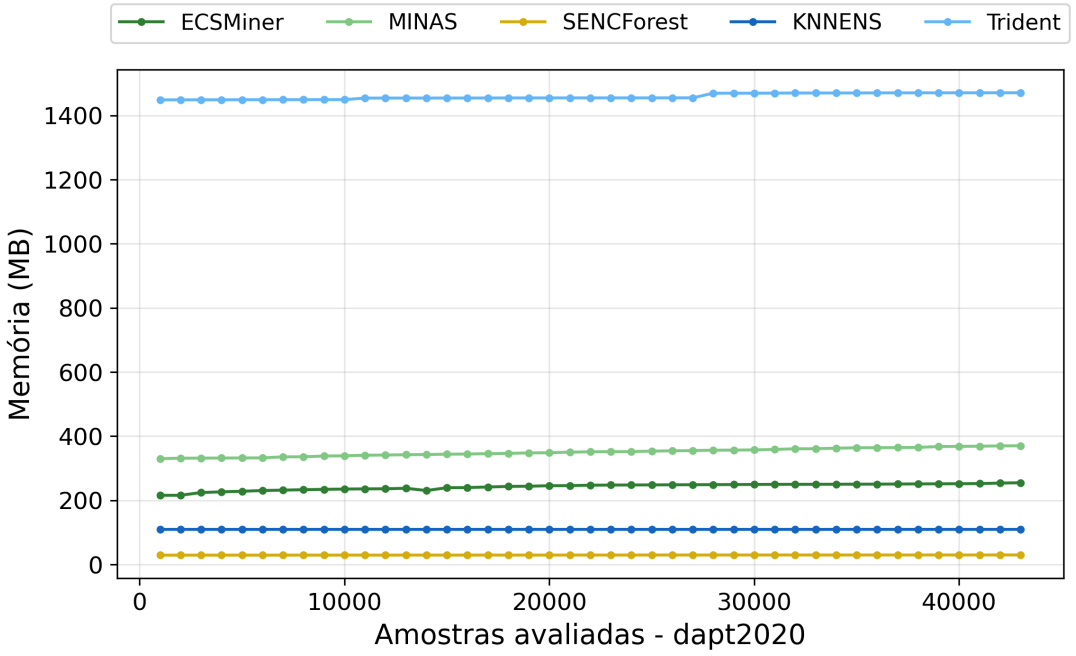

Figura 5.6: Consumo de memória ao longo do processamento do DAPT 2020.

O Trident apresentou maior consumo de memória que os demais algoritmos, enquanto o SENCForest obteve o menor consumo em todos os experimentos. No geral, todas as técnicas avaliadas mantiveram um consumo controlado de memória, refletindo a atuação de mecanismos de esquecimento.

5.3 Taxa de Processamento de Amostras

A taxa de processamento foi calculada como o número médio de amostras avaliadas por segundo durante a fase *online*. A Tabela 5.7 apresenta os valores obtidos para cada algoritmo em cada *dataset*, juntamente com o *throughput* estimado da rede simulada.

Tabela 5.7: Taxa de processamento e comparação com *throughput* do *dataset*.

Dataset	**Algoritmo**	**Taxa (amostras/s)**	***Throughput***
UNSW-NB15	ECSMiner	49,82	4,07
	MINAS	**1114,50**	
	SENCForest	5,36	
	KNNENS	48,84	
	Trident	47,94	
ToN-IoT-V2	ECSMiner	6,08	47,10
	MINAS	**113,40**	
	SENCForest	2,16	
	KNNENS	44,93	
	Trident	60,92	
DAPT 2020	ECSMiner	31,65	0,50
	MINAS	**379,02**	
	SENCForest	4,82	
	KNNENS	68,31	
	Trident	158,53	

O MINAS destacou-se como o algoritmo mais rápido em todos os *datasets*. O SENCForest, por sua vez, apresentou as menores taxas de processamento, com desempenho médio bastante inferior aos demais.

A comparação da taxa média de processamento de cada algoritmo com o *throughput* médio de cada dataset oferece uma noção sobre a capacidade de cada técnica de atender ao requisito de processamento dos dados em tempo real sem causar atrasos acumulativos na avaliação. No DAPT 2020, com *throughput* de apenas 0,50 amostras/s, todos os algoritmos atendem a este requisito, assim como no UNSW-NB15. No ToN-IoT-V2, entretanto, com *throughput* de 47,10 amostras/s, apenas o MINAS (113,40 amostras/s) e o Trident (60,92 amostras/s) demonstram capacidade de processamento em tempo real adequada. Em uma rede de computadores com perfil de tráfego semelhante ao ToN-IoT-V2, ECSMiner, SENCForest e KNNENS apresentariam gargalos de processamento durante sua operação.

Esses resultados evidenciam o compromisso entre capacidade preditiva e eficiência computacional. O MINAS, embora com limitações em acurácia em alguns *datasets*, destacou-se pela velocidade de processamento e pelo bom desempenho em H_{new}. O SENCForest, apesar de resultados variáveis em DN, apresentou limitações críticas de velocidade que podem inviabilizar aplicações em redes de alto tráfego.

5.4 Considerações Finais

Este capítulo apresentou os resultados dos experimentos de avaliação dos cinco algoritmos de DN nos três *datasets* selecionados. A análise da capacidade preditiva revelou que nenhum algoritmo se mostrou uniformemente superior: o MINAS obteve o melhor H_{new} no UNSW-NB15 e no DAPT 2020, enquanto o ECSMiner se destacou no ToN-IoT-V2. A análise por fases do DAPT 2020 evidenciou que métricas globais podem mascarar comportamentos críticos. Em relação ao TTD, o SENCForest demonstrou detecção quase imediata em todos os cenários. Quanto à eficiência computacional, todos os algoritmos mantiveram consumo de memória controlado ao longo do fluxo; entretanto, apenas o MINAS e o Trident demonstraram taxa de processamento adequada para o cenário de alto *throughput* representado pelo ToN-IoT-V2. O próximo capítulo apresenta as conclusões desta pesquisa, discutindo as contribuições obtidas, as limitações identificadas e as direções para trabalhos futuros.

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho investigou o desempenho de algoritmos de DN na tarefa de detecção de intrusão em redes de computadores, avaliando cinco técnicas relevantes, ECSMiner, MINAS, SENCForest, KNNENS e Trident, em três *datasets* contemporâneos de tráfego de rede: UNSW-NB15, ToN-IoT-V2 e DAPT 2020. A pesquisa adotou uma abordagem experimental abrangente, dando ênfase a métricas de capacidade preditiva, mas também abordando aspectos de eficiência computacional, essenciais para avaliar a viabilidade de implementação dessas técnicas em sistemas de detecção de intrusão.

6.1 Síntese dos Principais Resultados

Os experimentos revelaram que não existe um algoritmo uniformemente superior em todos os cenários avaliados. O desempenho de cada técnica mostrou-se dependente das características do tráfego de rede, particularmente quanto à proporção e ao padrão temporal de surgimento de novas classes de ataque.

No UNSW-NB15, onde há uma única classe nova, percentualmente bem representada no *dataset*, o Trident alcançou o melhor resultado de acurácia, indicando boa capacidade de generalização. O MINAS obteve o melhor H_{new}, demonstrando o equilíbrio mais efetivo na tarefa de DN. O SENCForest obteve o pior desempenho em ambas as métricas neste cenário.

No ToN-IoT-V2, caracterizado por um padrão onde a maior parte do tráfego de teste corresponde a novas classes que surgem sequencialmente sem sobreposição temporal, o ECSMiner destacou-se com o melhor H_{new}, impulsionado por uma taxa de falsos novos nula. O SENCForest obteve a maior acurácia global. O MINAS apresentou a menor taxa de perda de novos ataques (menor M_{new}), mas teve seu desempenho geral penalizado por uma taxa de falsos novos elevada.

No cenário de tráfego representado pelo DAPT 2020, caracterizado por algumas classes extremamente raras, o MINAS obteve o maior H_{new}, sendo a técnica que demonstrou maior capacidade para identificar as classes emergentes, ainda que ao custo de gerar mais falsos alarmes. Uma análise mais detalhada neste *dataset* revelou um aspecto crítico frequentemente oculto por métricas globais: o ECSMiner, apesar de apresentar a maior acurácia geral, obteve acurácia nula na classificação dos ataques. Este resultado evidenciou que sua alta acurácia decorreu da classificação correta da classe majoritária (*Normal*), não de uma capacidade discriminativa real. Em contraste, o SENCForest demonstrou a melhor capacidade de classificação dos ataques propriamente ditos, com acurácia superior em três das quatro fases do APT representado pelo *dataset*.

Quanto à eficiência computacional, o MINAS destacou-se consistentemente como o algoritmo mais rápido em todos os *datasets*, sendo capaz, juntamente com o Trident, de atender aos requisitos de processamento em tempo real de todas as bases de dados. O SENCForest, apesar do excelente desempenho em DN e TTD em alguns cenários, apresentou as menores taxas de processamento, o que poderia inviabilizar sua aplicação em redes de alto tráfego. Todos os algoritmos demonstraram gerenciamento aceitável de memória, com consumo controlado ao longo do processamento dos fluxos.

6.2 Validação das Hipóteses

Retomando as hipóteses apresentadas no Capítulo 1, os resultados experimentais permitem as seguintes conclusões:

Hipótese 1: *As técnicas de DN avaliadas são capazes de classificar corretamente amostras de classes conhecidas e identificar novas classes de intrusão em tráfego de redes de computadores, sem ter sua capacidade preditiva degradada ao longo do fluxo.*

Esta hipótese foi parcialmente validada. Os algoritmos demonstraram capacidade de DN, porém com desempenho variável conforme o *dataset*. Em cenários com uma única classe nova, bem representada numericamente (UNSW-NB15), a maioria dos algoritmos obteve bom desempenho na tarefa de DN. Em cenários com classes extremamente raras (DAPT 2020), o MINAS demonstrou capacidade de DN superior, enquanto outros algoritmos falharam mais em identificar os novos padrões. A classificação correta de classes conhecidas também variou significativamente, indicando desafios na identificação multiclasse.

A análise da evolução temporal das métricas revelou comportamentos distintos entre os algoritmos. O surgimento de classes novas provocou impacto inicial na capacidade preditiva de alguns algoritmos, seguido por recuperação gradual, sugerindo adaptação aos novos padrões. No entanto, essa capacidade de adaptação não foi uniforme: enquanto

alguns algoritmos demonstraram recuperação mais consistente do H_{new} após o impacto inicial, outros mantiveram desempenho degradado ao longo de todo o fluxo.

Hipótese 2: *As técnicas de DN avaliadas são capazes de processar sequencialmente as amostras de tráfego no mínimo na mesma taxa de transferência do FCD e com consumo de memória estável, garantindo a capacidade de análise em tempo real.*

Esta hipótese foi parcialmente validada. Nos *datasets* com *throughput* mais baixo (DAPT2020 e UNSW-NB15), todos os algoritmos atenderam aos requisitos de processamento em tempo real. No ToN-IoT-V2, com *throughput* bem superior aos demais, apenas MINAS e Trident demonstraram capacidade adequada. ECSMiner, SENCForest e KNNENS apresentariam gargalos em redes com perfil similar, comprometendo a viabilidade operacional. O consumo de memória principal permaneceu controlado ao longo do fluxo em todos os experimentos.

6.3 Contribuições da Pesquisa

Este trabalho aporta contribuições ao campo de detecção de novidade aplicada a sistemas de detecção de intrusão:

1. Avaliação experimental abrangente: condução de experimentos sistemáticos com cinco algoritmos de DN em três *datasets* contemporâneos de IDS, preenchendo lacuna identificada na literatura quanto ao uso predominante de bases obsoletas.

2. Métrica H_{new}: proposição de métrica balanceada que combina M_{new} e F_{new} via média harmônica, fornecendo medida unificada da capacidade de DN que penaliza desproporcionalidades e facilita comparação entre algoritmos.

3. Avaliação multidimensional: análise que contempla simultaneamente capacidade preditiva (acurácia, M_{new}, F_{new}, H_{new}), eficiência computacional (consumo de memória, taxa de processamento) e uma estimativa do tempo de detecção do surgimento de uma nova classe de ataque, oferecendo visão geral dos compromissos envolvidos na seleção de algoritmos para IDS.

4. Análise por perfil de tráfego: avaliação de como características do tráfego, tais como proporção de classes novas e padrão temporal de surgimento, impactam no desempenho relativo dos algoritmos, permitindo uma compreensão sobre o desempenho de cada algoritmo em diferentes cenários.

6.4 Limitações do Estudo e Trabalhos Futuros

Apesar das contribuições, o estudo apresenta limitações. Estas restrições podem ser oportunamente abordadas em trabalhos futuros.

1. Escopo de algoritmos avaliados: embora os cinco algoritmos selecionados representam uma boa diversidade de abordagens (baseadas em agrupamento, AD, KNN e *autoencoders*), há outras técnicas de DN multiclasse não contempladas, que poderiam dar uma visão mais abrangente das diversas estratégias de DN.

2. Ajuste de hiperparâmetros: a otimização via *Grid Search* foi conduzida em espaços de busca relativamente restritos, priorizando configurações documentadas na literatura. Outras técnicas de otimização mais sofisticadas poderiam ser avaliadas, dada a importância de uma boa configuração de hiperparâmetros.

3. Características dos *datasets*: embora mais contemporâneos que KDD99, os *datasets* utilizados foram coletados de forma controlada em ambiente de laboratório. Amostras de tráfego coletadas de redes reais podem apresentar características adicionais, tais como maior presença de ruído ou mudança de conceito, não capturadas pelos *datasets* com ataques emulados em laboratório.

4. Cenário de *feedback*: Os experimentos avaliaram técnicas que não utilizam *feedback* de rótulos verdadeiros durante a fase *online*. Cenários com *feedback* parcial ou atrasado, mais próximos de ambientes reais onde analistas podem fornecer rótulos, não foram explorados nesta pesquisa.

Referências

[1] CrowdStrike: *2025 global threat report*. Relatório Técnico, CrowdStrike, Inc., 2025. `https://www.crowdstrike.com/en-us/global-threat-report/`, acesso em 2025-12-10. 1

[2] Nguyen, Hai Long, Yew Kwong Woon e Wee Keong Ng: *A survey on data stream clustering and classification*. Knowledge and information systems, 45:535–569, 2015. 1

[3] Prachi, H Malhotra e Prabha Sharma: *Intrusion detection using machine learning and feature selection*. International Journal of Computer Network and Information security, 11(4):43–52, 2019. 1, 30

[4] Neupane, Subash, Jesse Ables, William Anderson, Sudip Mittal, Shahram Rahimi, Ioana Banicescu e Maria Seale: *Explainable intrusion detection systems (x-ids): A survey of current methods, challenges, and opportunities*. IEEE Access, 10:112392–112415, 2022. 1

[5] Kilincer, Ilhan Firat, Fatih Ertam e Abdulkadir Sengur: *Machine learning methods for cyber security intrusion detection: Datasets and comparative study*. Computer Networks, 188:107840, 2021. 1, 2, 30

[6] Saranya, T, S Sridevi, C Deisy, Tran Duc Chung e MKA Ahamed Khan: *Performance analysis of machine learning algorithms in intrusion detection system: A review*. Procedia Computer Science, 171:1251–1260, 2020. 1, 2

[7] Assy, Ahmed Tamer, Yahia Mostafa, Ahmed Abd El-khaleq e Maggie Mashaly: *Anomaly-based intrusion detection system using one-dimensional convolutional neural network*. Procedia Computer Science, 220:78–85, 2023. 2, 7

[8] Hidayat, Imran, Muhammad Zulfiqar Ali e Arshad Arshad: *Machine learning-based intrusion detection system: an experimental comparison*. Journal of Computational and Cognitive Engineering, 2(2):88–97, 2023. 2, 30

[9] Seth, S, G Singh e K Chahal: *Drift-based approach for evolving data stream classification in intrusion detection system*. Em *Proceedings of the Workshop on Computer Networks & Communications, Goa, India*, páginas 23–30, 2021. 2

[10] Paiva, Elaine Ribeiro de Faria: *Detecção de novidade em fluxos contínuos de dados multiclasse*. Tese de Doutoramento, Universidade de São Paulo, 2014. 2, 3, 10, 11, 12, 15, 27, 44

[11] Gama, Joao: *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, Florida, USA, 1st edição, 2010, ISBN 1439826110. 2, 9, 11, 12

[12] Zhang, Jianjun, Ting Wang, Wing WY Ng e Witold Pedrycz: *Knnens: A k-nearest neighbor ensemble-based method for incremental learning under data stream with emerging new classes*. IEEE transactions on neural networks and learning systems, 34(11):9520–9527, 2022. 2, 3, 20, 44

[13] Masud, Mohammad, Jing Gao, Latifur Khan, Jiawei Han e Bhavani M Thuraisingham: *Classification and novel class detection in concept-drifting data streams under time constraints*. IEEE Transactions on knowledge and data engineering, 23(6):859–874, 2010. 2, 3, 14, 27, 33, 44

[14] Mu, Xin, Kai Ming Ting e Zhi Hua Zhou: *Classification under streaming emerging new classes: A solution using completely-random trees*. IEEE Transactions on Knowledge and Data Engineering, 29(8):1605–1618, 2017. 2, 3, 18, 33, 34, 44

[15] Zhu, Yue, Kai Ming Ting e Zhi Hua Zhou: *Multi-label learning with emerging new labels*. IEEE Transactions on Knowledge and Data Engineering, 30(10):1901–1914, 2018. 2, 3

[16] Cai, Xin Qiang, Peng Zhao, Kai Ming Ting, Xin Mu e Yuan Jiang: *Nearest neighbor ensembles: An effective method for difficult problems in streaming classification with emerging new classes*. Em *2019 IEEE international conference on data mining (ICDM)*, páginas 970–975. IEEE, 2019. 2, 3

[17] Zhu, Yong Nan e Yu Feng Li: *Semi-supervised streaming learning with emerging new labels*. Em *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, páginas 7015–7022, 2020. 2, 3

[18] Gao, Yang, Swarup Chandra, Yifan Li, Latifur Khan e Thuraisingham Bhavani: *Saccos: A semi-supervised framework for emerging class detection and concept drift adaption over data streams*. IEEE Transactions on Knowledge and Data Engineering, 34(3):1416–1426, 2020. 2, 3, 33

[19] Zhao, Ziming, Zhaoxuan Li, Zhuoxue Song, Wenhao Li e Fan Zhang: *Trident: A universal framework for fine-grained and class-incremental unknown traffic detection*. Em *Proceedings of the ACM Web Conference 2024*, páginas 1608–1619, 2024. 2, 3, 33, 35, 44

[20] Din, Salah Ud, Junming Shao, Jay Kumar, Cobbinah Bernard Mawuli, SM Hasan Mahmud, Wei Zhang e Qinli Yang: *Data stream classification with novel class detection: a review, comparison and challenges*. Knowledge and Information Systems, 63:2231–2276, 2021. 3, 9, 11, 12, 14, 27

[21] Spinosa, Eduardo J, André Ponce de Leon F. de Carvalho e Joao Gama: *Olindda: A cluster-based approach for detecting novelty and concept drift in data streams*. Em *Proceedings of the 2007 ACM symposium on Applied computing*, páginas 448–452, 2007. 3

[22] Haque, Ahsanul, Latifur Khan e Michael Baron: *Sand: Semi-supervised adaptive novel class detection and classification over data stream.* Em *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 3

[23] Mu, Xin, Feida Zhu, Juan Du, Ee Peng Lim e Zhi Hua Zhou: *Streaming classification with emerging new class by class matrix sketching.* Em *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 3

[24] Zhou, Peng, Ni Wang, Shu Zhao, Yanping Zhang e Xindong Wu: *Difficult novel class detection in semisupervised streaming data.* IEEE Transactions on Neural Networks and Learning Systems, 34(10):6872–6886, 2022. 3

[25] Wang, Yu e Alexey Vinogradov: *Multi-classification generative adversarial network for streaming data with emerging new classes: method and its application to condition monitoring.* Authorea Preprints, 2023. 3

[26] Gaudreault, Jean Gabriel e Paula Branco: *A systematic literature review of novelty detection in data streams: Challenges and opportunities.* ACM Computing Surveys, 56(10):1–37, 2024. 3, 13, 24, 32, 49

[27] Farid, Dewan Md e Chowdhury Mofizur Rahman: *Novel class detection in concept-drifting data stream mining employing decision tree.* Em *2012 7th international conference on electrical and computer engineering*, páginas 630–633. IEEE, 2012. 3, 17

[28] Cassales, Guilherme Weigert, Hermes Senger, Elaine Ribeiro de Faria e Albert Bifet: *Idsa-iot: an intrusion detection system architecture for iot networks.* Em *2019 IEEE Symposium on Computers and Communications (ISCC)*, páginas 1–7. IEEE, 2019. 3, 33, 34

[29] Fuhrman, Sean, Onat Gungor e Tajana Rosing: *Cnd-ids: Continual novelty detection for intrusion detection systems.* arXiv preprint arXiv:2502.14094, 2025. 3

[30] Hettich, S. e S. D. Bay: *Kdd cup 1999 data*, 1999. `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, Accessed: 2024-07-25. 3

[31] Rahman, Md Mahbubur, Shaharia Al Shakil e Mizanur Rahman Mustakim: *A survey on intrusion detection system in iot networks.* Cyber Security and Applications, 3:100082, 2025. 6, 7

[32] Kurose, James F. e Keith W. Ross: *Redes de computadores e a internet: uma abordagem top-down.* Bookman, 8ª edição, 2013, ISBN 9788543014432. 6, 7

[33] Martins, Ines, Joao S Resende, Patricia R Sousa, Simao Silva, Luis Antunes e Joao Gama: *Host-based ids: A review and open issues of an anomaly detection system in iot.* Future Generation Computer Systems, 133:95–113, 2022. 6

[34] Li, Yuchong e Qinghui Liu: *A comprehensive review study of cyber-attacks and cyber security; emerging trends and recent developments.* Energy Reports, 7:8176–8186, 2021. 7, 9

[35] Maharana, Kiran, Surajit Mondal e Bhushankumar Nemade: *A review: Data preprocessing and data augmentation techniques.* Global Transitions Proceedings, 3(1):91–99, 2022. 7

[36] Miani, Rodrigo Sanches, Gustavo Di Giovanni Bernardo, Guilherme Weigert Cassales, Hermes Senger e Elaine Ribeiro Faria: *A survey of data stream-based intrusion detection systems.* IEEE Access, 2025. 8, 31, 32, 36

[37] Souza, Vinicius MA, Denis M dos Reis, Andre G Maletzke e Gustavo EAPA Batista: *Challenges in benchmarking stream learning algorithms with real-world data.* Data Mining and Knowledge Discovery, 34(6):1805–1858, 2020. 9, 49

[38] Al-amri, Redhwan, Raja Kumar Murugesan, Mustafa Man, Alaa Fareed Abdulateef, Mohammed A Al-Sharafi e Ammar Ahmed Alkahtani: *A review of machine learning and deep learning techniques for anomaly detection in iot data.* Applied Sciences, 11(12):5320, 2021. 9, 11

[39] De Lange, Matthias e Tinne Tuytelaars: *Continual prototype evolution: Learning online from non-stationary data streams.* Em *Proceedings of the IEEE/CVF international conference on computer vision*, páginas 8250–8259, 2021. 9

[40] Agrahari, Supriya e Anil Kumar Singh: *Concept drift detection in data stream mining: A literature review.* Journal of King Saud University-Computer and Information Sciences, 34(10):9523–9540, 2022. 9

[41] Suárez-Cetrulo, Andrés L, David Quintana e Alejandro Cervantes: *A survey on machine learning for recurring concept drifting data streams.* Expert Systems with Applications, 213:118934, 2023. 9

[42] Korycki, Łukasz e Bartosz Krawczyk: *Adversarial concept drift detection under poisoning attacks for robust data stream mining.* Machine Learning, 112(10):4013–4048, 2023. 9

[43] Hoi, Steven CH, Doyen Sahoo, Jing Lu e Peilin Zhao: *Online learning: A comprehensive survey.* Neurocomputing, 459:249–289, 2021. 10

[44] Gomes, Heitor Murilo, Jesse Read, Albert Bifet, Jean Paul Barddal e João Gama: *Machine learning for streaming data: state of the art, challenges, and opportunities.* ACM SIGKDD Explorations Newsletter, 21(2):6–22, 2019. 10, 39

[45] Bifet, Albert, Ricard Gavalda, Geoffrey Holmes e Bernhard Pfahringer: *Machine learning for data streams: with practical examples in MOA.* MIT press, 2018. 10

[46] Bahri, Maroua, Albert Bifet, João Gama, Heitor Murilo Gomes e Silviu Maniu: *Data stream analysis: Foundations, major tasks and tools.* Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 11(3):e1405, 2021. 10

[47] Molina-Coronado, Borja, Usue Mori, Alexander Mendiburu e Jose Miguel-Alonso: *Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process.* IEEE Transactions on Network and Service Management, 17(4):2451–2479, 2020. 10

[48] Garcia, Kemilly Dearo, Mannes Poel, Joost N Kok e André CPLF de Carvalho: *Online clustering for novelty detection and concept drift in data streams*. Em *Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3–6, 2019, Proceedings, Part II 19*, páginas 448–459. Springer, 2019. 11, 19

[49] Žliobaitė, Indrė, Mykola Pechenizkiy e Joao Gama: *An overview of concept drift applications*. Big data analysis: new algorithms for a new society, páginas 91–114, 2016. 11

[50] Schlkopf, B, R Williamson, A Smola, J Shawe-Taylor e J Platt: *Support vector method for novelty detection, solla sa, leen tk, muller kr, editors*. Advances in Neural Information Processing Systems, 12, 2000. 13

[51] Yu, Hwanjo: *Single-class classification with mapping convergence*. Machine Learning, 61:49–69, 2005. 13

[52] Agrahari, Supriya, Sakshi Srivastava e Anil Kumar Singh: *Review on novelty detection in the non-stationary environment*. Knowledge and Information Systems, 66(3):1549–1574, 2024. 13, 14

[53] Liu, Fei Tony, Kai Ming Ting e Zhi Hua Zhou: *Isolation forest*. Em *2008 eighth ieee international conference on data mining*, páginas 413–422. IEEE, 2008. 18

[54] Liu, Fei Tony, Kai Ming Ting, Yang Yu e Zhi Hua Zhou: *Spectrum of variable-random trees*. Journal of Artificial Intelligence Research, 32:355–384, 2008. 18

[55] Aggarwal, Charu C, S Yu Philip, Jiawei Han e Jianyong Wang: *A framework for clustering evolving data streams*. Em *Proceedings 2003 VLDB conference*, páginas 81–92. Elsevier, 2003. 19

[56] Zhou, Da Wei, Yang Yang e De Chuan Zhan: *Learning to Classify With Incremental New Class*. IEEE Transactions on Neural Networks and Learning Systems, 33(6):2429–2443, 2022. 21, 33, 35

[57] Carreño, Ander, Iñaki Inza e Jose A Lozano: *Sndprob: A probabilistic approach for streaming novelty detection*. IEEE Transactions on Knowledge and Data Engineering, 35(6):6335–6348, 2022. 23

[58] Gama, João, Pedro Pereira Rodrigues e Gladys Castillo: *Evaluating algorithms that learn from data streams*. Em *ICML 2008 Workshop on Evaluation Methods for Machine Learning (The Third Workshop on Evaluation Methods for Machine Learning)*, Helsinki, Finland, 2008. Held in conjunction with ICML 2008 (Joint workshop day: July 9, 2008). Available as a workshop paper (PDF). 24

[59] Hossin, Mohammad e Md Nasir Sulaiman: *A review on evaluation metrics for data classification evaluations*. International journal of data mining & knowledge management process, 5(2):1, 2015. 26

[60] Van Thieu, Nguyen: *Permetrics: A framework of performance metrics for machine learning models*. Journal of Open Source Software, 9(95):6143, 2024. 26

[61] Faria, Elaine Ribeiro de, Isabel Ribeiro Goncalves, Joao Gama, Andre Carlos Ponce de Leon Ferreira *et al.*: *Evaluation of multiclass novelty detection algorithms for data streams.* IEEE Transactions on Knowledge and Data Engineering, 27(11):2961–2973, 2015. 26, 27, 33, 34

[62] Farnaaz, Nabila e MA Jabbar: *Random forest modeling for network intrusion detection system.* Procedia Computer Science, 89:213–217, 2016. 30

[63] Belavagi, Manjula C e Balachandra Muniyal: *Performance evaluation of supervised machine learning algorithms for intrusion detection.* Procedia Computer Science, 89:117–123, 2016. 30

[64] Taher, Kazi Abu, Billal Mohammed Yasin Jisan e Md Mahbubur Rahman: *Network intrusion detection using supervised machine learning technique with feature selection.* Em *2019 International conference on robotics, electrical and signal processing techniques (ICREST)*, páginas 643–646. IEEE, 2019. 30

[65] Li, Yinhui, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai e Kuobin Dai: *An efficient intrusion detection system based on support vector machines and gradually feature removal method.* Expert systems with applications, 39(1):424–430, 2012. 30

[66] Islam, Nahida, Fahiba Farhin, Ishrat Sultana, M Shamim Kaiser, Md Sazzadur Rahman, Mufti Mahmud, ASM SanwarHosen e Gi Hwan Cho: *Towards machine learning based intrusion detection in iot networks.* Computers, Materials & Continua, 69(2), 2021. 30

[67] Meira, Jorge, Rui Andrade, Isabel Praça, João Carneiro e Goreti Marreiros: *Comparative results with unsupervised techniques in cyber attack novelty detection.* Em *Ambient Intelligence–Software and Applications–, 9th International Symposium on Ambient Intelligence*, páginas 103–112. Springer, 2019. 30

[68] Whelan, Jason, Thanigajan Sangarapillai, Omar Minawi, Abdulaziz Almehmadi e Khalil El-Khatib: *Novelty-based intrusion detection of sensor attacks on unmanned aerial vehicles.* Em *Proceedings of the 16th ACM symposium on QoS and security for wireless and mobile networks*, páginas 23–28, 2020. 30

[69] Yang, Kun, Samory Kpotufe e Nick Feamster: *Feature extraction for novelty detection in network traffic.* arXiv preprint arXiv:2006.16993, 2020. 30

[70] Leichtnam, Laetitia, Eric Totel, Nicolas Prigent e Ludovic Mé: *Sec2graph: Network attack detection based on novelty detection on graph structured data.* Em *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, páginas 238–258. Springer, 2020. 30

[71] Hamad, Salma Abdalla, Quan Z Sheng, Dai Hoang Tran, Wei Emma Zhang e Surya Nepal: *A behavioural network traffic novelty detection for the internet of things infrastructures.* Em *Parallel Architectures, Algorithms and Programming: 11th International Symposium, PAAP 2020, Shenzhen, China, December 28–30, 2020, Proceedings 11*, páginas 174–186. Springer, 2021. 30

[72] Campazas-Vega, Adrián, Ignacio Samuel Crespo-Martínez, Ángel Manuel Guerrero-Higueras, Claudia Álvarez-Aparicio, Vicente Matellán e Camino Fernández-Llamas: *Malicious traffic detection on sampled network flow data with novelty-detection-based models*. Scientific Reports, 13(1):15446, 2023. 30

[73] Pimentel, Marco AF, David A Clifton, Lei Clifton e Lionel Tarassenko: *A review of novelty detection*. Signal processing, 99:215–249, 2014. 31

[74] Faria, Elaine R., Isabel J. C. R. Gonçalves, André C. P. L. F. de Carvalho e João Gama: *Novelty detection in data streams*. Artificial Intelligence Review, 45(2):235–269, 2016. 31

[75] Al-Khateeb, Tahseen, Mohammad M Masud, Latifur Khan, Charu Aggarwal, Jiawei Han e Bhavani Thuraisingham: *Stream classification with recurring and novel class detection using class-based ensemble*. Em *2012 IEEE 12th international conference on data mining*, páginas 31–40. IEEE, 2012. 33

[76] Siahroudi, Sajjad Kamali, Poorya ZareMoodi e Hamid Beigy: *Detection of evolving concepts in non-stationary data streams: A multiple kernel learning approach*. Expert Systems with Applications, 91:187–197, 2018. 33, 34

[77] Cristiani, Andre L., Douglas D. Lieira, Rodolfo I. Meneguette, Heloisa A. Camargo e R. Velazquez: *A fuzzy intrusion detection system for identifying cyber-attacks on iot networks*. Em *2020 IEEE Latin-American Conference on Communications (LATINCOM)*, páginas 1–6. IEEE, 2020. 33, 34

[78] Gama, João, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy e Abdelhamid Bouchachia: *A survey on concept drift adaptation*. ACM computing surveys (CSUR), 46(4):1–37, 2014. 38, 39, 49

[79] Goodfellow, Ian, Yoshua Bengio e Aaron Courville: *Deep Learning*. MIT Press, 2016. 39

[80] Kuhn, Max e Kjell Johnson: *Applied Predictive Modeling*. Springer, 2013. 39

[81] Ring, Markus, Sarah Wunderlich, Deniz Scheuring, Dieter Landes e Andreas Hotho: *A survey of network-based intrusion detection data sets*. Computers & Security, 86:147–167, 2019. 41

[82] Milenkoski, Aleksandar, Marco Vieira, Samuel Kounev, Alberto Avritzer e Bryan D. Payne: *Evaluating computer intrusion detection systems: A survey of common practices*. ACM Computing Surveys, 48(1):12:1–12:41, 2015. 41

[83] Wasielewska, Karolina, Dominik Soukup, Tomáš Čejka e Joaquín Camacho: *Evaluation of the limit of detection in network dataset quality assessment with perqoda*. Em *Machine Learning and Principles and Practice of Knowledge Discovery in Databases: ECML PKDD 2022*, volume 1753 de *Communications in Computer and Information Science*. Springer, 2023. 41

[84] Ayoubi, Solayman, Gregory Blanc, Houda Jmila, Sébastien Tixeuil e Thomas Silverston: *Evaluation framework for ml-based ids.* Em *RESSI 2023: Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*, 2023. 41, 49

[85] Mondragon, Jose Carlos, Paula Branco, Guy Vincent Jourdan, Andres Eduardo Gutierrez-Rodriguez e Rajesh Roshan Biswal: *Advanced ids: a comparative study of datasets and machine learning algorithms for network flow-based intrusion detection systems.* Applied Intelligence, 55(7):608, 2025. 41

[86] Moustafa, Nour e Jill Slay: *Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set).* Em *2015 military communications and information systems conference (MilCIS)*, páginas 1–6. IEEE, 2015. 41

[87] Moustafa, Nour: *A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets.* Sustainable Cities and Society, 72:102994, 2021. 41

[88] Myneni, Sowmya, Ankur Chowdhary, Abdulhakim Sabur, Sailik Sengupta, Garima Agrawal, Dijiang Huang e Myong Kang: *Dapt 2020-constructing a benchmark dataset for advanced persistent threats.* Em *Deployable Machine Learning for Security Defense: First International Workshop, MLHat 2020, San Diego, CA, USA, August 24, 2020, Proceedings 1*, páginas 138–163. Springer, 2020. 42, 55

[89] Gaudreault, Jean Gabriel: *Streamndr: Stream novelty detection for river*, 2023. `https://github.com/jgaud/streamndr`, Accessed: 2024-07-25. 47

[90] Silva, Alessandra de Melo e, João José Costa Gondim, Robson de Oliveira Albuquerque e Luis Javier García Villalba: *A methodology to evaluate standards and platforms within cyber threat intelligence.* Future Internet, 12(6):108, 2020. 49

[91] Silva, Rogerio Machado da, João José Costa Gondim e Robson de Oliveira Albuquerque: *Methodology to improve the quality of cyber threat intelligence production through open source platforms.* Em *International Conference on Computer Science, Electronics and Industrial Engineering (CSEI)*, páginas 86–98. Springer, 2022. 49

[92] Dimitriadis, Athanasios, Angelos Papoutsis, Dimitrios Kavalieros, Theodora Tsikrika, Stefanos Vrochidis e Ioannis Kompatsiaris: *Evacti: evaluating the actionability of cyber threat intelligence.* International Journal of Information Security, 24(3):123, 2025. 49

[93] Gaudreault, Jean Gabriel e Paula Branco: *Prioritizing the essential: A robust evaluation framework for novelty detection.* Machine Learning, 114(6):143, 2025. 50