Detecção de Ataques na Borda da Rede com Embeddings de Séries Temporais

Gabriel Violante[1], **Felipe Melo**[1], **Fernando Nakayama**[1], **Michele Nogueira**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Caixa Postal 567 – 31270-901 – Belo Horizonte – MG – Brasil

(gabriel.violante, felipemelo, fernandonakayama, michele)@dcc.ufmg.br

Abstract. *Network intrusion detection systems at the network edge identify suspicious or malicious activities positioned on edge devices. They allow identifying and reacting to threats before they propagate. This type of detection system is valuable in network architectures such as cloud, IoT networks, and 5G/6G networks. The edge is the point of highest exposure and edge devices present limitations that must be considered given current defenses, which are computationally expensive. This paper proposes TinyFlow-EdgeNIDS, an attack detection system based on lightweight models trained with time-series embeddings. It explores the embedding representation to enable accurate detection with reduced computational cost on constrained edge devices. The results show that the system achieves F1-scores up to 0.954 on network traffic using quantized lightweight models as small as 1.45 KB, supporting deployment at the network edge. Also, experiments with perturbed network flows indicate stable performance under plausible traffic manipulation.*

Resumo. *Os sistemas de detecção de intrusão na borda da rede identificam atividades suspeitas ou maliciosas posicionados nos dispositivos de borda. Eles permitem identificar e reagir a ameaças antes que elas se propaguem. Esse tipo de sistema de detecção é valioso em arquiteturas de rede como a nuvem, redes IoT e redes 5G/6G. Nesses casos, a borda é o ponto com maior exposição e os dispositivos de borda apresentam limitações que devem ser consideradas diante das defesas atuais, caras computacionalmente. Este artigo propõe o TinyFlow-EdgeNIDS, um sistema de detecção de ataques baseado em modelos leves treinados com embeddings de séries temporais. Ele explora a representação em embeddings para viabilizar alta eficácia de detecção com baixo custo computacional em cenários de borda. Os resultados indicam F1-score de até 0,954 utilizando modelos quantizados de apenas 1,45 KB, favorecendo a implantação em dispositivos restritos. Também, a avaliação com tráfego perturbado sugere estabilidade do detector sob alterações em atributos observáveis do tráfego.*

1. Introdução

A proliferação de dispositivos inteligentes consolidou a Internet das Coisas (IoT) como um vetor crítico de vulnerabilidades de segurança. As redes IoT são compostas por dispositivos com recursos computacionais limitados, como lâmpadas, câmeras e sensores

inteligentes, que muitas vezes atuam como porta de entrada para atacantes devido à sua simplicidade de *hardware* e *software*. Neste contexto, os Sistemas de Detecção de Intrusão de Rede (NIDS) desempenham um papel fundamental no monitoramento contínuo do tráfego para identificar atividades maliciosas. Os NIDS tradicionais operam em servidores centrais. Novas abordagens na borda propõem monitorar o tráfego, permitindo uma defesa distribuída, escalável e mais específica para o contexto IoT [Anuva 2025]. Entretanto, os modelos tradicionais de detecção possuem um custo computacional elevado [Anuva 2025]. Como alternativa, o paradigma *Tiny Machine Learning* (TinyML) viabiliza a execução de modelos de aprendizado de máquina em dispositivos com recursos computacionais limitados, como por exemplo dispositivos de borda IoT com poucos kilobytes de memória [Capogrosso et al. 2024, David et al. 2021], proporcionando benefícios como baixa latência, privacidade e eficiência energética [Im and Lee 2025]. Dessa forma, o TinyML apresenta-se como uma solução crítica para viabilizar a execução de NIDS diretamente na borda, contornando as limitações de recursos destes dispositivos.

Apesar da eficiência do TinyML, a detecção na borda enfrenta um desafio prático recorrente: produzir representações informativas do tráfego com baixo custo computacional, sem depender de pipelines extensos de engenharia de atributos. Em particular, as abordagens que partem de registros por fluxo frequentemente exigem múltiplas etapas de pré-processamento e seleção de atributos, o que aumenta a complexidade operacional e dificulta a implantação em pontos de observação na borda. Nesse contexto, representações baseadas em *embeddings* de séries temporais, construídas a partir de metadados de fluxos em janelas temporais, despontam como alternativa para reduzir a etapa de preparação dos dados e preservar padrões temporais relevantes para a detecção [Nakayama et al. 2025].

Os trabalhos recentes propõem NIDS leves para execução na borda. Estudos como [Anuva 2025] implementam detecção em *gateways* IoT, enquanto [Im and Lee 2025] focam em redes veiculares e [Sun and Zhao 2025] em ambientes *6G*. Embora tais propostas comprovem a viabilidade de inferência em *hardware* restrito, ainda é necessário avançar em sistemas que combinem a observação em pontos de agregação de tráfego na borda e representações compactas capazes de capturar a dinâmica temporal do tráfego com baixa sobrecarga. Em particular, a literatura recente sobre *embeddings* temporais em fluxos sugere que a qualidade da representação pode permitir classificadores mais simples, favorecendo implantação com baixa latência e poucos kilobytes de memória [Nakayama et al. 2025].

Este artigo aborda essa lacuna da ausência de sistemas eficientes para a borda e propõe o TinyFlow-EdgeNIDS, um NIDS voltado à detecção de ataques na borda e projetado para execução em pontos de acesso e *gateways* que concentram tráfego local. O sistema é projetado especificamente para operar com restrições de memória e processamento, compatível com dispositivos embarcados de baixo consumo. A arquitetura proposta abandona modelos complexos em favor de modelos supervisionados leves, como *Deep Neural Networks* compactas e Regressão Logística, treinados com *embeddings* de séries temporais multivariadas de fluxos de rede [Nakayama et al. 2025]. A hipótese central é que a combinação de modelos TinyML com uma representação temporal por *embeddings* melhora o compromisso entre acurácia de detecção e custo computacional na borda, ao capturar padrões temporais relevantes com dimensionalidade fixa e baixa sobrecarga. Como análise complementar, a avaliação também considera cenários de manipulação do

tráfego (evasão) para verificar a estabilidade do desempenho do detector sob perturbações.

A avaliação foca na análise prática do sistema em cenários de IoT realistas. O treinamento, teste e conversão dos modelos foram realizados em uma máquina com sistema operacional Ubuntu, processador AMD Ryzen 5 7600, 32GB de memória RAM DDR5 e GPU NVIDIA RTX 5060 Ti. A avaliação de inferência e o cálculo das métricas de eficiência priorizaram a conversão para *TensorFlow Lite for Microcontrollers* (TFLite) [David et al. 2021], visando a implantação em hardware restrito. Neste cenário, foi utilizado o Arduino Nano 33 BLE Sense Rev2 (Cortex-M4, 64 MHz, 256KB RAM) para simular a execução em um dispositivo IoT. Utilizando os datasets NF-UNSW-NB15-v3 [Moustafa and Slay 2015] e NF-ToN-IoT-v3 [Alsaedi et al. 2020], o estudo quantifica o *trade-off* entre desempenho de detecção e custo computacional, como tamanho do modelo (KB) e latência de inferência (ms). Os resultados indicam que a representação por *embeddings* associada a modelos TinyML sustenta alta eficácia de detecção com poucos kilobytes, favorecendo a implantação em APs na borda.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados, abordando NIDS na borda, TinyML e representações baseadas em *embeddings* para tráfego de rede. A Seção 3 descreve a metodologia proposta, detalhando a geração de *embeddings* e o fluxo de detecção na borda. A Seção 4 define a configuração experimental, incluindo os datasets e as métricas de desempenho e custo computacional, além de discutir os resultados obtidos com ênfase na viabilidade de implantação na borda. Por fim, a Seção 5 apresenta as conclusões e os trabalhos futuros.

2. Trabalhos Relacionados

A migração da inteligência para a borda da rede tem impulsionado o desenvolvimento de NIDS otimizados para dispositivos com recursos restritos [Anuva 2025, Im and Lee 2025, Sun and Zhao 2025]. Estudos recentes demonstram a viabilidade técnica de executar modelos de aprendizado de máquina em *hardware* de IoT. Em [Anuva 2025], o autor propôs o *framework* LIDIT, validando o uso de modelos baseados em LSTM em dispositivos como o Raspberry Pi Zero 2 W para detecção de anomalias em *Internet of Medical Things* (IoMT). Similarmente, em [Im and Lee 2025], os autores implementaram uma CNN leve em um microcontrolador (MCU) nRF52840 para proteger redes veiculares, com foco na redução do consumo energético, enquanto em [Sun and Zhao 2025], os autores introduziram técnicas de quantização para implantar modelos em ambientes 6G. Embora estes trabalhos avancem significativamente na eficiência computacional e na viabilidade de inferência na borda, eles ainda deixam em aberto desafios práticos ligados ao ponto de observação e execução na borda e à representação do tráfego utilizada pelo classificador. Em particular, a literatura frequentemente assume a execução em nós de borda genéricos ou dispositivos finais, sem discutir explicitamente o benefício de executar a detecção em pontos de agregação de tráfego, como APs e *gateways*. Além disso, muitas propostas mantêm forte dependência de *features* tradicionais por fluxo de rede, o que pode aumentar o custo e a complexidade do pipeline de dados para implantação contínua na borda.

Além da acurácia em cenários padrão, NIDS implantados na borda precisam manter desempenho sob variações do tráfego, mudanças de contexto e tentativas de evasão, pois tais fatores afetam a estabilidade operacional do detector. Em [He et al. 2023], os autores destacam que, diferentemente de domínios como visão computacional, o tráfego

de rede possui restrições semânticas rígidas, em que um ataque não pode ser perturbado arbitrariamente sem perder sua funcionalidade maliciosa. Em [Kumar et al. 2025], os autores formalizaram este conceito através da distinção entre *features* funcionais e não-funcionais, demonstrando que atacantes podem evadir a detecção modificando apenas atributos não essenciais do fluxo. No contexto de IoT, em [Khazane et al. 2024], os autores revisaram a literatura e indicaram que dispositivos de borda são alvos preferenciais para ataques de evasão devido à dificuldade de implementar defesas complexas localmente. Além disso, em [Zhang et al. 2025], os autores demonstraram a viabilidade de Perturbações Adversariais Universais em *Industrial Internet of Things* (IIoT), onde uma única perturbação calculada enganou múltiplos modelos. Esses resultados motivam que sistemas de detecção na borda avaliem, ao menos de forma complementar, a estabilidade do desempenho frente a manipulações e mudanças no tráfego, ainda que o objetivo central permaneça a eficácia de detecção com baixo custo computacional.

Para mitigar limitações de custo e complexidade do pipeline, diversas estratégias têm sido discutidas na literatura. Em [Alaliwat et al. 2025], os autores categorizam defesas e aprimoramentos em pré-processamento, detecção de anomalias e otimização do modelo. Uma abordagem comum é a utilização de arquiteturas profundas ou modelos auxiliares [Kumar et al. 2025]. Contudo, em ambientes TinyML, a execução de arquiteturas profundas ou múltiplos modelos simultâneos pode ser proibitiva em termos de memória e latência. Nesse contexto, uma direção particularmente relevante para a detecção na borda é reduzir a dependência de engenharia manual de atributos por meio de representações mais informativas do tráfego. Uma alternativa promissora é o uso de representações de dados mais ricas, como *embeddings*, que podem simplificar a fronteira de decisão para o classificador [Nakayama et al. 2025]. Em especial, *embeddings* de séries temporais multivariadas permitem mapear janelas de tráfego para vetores de dimensionalidade fixa, preservando padrões temporais úteis à detecção com baixa sobrecarga de pré-processamento.

O sistema TinyFlow-EdgeNIDS preenche essas lacunas ao priorizar a eficácia de detecção na borda por meio da combinação entre execução em pontos de agregação de tráfego e uma representação temporal compacta baseada em *embeddings*. Diferente de soluções dependentes de *Deep Learning* pesado [Sun and Zhao 2025] ou Autoencoders complexos, esta abordagem utiliza *embeddings* de séries temporais multivariadas [Nakayama et al. 2025] e transforma fluxos de rede em um espaço latente onde padrões associados a ataques tendem a se tornar mais separáveis com classificadores simples. Isso permite o uso de modelos supervisionados leves, como Regressão Logística e DNNs, mantendo alta eficácia de detecção com modelos de poucos kilobytes. Dessa forma, ele elimina a latência proibitiva de modelos profundos e fornece uma análise quantitativa dos *trade-offs* em um microcontrolador real, comprovando que a simplicidade do modelo, aliada à representação correta dos dados, é uma contribuição significativa para a segurança na borda. A avaliação considera também cenários de evasão/manipulação para observar a estabilidade do detector em condições adversas de operação.

3. Proposta

Esta seção descreve a proposta do sistema *TinyFlow-EdgeNIDS*, um sistema de detecção de ataques com atuação na borda da rede. O sistema foi proposto considerando sua execução em dispositivos de borda, posicionados preferencialmente em pontos de agregação de tráfego, como ponto de acesso (*Access Point*, AP), incluindo a possibilidade

de uso de hardware embarcado, com ênfase em viabilidade de implantação em dispositivos limitados e eficácia de detecção no tráfego observado. A proposta combina uma representação temporal compacta do tráfego via *embeddings* derivados de metadados de fluxos de rede e classificadores TinyML leves, buscando reduzir custo computacional e simplificar o pipeline de preparação dos dados na borda. As próximas subseções apresentam a visão geral e detalham cada um dos módulos do sistema.

3.1. Visão Geral

O *TinyFlow-EdgeNIDS* opera continuamente em um ponto de observação na borda da rede local, tipicamente em um AP que concentra o tráfego de múltiplos dispositivos finais. A proposta não pressupõe visibilidade global de toda a rede, em vez disso, assume-se que o dispositivo tem acesso apenas ao recorte local do tráfego associado ao seu ponto de conexão na rede, ou seja, aos pacotes observados a partir de sua(s) interface(s) de rede no segmento em que está inserido. Essa escolha endereça um ponto prático: executar a detecção no dispositivo final tende a limitar a visibilidade e o volume de tráfego observável, além de competir diretamente com a função primária do dispositivo. Ao contrário, APs agregam tráfego e, em geral, dispõem de mais recursos do que sensores e atuadores, tornando o cenário mais realista para implantação contínua de NIDS na borda. Esse cenário é compatível com o contexto IoT no qual a detecção local é desejável por razões de latência, autonomia e limitações de infraestrutura e também é coerente com o contexto de NIDS na borda sob restrições de recursos computacionais e de memória do dispositivo embarcado [Im and Lee 2025, Sun and Zhao 2025].

A Figura 1 ilustra o sistema composto por quatro módulos encadeados, executados de forma sequencial e cíclica. O módulo de extração monitora o tráfego disponível no ponto de observação do dispositivo e o agrega em registros de fluxo de rede (5-tupla) acompanhados de metadados numéricos (Seção 3.2). Em seguida, o módulo de montagem de *embeddings* organiza esses registros de fluxo em janelas temporais e produz, para cada janela, um vetor de dimensão fixa e_i que resume o comportamento recente do tráfego (Seção 3.3). O módulo de detecção recebe e_i e executa um classificador leve de aprendizado de máquina implantado no dispositivo, produzindo um *score* de ataque p_i (Seção 3.4). Por fim, o módulo de decisão transforma p_i em um rótulo binário e aciona uma política local de resposta, que é propositalmente mantida em aberto para adaptações ao cenário de aplicação (Seção 3.5).

A Figura 2 ilustra o posicionamento do sistema em um cenário concreto de aplicação alinhado à proposta. O sistema *TinyFlow-EdgeNIDS* é executado no *Access Point*, que atua como ponto de agregação e observação do tráfego da rede local. Nesse cenário, múltiplos dispositivos finais (*laptop*, *smartphone* e câmera inteligente) comunicam-se através do AP, gerando o tráfego de rede indicado pelas setas centrais. A câmera inteligente destacada na figura representa um dispositivo final potencialmente comprometido, isto é, um nó que pode ter sido atacado e passa a produzir padrões de tráfego associados a atividades maliciosas (por exemplo, variações abruptas de volume, mudanças de periodicidade, variações de interchegada e outras alterações comportamentais observáveis em metadados de fluxo). Ao posicionar o NIDS no AP, o sistema passa a observar o tráfego agregado que inclui comunicações benignas típicas do ambiente e comunicações oriundas de dispositivos comprometidos, sem depender de instrumentação dentro do dispositivo final. O bloco "Ação Predefinida" representa a saída operacional

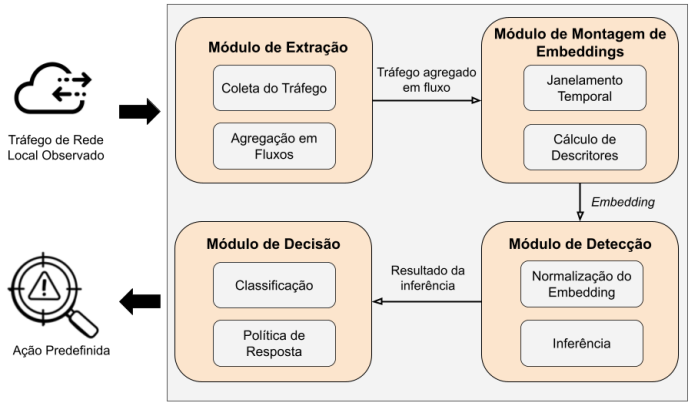

Figura 1. Visão geral do sistema *TinyFlow-EdgeNIDS*

do sistema, em que a cada janela temporal processada no ponto de acesso, o TinyFlow-EdgeNIDS determina (Seção 3.5) se o comportamento observado indica ataque ou não e, em seguida, executa uma ação pré-programada associada a esse resultado, como registrar o evento, sinalizar um alerta ou aplicar políticas locais no próprio AP. Essa ilustração reforça o posicionamento do sistema na borda da rede local, onde a execução do detector tende a ser mais defensável do que no dispositivo final.

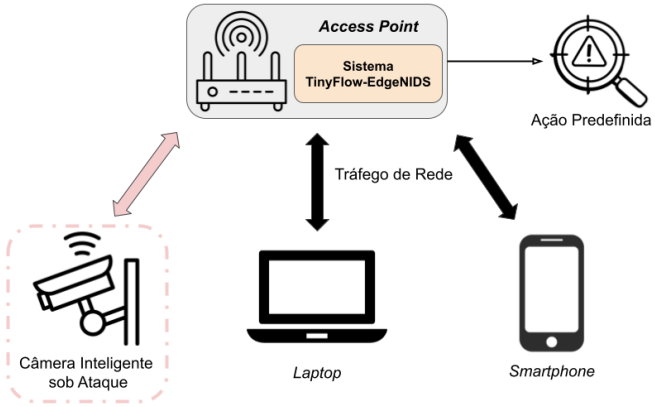

Figura 2. Posicionamento do sistema *TinyFlow-EdgeNIDS* em um *Access Point*

Em operação, pressupõe-se um classificador leve de aprendizado de máquina (Seções 3.4 e 3.5), voltado à detecção binária (*ataque* vs. *não-ataque*), que seja treinado externamente e, em seguida, implantado no AP para execução contínua em paralelo à função primária do dispositivo. Nesse contexto, o dispositivo mantém suas funções de conectividade enquanto executa o pipeline de detecção de forma cíclica sobre o tráfego que cruza o ponto de observação. O sistema processa o tráfego local disponível ao ponto de observação do dispositivo e executa ciclicamente os quatro módulos ilustrados na Figura 1. Assim, ele converte o tráfego observado na janela temporal em registros de fluxo de rede (5-tupla), transforma os fluxos em uma representação vetorial temporal, alimenta um classificador leve compatível com execução embarcada para rotular a janela como *ataque* ou *não-ataque* e toma uma decisão. Para a decisão, a política de resposta é propositalmente deixada em aberto para adaptações ao cenário de aplicação do sistema, como

por exemplo registrar evento, sinalizar alerta ou acionar mitigação.

3.2. Módulo de Extração

O módulo de extração pode operar tanto a partir da observação direta de pacotes no plano de dados quanto a partir de registros já agregados pelo próprio dispositivo (por exemplo, via exportação de fluxos). Em ambos os casos, a função do módulo é padronizar a entrada como registros por fluxo (5-tupla + metadados), que serão consumidos pelo módulo de *embeddings*. O módulo de extração converte o tráfego de rede local observável pelo dispositivo, ou seja, pacotes vistos a partir de sua(s) interface(s) de rede, em registros de fluxo associados a uma 5-tupla (IP de origem, IP de destino, porta de origem, porta de destino e protocolo). Neste trabalho, entende-se por **registro de fluxo** um resumo numérico de uma comunicação identificada pela 5-tupla em um intervalo, obtido a partir da agregação dos pacotes pertencentes ao fluxo (por exemplo, com *timeouts* e chaves de fluxo usuais em exportadores de fluxos). Como resultado da agregação, cada fluxo de rede é representado por atributos numéricos que sintetizam seu comportamento e que são utilizados diretamente pelo módulo seguinte para construir os *embeddings*. Os metadados utilizados neste trabalho são: instante de início do fluxo, duração, tamanho agregado e medidas de temporalidade como tempos médios entre chegadas (*inter-arrival time*). Por exemplo, para um fluxo TCP identificado pela 5-tupla $(A : port_a \rightarrow B : 80)$, o módulo de extração pode acumular, ao longo da janela, os bytes enviados/recebidos, estimar a duração e computar médias de interchegada em cada sentido, produzindo um registro numérico que representa aquele fluxo.

3.3. Módulo de Montagem de *Embeddings*

O módulo de montagem de *embeddings* transforma o conjunto variável de fluxos observados em uma janela temporal em um único vetor e_i, apropriado como entrada de modelos leves de aprendizado supervisionado. A entrada deste módulo pode ser vista como uma lista $F_i = \{f_1, \ldots, f_n\}$ de registros de fluxo de rede gerados na janela, em que cada f_k contém o tempo de início do fluxo e um conjunto pequeno de metadados numéricos (tamanho, duração e medidas de interchegada). Essa estratégia segue a motivação de *time series embeddings* aplicados a fluxos de rede, em que descritores estatísticos e temporais sintetizam o comportamento recente do tráfego em uma representação compacta [Nakayama et al. 2025].

A Figura 3 ilustra a composição do *embedding*. Na parte superior da imagem, o tráfego é inicialmente observado como uma sequência de pacotes e, em seguida, é agregado em fluxos; dentro de uma janela temporal (indicada como *JT*), esses fluxos são ordenados no tempo e dão origem a uma ou mais séries temporais $x_1, \ldots, x_n$. Como os fluxos ocorrem de forma irregular ao longo da janela, o módulo pode empregar subamostragens (*ST*) para obter amostras em instantes consistentes dentro de *JT*, o que viabiliza tratar a evolução dos metadados como sinais temporais no intervalo. No lado direito, cada série temporal é convertida em um conjunto de **descritores**, isto é, dados numéricos que resumem propriedades relevantes da série, agrupados em cinco tipos: estatística básica, regressão linear, análise harmônica, autocorrelação e distribuição. Por fim, os descritores de todas as famílias são concatenados, formando um vetor de dimensão fixa que representa a janela inteira.

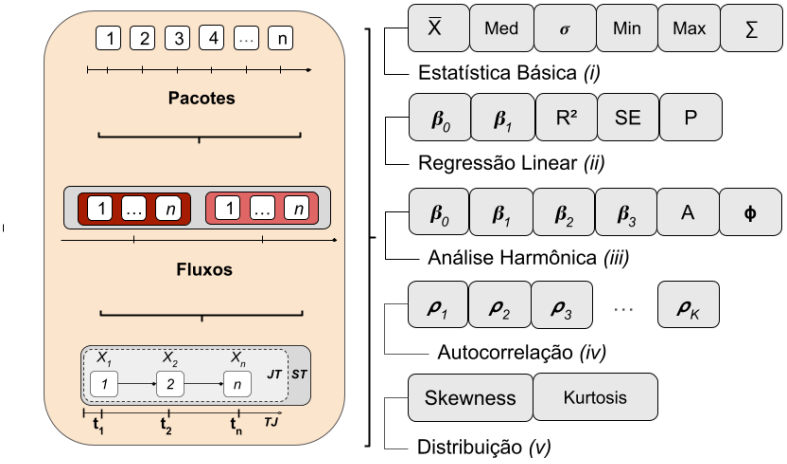

Figura 3. Exemplo conceitual de composição de *embedding* temporal.

3.3.1. Construção do *embedding*

A construção do *embedding* segue o princípio ilustrado na Figura 3. Primeiro, o tráfego observado pelo dispositivo é decomposto em fluxos de rede, em que um fluxo corresponde à agregação de pacotes que compartilham a mesma 5-tupla em um intervalo, resultando em um registro numérico por comunicação (Seção 3.2). Em seguida, dentro da janela temporal *JT*, o comportamento do tráfego é representado por séries temporais derivadas de metadados dos fluxos. Neste trabalho, os metadados considerados para compor as séries refletem duas classes de informação amplamente disponíveis em registros por fluxo: volume e temporalidade. Em particular, o sistema utiliza métricas como tamanho agregado do fluxo, duração (volume) e medidas de interchegada média (temporalidade), pois essas variáveis capturam padrões que diferenciam atividade benigna e maliciosa e são coerentes com a literatura de *embeddings* temporais em fluxos para detecção de ataques [Nakayama et al. 2025].

A etapa final converte cada série temporal em um conjunto de descritores, isto é, medidas que resumem aspectos complementares do sinal ao longo de *JT*. Conforme destacado na Figura 3, os descritores são organizados em cinco grupos. (i) A **estatística básica** captura magnitude e dispersão (por exemplo, média, mediana, desvio padrão, mínimo, máximo e soma), oferecendo um resumo direto do comportamento observado. (ii) A **regressão linear** descreve tendência global na janela (por exemplo, inclinação e intercepto), além de medidas de ajuste, indicando se o tráfego cresce, decresce ou permanece estável no intervalo. (iii) A **análise harmônica** resume componentes periódicas, úteis para capturar oscilações e padrões repetitivos. (iv) A **autocorrelação** quantifica o quanto valores atuais se relacionam com valores passados em diferentes defasagens, refletindo dependências temporais. (v) Por fim, a **distribuição** descreve a forma do sinal (por exemplo, assimetria e curtose), capturando comportamentos com caudas, picos e irregularidades que podem não aparecer apenas na média e variância.

O resultado é um vetor compacto que codifica, ao mesmo tempo, magnitude, variação, tendência e estrutura temporal do tráfego observado na janela *JT*, mantendo

dimensionalidade constante independentemente do número de pacotes/fluxos. O uso de *embeddings* é justificado neste trabalho por três razões. Primeiro, eles transformam um tráfego de tamanho variável em uma entrada fixa e padronizada, adequada para classificadores leves e execução embarcada [Nakayama et al. 2025, Im and Lee 2025, Sun and Zhao 2025]. Segundo, ao incorporar descritores temporais, a representação preserva a dinâmica do tráfego que tende a se perder em agregações puramente estáticas [Nakayama et al. 2025]. Terceiro, a síntese por descritores reduz a dependência de engenharia manual extensa ao oferecer uma representação compacta e consistente com a literatura de *embeddings* temporais para fluxos [Nakayama et al. 2025]. Como saída, o módulo de montagem produz, para cada janela, um vetor numérico de dimensão fixa e_i (com seus elementos correspondendo aos descritores concatenados), sendo e_i o artefato efetivamente consumido pelo módulo de detecção.

3.4. Módulo de Detecção

O módulo de detecção recebe, ao final de cada janela temporal Δ, isto é, após a construção do *embedding* da janela, o *embedding* e_i produzido pelo módulo anterior e executa um classificador supervisionado leve previamente treinado em ambiente externo e então implantado no dispositivo de borda. Em termos funcionais, este módulo computa um *score* $p_i \in [0, 1]$ associado à janela, interpretado como a probabilidade estimada de ocorrência de ataque na janela. Essa saída contínua decorre do uso de ativação Sigmoid na camada final do classificador, tanto para a regressão logística quanto para a DNN, em que, ao restringir a saída ao intervalo $[0, 1]$, obtém-se uma medida comparável entre janelas e facilmente limiarizável, além de compatível com o objetivo binário de detecção. A escolha por modelos leves permite que a eficácia de detecção dependa primariamente da qualidade da representação, mantendo a viabilidade de implantação em hardware restrito. Para manter compatibilidade com execução embarcada, o classificador é concebido para ser convertido e executado em dispositivos embarcados como microcontroladores. Assim, na Figura 1, este módulo é responsável por produzir o *score* p_i a partir de e_i, enquanto a transformação desse *score* em decisão binária e a definição de uma ação de resposta ficam a cargo do módulo de decisão.

3.5. Módulo de Decisão

O módulo de decisão consome o *score* p_i produzido pelo módulo de detecção e o converte em uma decisão operacional para o dispositivo de borda. Em sua forma mais simples, essa conversão é realizada pela aplicação de um limiar τ, produzindo o rótulo binário da janela: *ataque* quando $p_i \geq \tau$ e *não-ataque* caso contrário. O valor de τ é um parâmetro de implantação e pode ser ajustado conforme o custo relativo de falsos positivos e falsos negativos no cenário-alvo. Esse ajuste é particularmente relevante, tendo em vista que um limiar mais conservador pode reduzir bloqueios indevidos (falsos positivos) que afetariam múltiplos usuários, enquanto um limiar mais agressivo pode priorizar contenção rápida quando há evidência de comprometimento. Uma vez obtido o rótulo binário, este módulo aplica uma política local de resposta, deliberadamente mantida em aberto para acomodar diferentes contextos de uso do sistema. Os exemplos de resposta incluem registrar o evento, sinalizar um alerta para um coletor externo, acionar mitigação local de tráfego ou acionar um atuador.

4. Avaliação do Sistema

Esta seção apresenta a metodologia de avaliação e os resultados alcançados pelo *TinyFlow-EdgeNIDS*. O foco desta avaliação é verificar se modelos *TinyML* aliados à representação por *embeddings* temporais detectam ataques e permanecem compatíveis com restrições de implantação na borda, assumindo como cenário-alvo a execução do classificador em APs. Também é considerado um conjunto de dados de entrada *perturbado* para estressar o classificador sob alterações em atributos observáveis do tráfego, aproximando estratégias práticas de evasão discutidas na literatura como *padding*, *timing* e *mimicry* [He et al. 2023, Heydari and Nyarko 2025, Zhang et al. 2025].

Embora o sistema completo inclua extração de fluxos e montagem de *embeddings* no próprio AP, o protótipo avaliado neste artigo implementa integralmente (i) o pipeline de treinamento e avaliação, (ii) a conversão do modelo para TFLite com quantização pós-treinamento e (iii) a validação funcional de inferência do modelo quantizado em um dispositivo embarcado, com entrada de *embeddings* pré-computados e transmitidos por um *host* externo via USB/serial. Os experimentos consideram dois conjuntos de dados de tráfego rotulado: NF-ToN-IoT-v3 e NF-UNSW-NB15-v3. Em ambos os casos, adota-se a representação por *fluxos*, isto é, registros agregados por conexão, em vez de pacotes individuais, o que padroniza o pré-processamento e evita a etapa de conversão explícita de *pacotes* para *fluxos*. Para cada conjunto de dados, são avaliadas duas representações, a primeira é uma variante *baseline*, baseada em atributos de fluxo; a segunda é uma variante baseada em *embeddings* temporais. Na variante *baseline*, cada amostra corresponde a um fluxo descrito por atributos numéricos selecionados, incluindo o instante de início do fluxo, duração, estatísticas de *inter-arrival time* (IAT) e volume agregado do fluxo, além do rótulo binário. Na variante baseada em *embeddings*, os fluxos são agrupados pelo instante de início em janelas temporais de tamanho $\Delta = 3000$ ms e, para cada janela, produz-se um vetor compacto de 52 dimensões que resume o comportamento agregado dos fluxos observados no intervalo. Assim, a amostra no conjunto de *embeddings* representa um resumo temporal por janela da dinâmica do tráfego (ver Seção 3.3). O rótulo da janela é derivado dos rótulos dos fluxos contidos nela, de modo que a janela é marcada como ataque quando houver pelo menos um fluxo malicioso no intervalo.

4.1. Ambiente Experimental e Formas de Avaliação

O ambiente experimental considera um *host* para treinamento, conversão e testes de desempenho dos modelos e um dispositivo embarcado para validação de implantação do modelo quantizado. No *host*, os experimentos foram executados em uma máquina com Sistema Operacional Ubuntu, 32 GB de RAM e CPU AMD Ryzen 5 7600, com uma GPU RTX 5060 Ti. Contudo, para reduzir variações e aproximar o cenário de computação restrita, os experimentos foram forçados a executar em CPU. O dispositivo embarcado utilizado foi um Arduino Nano 33 BLE Sense Rev2, empregado para executar o modelo quantizado e validar a viabilidade de inferência em ambiente embarcado; na aplicação-alvo, o mesmo artefato TFLite é destinado à execução no AP.

O plano experimental avalia duas famílias de classificadores leves, Regressão Logística (*LogReg*) e uma *Deep Neural Network* (*DNN*) compacta, nas duas representações (*baseline* e *embeddings*). Para lidar com o desbalanceamento típico em dados de intrusão, foram considerados dois regimes de treino: *VULN*, treino padrão, no

qual a função de perda pondera igualmente as classes; e *VULN_CW*, no qual se aplica *class weighting* (*CW*) para penalizar mais fortemente erros na classe minoritária (tipicamente a classe *ataque*), reduzindo o viés do modelo em direção à classe majoritária e buscando melhorar a revocação sem degradar excessivamente a precisão. Após o treinamento, cada modelo foi convertido para TFLite com quantização pós-treinamento, e avaliado na execução padrão e na execução quantizada. Por fim, foi realizada uma validação no dispositivo embarcado com entrada de *embeddings* pré-computados enviados via USB/serial.

A avaliação reporta F1-score como métrica principal, por capturar o *trade-off* entre precisão e revocação no cenário binário ataque/não-ataque. Para tornar a análise comparável e reduzir a dependência de um único *split*, os resultados são apresentados separando-se dois conjuntos de teste, denominados **A** e **B**. O conjunto **A** corresponde a um subconjunto menor, utilizado como verificação de consistência e para acelerar iterações durante a avaliação; o conjunto **B** corresponde ao teste principal, contendo uma fração significativamente maior das amostras e, portanto, mais representativo do cenário operacional em escala (maior diversidade de fluxos/janelas e maior variabilidade de padrões). Em ambos, o desempenho é medido sob duas condições de entrada: (*i*) **tráfego limpo** (*clean*), isto é, amostras originais do conjunto de dados; e (*ii*) **tráfego perturbado** (*adv*), no qual amostras rotuladas como ataque são modificadas por perturbações controladas em atributos observáveis (por exemplo, variáveis temporais e de volume), preservando o rótulo original e aproximando estratégias práticas de evasão baseadas em camuflagem estatística no nível de fluxo. Assim, a condição *adv* não representa um novo tipo de ataque, mas um estresse do classificador sob desvio de distribuição induzido por alterações plausíveis nos atributos de fluxo. Como o conjunto **B** é o mais representativo para implantação em AP, a discussão prioriza seus resultados, mantendo o conjunto **A** como referência.

4.2. Resultados

A Tabela 1 apresenta o desempenho, em F1-score, na execução do modelo quantizado nos quatro cenários de teste (A/B, limpo/perturbado). Optou-se por apresentar resultados em TFLite por ser o formato de implantação no dispositivo de borda. Observou-se que a quantização pós-treinamento preservou a qualidade preditiva, com diferenças desprezíveis em relação à execução padrão, indicando que a redução de precisão numérica não degradou de forma relevante a detecção.

Tabela 1. Desempenho no modelo quantizado em quatro cenários de teste.

Dataset	Representação	Modelo	Regime	A→Limpo	A→Pert.	B→Limpo	B→Pert.
NF-ToN-IoT-v3	Baseline	DNN	VULN_CW	0,8756	0,4855	0,8708	0,4939
NF-ToN-IoT-v3	Baseline	LogReg	VULN_CW	0,4930	0,6288	0,4974	0,6392
NF-ToN-IoT-v3	Embeddings	DNN	VULN	0,9599	0,9123	0,9536	0,9218
NF-ToN-IoT-v3	Embeddings	LogReg	VULN_CW	0,9056	0,8951	0,9032	0,9108
NF-UNSW-NB15-v3	Baseline	DNN	VULN	0,7416	0,5789	0,7480	0,4362
NF-UNSW-NB15-v3	Baseline	LogReg	VULN_CW	0,4169	0,0000	0,4496	0,0141
NF-UNSW-NB15-v3	Embeddings	DNN	VULN	0,8874	0,8298	0,8888	0,7449
NF-UNSW-NB15-v3	Embeddings	LogReg	VULN_CW	0,8514	0,8213	0,8511	0,8190

Os resultados na Tabela 1 reforçam o argumento central do artigo: a representação por *embeddings* temporais é benéfica para detecção na borda. Em ambos os conjuntos de dados, a DNN com *embeddings* apresenta o melhor desempenho em tráfego limpo no conjunto B (NF-ToN-IoT-v3: F1=0,9536; NF-UNSW-NB15-v3: F1=0,8888), superando a contraparte *baseline* (NF-ToN-IoT-v3: F1=0,8708; NF-UNSW-NB15-v3: F1=0,7480).

Além disso, observa-se que os *embeddings* tornam a *LogReg* competitiva sob restrições: em NF-UNSW-NB15-v3, a *LogReg* com *embeddings* mantém F1 acima de 0,85 no teste B limpo, enquanto a *LogReg* na variante *baseline* apresenta desempenho inferior e instável.

No tráfego *perturbado* (adversarial), os *embeddings* também exibem maior estabilidade sob desvio de distribuição. Em NF-ToN-IoT-v3, a DNN em *baseline* sofre queda acentuada no conjunto B (de F1=0,8708 para 0,4939), enquanto a DNN com *embeddings* mantém F1=0,9218. Em NF-UNSW-NB15-v3, a DNN em *baseline* reduz para F1=0,4362 em B, ao passo que com *embeddings* alcança F1=0,7449. Esse comportamento é coerente com a intuição de que a decisão baseada em um resumo temporal agregado reduz a sensibilidade a perturbações localizadas em atributos de fluxos individuais, ainda que tal estabilidade não constitua, por si só, prova de robustez adversarial geral.

Para avaliar generalização entre domínios (um ponto crítico em cenários reais de APs, onde a distribuição de tráfego varia por ambiente), a Tabela 2 sumariza o desempenho *cross-dataset*, em que o modelo é treinado em um conjunto de dados e testado no outro, reportando o melhor modelo dentro de cada representação no cenário B→Limpo. A representação *baseline* degrada fortemente sob mudança de domínio, enquanto *embeddings* preservam parte relevante do poder de detecção, sustentando a hipótese de que a representação temporal agrega invariâncias úteis para implantação na borda.

Tabela 2. Generalização *cross-dataset* no cenário B→Limpo.

Treino→Teste	Baseline	Embeddings
NF-UNSW-NB15-v3→NF-ToN-IoT-v3	0,3213	0,8118
NF-ToN-IoT-v3→NF-UNSW-NB15-v3	0,1304	0,7131

Tabela 3. Custo computacional e tamanho do modelo quantizado.

Dataset	Representação	Modelo	Treino VULN (s)	Treino VULN_CW (s)	TFLite (KB)
NF-ToN-IoT-v3	Baseline	DNN	510,50	654,26	5,44
NF-ToN-IoT-v3	Baseline	LogReg	158,42	174,46	1,30
NF-ToN-IoT-v3	Embeddings	DNN	9,72	13,21	6,35
NF-ToN-IoT-v3	Embeddings	LogReg	20,07	19,81	1,46
NF-UNSW-NB15-v3	Baseline	DNN	592,85	186,64	4,94
NF-UNSW-NB15-v3	Baseline	LogReg	144,62	105,75	1,29
NF-UNSW-NB15-v3	Embeddings	DNN	9,47	6,38	6,34
NF-UNSW-NB15-v3	Embeddings	LogReg	8,13	8,21	1,45

A Tabela 3 sumariza custo computacional e tamanho dos modelos. Em armazenamento, os modelos TFLite resultam em poucos kilobytes, o que favorece implantação em dispositivos de borda. Em treinamento, a variante com *embeddings* reduz drasticamente o tempo, pois opera sobre janelas agregadas em vez de fluxos individuais: por exemplo, em NF-ToN-IoT-v3 (*DNN*), o treino padrão reduz de 510,50 s (*baseline*) para 9,72 s (*embeddings*); e, em NF-UNSW-NB15-v3, de 592,85 s para 9,47 s. Esse resultado é particularmente relevante no cenário de AP, pois facilita retreino e recalibração com baixo custo quando há mudança de domínio e atualização de políticas locais.

4.3. Discussão

Os resultados sustentam três conclusões principais sobre o *TinyFlow-EdgeNIDS*. Primeiro, a representação por *embeddings* temporais é consistentemente superior à variante

baseline para detecção em tráfego limpo, inclusive com modelos compactos: no cenário B→Limpo, a DNN com *embeddings* lidera em ambos os conjuntos de dados e a *LogReg* com *embeddings* permanece competitiva. Isso é relevante para implantação em APs, pois amplia o espaço de escolhas entre modelos sem perda severa de desempenho, preservando a possibilidade de decisões por janela com custo reduzido. Segundo, a avaliação *cross-dataset* evidencia que a principal fragilidade prática não é apenas o desempenho nominal, mas a generalização sob mudança de domínio. A variante *baseline* degrada fortemente quando testada em outro conjunto (Tabela 2), enquanto a variante com *embeddings* preserva desempenho substancial. Esse efeito é alinhado ao objetivo do trabalho: em ambientes reais, um AP pode observar padrões heterogêneos de tráfego ao longo do tempo e entre implantações; portanto, representações que capturam invariâncias temporais tendem a favorecer operação mais estável. Terceiro, do ponto de vista de viabilidade, os modelos quantizados em TFLite são pequenos e o custo de treinamento com *embeddings* é uma ordem de grandeza menor (Tabela 3). Para o caso de uso em AP, isso sugere um caminho pragmático em que é possível operar com inferência local por janela e manter capacidade de atualização do modelo com baixo custo computacional.

5. Conclusão

Este trabalho apresentou o TinyFlow-EdgeNIDS, um sistema de detecção de ataques projetado para operar na borda da rede local, com foco em implantação em pontos de agregação de tráfego como *Access Points*. A proposta combina (i) uma representação temporal compacta do tráfego, baseada em embeddings derivados de metadados de fluxos agregados em janelas, e (ii) classificadores supervisionados leves (DNN compacta e Regressão Logística), visando reduzir a dependência de engenharia extensa de atributos e manter viabilidade sob restrições típicas de dispositivos de borda. A avaliação em dois datasets representativos da literatura, considerando modelos quantizados em TFLite e validação funcional em hardware embarcado, sustenta que a representação por embeddings melhora o compromisso entre eficácia e custo para detecção na borda. Em particular, os embeddings elevaram o desempenho no cenário de teste principal em tráfego limpo e mantiveram maior estabilidade sob tráfego perturbado, enquanto variantes *baseline* apresentaram degradações acentuadas em condição adversa.

Do ponto de vista de viabilidade, os modelos quantizados resultaram em poucos kilobytes, e o treinamento com embeddings reduziu substancialmente o custo computacional, favorecendo ciclos de retreino e recalibração. Como limitações, a validação embarcada utilizou embeddings pré-computados enviados ao dispositivo, não caracterizando ainda o pipeline fim-a-fim no AP com extração de fluxos e montagem de embeddings em tempo real, nem a latência ponta-a-ponta no alvo final. Como trabalhos futuros, torna-se prioritário estressar a avaliação do sistema proposto, caracterizar custo e latência em operação contínua e ampliar a avaliação incluindo modelos de ameaça mais abrangentes para ataques de manipulação/evasão.

Referências

Alaliwat, F., Alqahtani, L., Alzahrani, M., Alamoudi, N., Hakami, S., and Alharby, A. (2025). Detection of adversarial evasion attack on AI model running on IoT devices. In *12th International Conference on Information Technology (ICIT)*, páginas 47–54. IEEE.

Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A. N., and Anwar, A. (2020). TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150.

Anuva, S. T. (2025). LIDIT: Low-latency intrusion detection in IoMT devices using TinyML. Master's thesis, Queen's University, Kingston, Ontario, Canada.

Capogrosso, L., Cunico, F., Cheng, D. S., Fummi, F., and Cristani, M. (2024). A machine learning-oriented survey on Tiny Machine Learning. *IEEE Access*, 12:23406–23426.

David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T., Warden, P., and Rhodes, R. (2021). TensorFlow Lite Micro: Embedded machine learning for tinyml systems. In Smola, A., Dimakis, A., and Stoica, I., editors, *Proceedings of Machine Learning and Systems*, volume 3, páginas 800–811.

He, K., Kim, D. D., and Asghar, M. R. (2023). Adversarial machine learning for network intrusion detection systems: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 25(1):538–566.

Heydari, V. and Nyarko, K. (2025). Enhancing adversarial robustness in network intrusion detection: A novel adversarially trained neural network approach. *Electronics*, 14(16):3249.

Im, H. and Lee, S. (2025). TinyML-based intrusion detection system for in-vehicle network using convolutional neural network on embedded devices. *IEEE Embedded Systems Letters*, 17(2):67–70.

Khazane, H., Ridouani, M., Salahdine, F., and Kaabouch, N. (2024). A holistic review of machine learning adversarial attacks in IoT networks. *Future Internet*, 16(1):32.

Kumar, V., Kumar, K., Singh, M., and Kumar, N. (2025). NIDS-DA: Detecting functionally preserved adversarial examples for network intrusion detection system using deep autoencoders. *Expert Systems with Applications*, 270:126513.

Moustafa, N. and Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference (MilCIS)*, páginas 1–6. IEEE.

Nakayama, F., Melo, F., Brezolin, U., and Nogueira, M. (2025). A time series embedding method for attack detection in network flows. In *2025 IEEE Latin-American Conference on Communications (LATINCOM)*, páginas 1–6. IEEE.

Sun, B. and Zhao, Y. (2025). TinyNIDS: CNN-based network intrusion detection system on TinyML models in 6g environments. *Internet Technology Letters*, 8(6):e629.

Zhang, S., Xu, Y., and Xie, X. (2025). Universal adversarial perturbations against machine-learning-based intrusion detection systems in Industrial Internet of Things. *IEEE Internet of Things Journal*, 12(2):1867–1889.