Da Teoria à Implantação: Um Framework Metodológico para *In-Network Machine Learning* em Redes Programáveis

Icaro M. da Silva, Caio Luiz L. T. Silva, Thiago Gouveia e Leandro C. de Almeida

[1]Unidade Acadêmica de Informática – Instituto Federal da Paraíba (IFPB)
João Pessoa – PB – Brazil

{caio.luiz, icaro.silva}@academico.ifpb.edu.br,

{thiago.gouveia,leandro.almeida}@ifpb.edu.br

Abstract. *Machine learning has consolidated itself as a central component in computer networks, and the evolution of programmable devices has enabled the In-Network ML paradigm, allowing inferences at hardware speed. However, the practical adoption and materialization of these solutions face significant barriers due to architectural constraints and a lack of consolidated methodologies, often resulting in approaches restricted to specific algorithms or hardware. To bridge this gap, this paper proposes a structured methodology and a generalist framework to guide the materialization of In-Network ML pipelines. The roadmap connects everything from data preparation and offline training to the translation of the inference logic into Match-Action Tables compatible with the P4 language. The reproducibility of the proposal is demonstrated through a practical traffic classification use case using a Decision Tree model in a virtualized environment with the BMv2 switch. The results show the feasibility of the deployment, with low resource overhead, such as an average CPU usage increase of only 7.3%.*

Resumo. *O aprendizado de máquina tem se consolidado como componente central em redes de computadores, e a evolução dos dispositivos programáveis viabilizou o paradigma de In-Network ML, permitindo inferências na velocidade do hardware. No entanto, a adoção prática e a materialização dessas soluções enfrentam barreiras significativas devido a restrições arquiteturais e à carência de metodologias consolidadas, resultando frequentemente em abordagens restritas a algoritmos ou hardwares específicos. Para preencher essa lacuna, este artigo propõe uma metodologia estruturada e um framework generalista para orientar a materialização de pipelines de In-Network ML. O roteiro conecta desde a preparação de dados e treinamento offline até a tradução da lógica de inferência para Tabelas compatíveis com a linguagem P4. A reprodutibilidade da proposta é demonstrada por meio de um caso de uso de classificação de tráfego utilizando uma Árvore de Decisão em um ambiente virtualizado com o switch BMv2. Os resultados evidenciam a viabilidade da implantação, com baixo overhead de recursos, como um aumento médio de apenas 7,3% no uso de CPU.*

1. Introdução

O avanço da capacidade computacional e a crescente disponibilidade de dados têm consolidado o aprendizado de máquina (ML, do inglês *Machine Learning*) como componente central em diferentes domínios de aplicação [Wang et al. 2020]. Em redes de computadores, esse movimento é particularmente relevante, pois operadores e pesquisadores demandam mecanismos cada vez mais responsivos para classificação de tráfego, detecção de anomalias, automação de políticas e otimização de desempenho em ambientes dinâmicos e de larga escala. Adicionalmente, a complexidade das aplicações distribuídas e a crescente variabilidade do comportamento do tráfego tornam abordagens baseadas em regras estáticas insuficientes, reforçando a necessidade de soluções orientadas a dados que sejam capazes de aprender e evoluir continuamente em tempo de operação.

Em paralelo, a evolução dos dispositivos de rede programáveis ampliou o espaço de projeto para além do paradigma tradicional de encaminhamento estático. Switches programáveis e placas de rede inteligentes (SmartNICs), por exemplo, viabilizam a inserção de lógica especializada no plano de dados, permitindo explorar recursos antes subutilizados e aproximando processamento e tomada de decisão do próprio caminho percorrido pelos pacotes. Isto reduz drasticamente a latência frente às abordagens tradicionais. Enquanto o envio de pacotes ao controlador gera atrasos na ordem de milissegundos, processar a inferência no plano de dados permite decisões na velocidade do *hardware* (nanosegundos), viabilizando redes de alta vazão sem sobrecarregar o plano de controle.

Nesse contexto, o termo *In-Network Computing* descreve a execução de cargas computacionais dentro da infraestrutura de rede, reduzindo a dependência exclusiva de servidores de propósito geral [Kianpisheh and Taleb 2023]. Essa abordagem pode diminuir latência fim a fim, reduzir movimentação de dados e melhorar eficiência energética, características importantes para aplicações com requisitos estritos de tempo de resposta e alto volume de tráfego. Uma vertente desse paradigma é o *In-Network ML*, no qual modelos de ML são parcial ou totalmente executados em dispositivos de rede programáveis [Zheng et al. 2024a]. Em vez de apenas transportar dados para processamento externo, a rede passa a atuar como agente computacional ativo, capaz de extrair atributos, aplicar decisões inferenciais e, quando necessário, acionar respostas em tempo real com o tráfego observado.

Trabalhos recentes evidenciam duas direções complementares de uso. A primeira concentra-se na aceleração do treinamento de ML baseado em host por meio de operações distribuídas assistidas pela rede [Sapio et al. 2021, Gebara et al. 2021, Lao et al. 2021]. A segunda foca na inferência diretamente no plano de dados, em regime *line-rate*, para tarefas como classificação de fluxos e detecção em tempo real [Xiong and Zilberman 2019, Sada et al. 2025, Zhang et al. 2026]. Apesar desse progresso, a adoção prática ainda encontra barreiras significativas. A materialização de soluções envolve conhecimento multidisciplinar em redes programáveis, compiladores, representação de modelos e restrições arquiteturais de *targets* como Tofino[1], BMv2[2], P4-DPDK[3] e diferentes classes de SmartNICs. Na prática, essa curva de aprendizado eleva o custo de prototipação e dificulta a

[1]Intel Tofino: `https://www.intel.com/content/www/us/en/products/details/network-io/programmable-ethernet-switch/tofino-series.html`.

[2]Behavioral Model v2 (BMv2): `https://github.com/p4lang/behavioral-model`.

[3]P4-DPDK: `https://github.com/p4lang/p4c/tree/main/backends/dpdk`.

transição de propostas conceituais para sistemas reprodutíveis em ambientes experimentais e de produção.

Adicionalmente, observa-se carência de metodologias consolidadas que conectem, de forma sistemática, todas as etapas do ciclo de vida desses sistemas: aquisição e preparação de dados (tipicamente fora do plano de dados), treinamento e seleção de modelos, tradução para artefatos compatíveis com P4 e implantação para inferência. Como consequência, a ausência desse encadeamento metodológico tende a gerar avaliações pouco comparáveis, comprometer reprodutibilidade e dificultar análises justas entre implementações.

Diante desse cenário, este artigo propõe uma metodologia estruturada, acompanhada de um *framework* de apoio, para orientar a materialização de pipelines de *In-Network ML* em casos de uso genéricos. A proposta se destaca por organizar o processo do início ao fim, conectando desde a aquisição e preparação do dado bruto até a tradução do modelo e à inferência no dispositivo programável. Diferentemente de abordagens restritas a algoritmos ou *hardwares* específicos, a principal contribuição deste trabalho é fornecer um roteiro generalista e reprodutível[4], capaz de acomodar múltiplos modelos de aprendizado e diferentes *targets* P4. Complementarmente, o estudo disponibiliza uma infraestrutura de experimentação automatizada, validada de ponta a ponta por meio de um caso de uso de classificação de tráfego.

O restante deste trabalho está organizado como segue: a Seção 2 revisa os conceitos fundamentais para a implementação de *In-Network Machine Learning*, em seguida, a Seção 3, que posiciona este artigo frente à literatura. Na Seção 4, detalha-se o roteiro ponta a ponta para a materialização de modelos de ML no plano de dados programável. Posteriormente, para demonstrar a reprodutibilidade da metodologia, a Seção 5 descreve um caso de uso experimental. Por fim, o artigo é concluído na Seção 6, que apresenta as considerações finais e propostas para trabalhos futuros.

2. Conceitos Fundamentais

Esta seção oferece uma breve revisão dos fundamentos necessários para uma boa compreensão deste artigo. Para tal, discute-se inicialmente os princípios do plano de dados programável e da linguagem P4. Na sequência, é detalhado o paradigma de *In-Network Machine Learning*, contextualizando como modelos de aprendizado de máquina podem ser integrados e executados diretamente nos dispositivos de rede.

2.1. Plano de dados programável

Com o surgimento das redes definidas por software (SDN) [Casado et al. 2007, McKeown et al. 2008], vieram os primeiros esforços na direção da programabilidade da rede [Hauser et al. 2021]. A separação dos planos de dados e controle permitiu uma maior flexibilidade e abriu novos horizontes de pesquisa. Em operações rotineiras, o plano de controle atua na tomada de decisões globais, como a descoberta de topologia e o cálculo de rotas, operando em uma escala de tempo mais alta. O plano de dados, por sua vez, é estritamente responsável pela execução dessas regras, realizando o processamento e o encaminhamento de cada pacote na velocidade do *hardware*, tipicamente na escala de nanosegundos.

[4] `https://github.com/ifpb/In-NetRoadmap`

Nesta direção, a linguagem P4 [Bosshart et al. 2014] surgiu em 2014 como uma alternativa e evolução natural do paradigma de programabilidade de redes, trazendo a possibilidade de se programar o plano de dados de ativos de rede ou placas de rede inteligentes (*SmartNics*). A linguagem P4 possui duas versões oficiais, $P4_{14}$ e $P4_{16}$, sendo a última mais utilizada pela comunidade acadêmica.

Partindo da programabilidade do plano de dados, é possível implementar algoritmos/inteligência durante o processo de manipulação de pacotes no nível mais próximo do *hardware*, sem precisar recorrer ao plano de controle. A linguagem P4 se enquadra neste contexto, permitindo que programadores definam alguma lógica no *pipeline* do processamento de pacotes diretamente no plano de dados. O principal atrativo para se programar no plano de dados é o desempenho alcançado pelos algoritmos. De acordo com [Kurose and Ross 2016], o tempo de resposta do plano de controle é na escala de segundos ou milissegundos, enquanto que no plano de dados é de nanosegundos.

O código compilado em linguagem P4 pode executar em diversas arquiteturas, sendo que cada arquitetura possui diferentes estágios (*match-action pipelines*) e características para processar os pacotes. De um modo geral, as arquiteturas são orientadas por um modelo de abstração que mapeia as instruções e as especificidades de cada *hardware* (*target*). Inicialmente, definiu-se a arquitetura PISA (*Protocol-Independent Switch Architecture*). Para cada categoria de equipamento, comumente chamado no ecosistema P4 de *target*, existe uma arquitetura específica. Para o caso da NetFPGA, a arquitetura utilizada é a *Simple Sume Architecture*. No caso do Tofino, a arquitetura utilizada é a *Tofino Native Architecture*. Já para o BMv2, a arquitetura utilizada é a *V1model* [Hauser et al. 2021]. Conforme ilustrado na Figura 1, a arquitetura PISA é composta por três componentes principais: *parser* programável, *match-action pipeline* programável e um *deparser* programável.

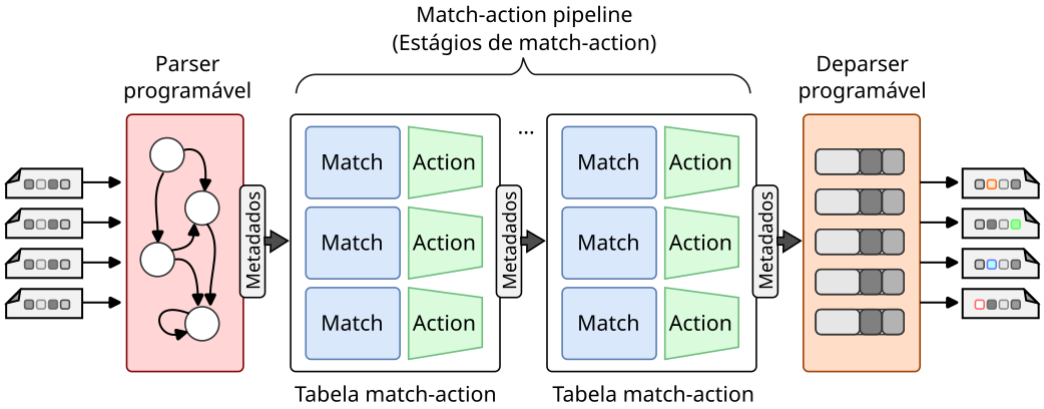

Figura 1. Modelo de abstração PISA. Adaptado de [Hauser et al. 2021]

O *parser* programável é uma máquina de estados finita que define a ordem da extração dos cabeçalhos do pacote que entra no dispositivo de rede, para que seus campos possam ser analisados nos próximos estágios (*match-action pipelines*). Uma vez que os cabeçalhos e seus respectivos campos sejam expostos pelo *parser* programável, eles podem ser tratados em múltiplos estágios de *match-action*. Neste caso, tabelas são definidas para realizar o tratamento adequado dos metadados, seguindo a lógica definida pelo programador. Por exemplo, para o roteamento através do IPv4, uma tabela pode ser defi-

nida para realizar uma correspondência (*match*) com o endereço de destino e tomar uma ação (*action*) para: *i*) decrementar o campo TTL; *ii*) ajustar os endereços físicos; e/ou *iii*) definir a interface de saída. Por fim, o *deparser* é o componente programável em que se declara como o pacote deverá ser remontado (serialização) para transmissão.

Dessa forma, a programabilidade do plano de dados estabelece a base técnica para deslocar parte da inteligência para dentro da própria rede, reduzindo a distância entre observação, decisão e ação sobre o tráfego. Esse movimento abre espaço para o paradigma de *In-Network Machine Learning*, no qual modelos de aprendizado de máquina passam a ser executados, de forma total ou parcial, diretamente em elementos de rede programáveis. Na seção seguinte, discutimos os fundamentos de aprendizado de máquina necessários para compreender como essa integração é materializada na prática.

2.2. *In-Network Machine Learning*

Modelos de aprendizado de máquina dependem diretamente de dados representativos para extrair padrões, treinar com robustez e sustentar boas decisões em fase de inferência. Em redes de computadores, essa condição é particularmente favorável, pois o tráfego observado ao longo do tempo constitui uma fonte contínua de informações sobre comportamento, desempenho e eventos anômalos [Boutaba et al. 2018].

Quando essa disponibilidade de dados se combina com a maturidade recente das técnicas de ML, surgem oportunidades concretas para ampliar a automação e a inteligência operacional da rede. Entre as aplicações mais relevantes estão a classificação de tráfego, a detecção de ataques, a identificação de degradação de qualidade de serviço e o suporte à tomada de decisão em tempo de execução. Historicamente, esse cenário foi implementado por meio de arquiteturas em que o tráfego é redirecionado ao controlador ou a servidores externos ao plano de dados para execução da inferência. Com a evolução dos dispositivos programáveis, esse fluxo passou a migrar para o paradigma de *In-Network ML*, no qual a inferência é aproximada do próprio caminho dos pacotes.

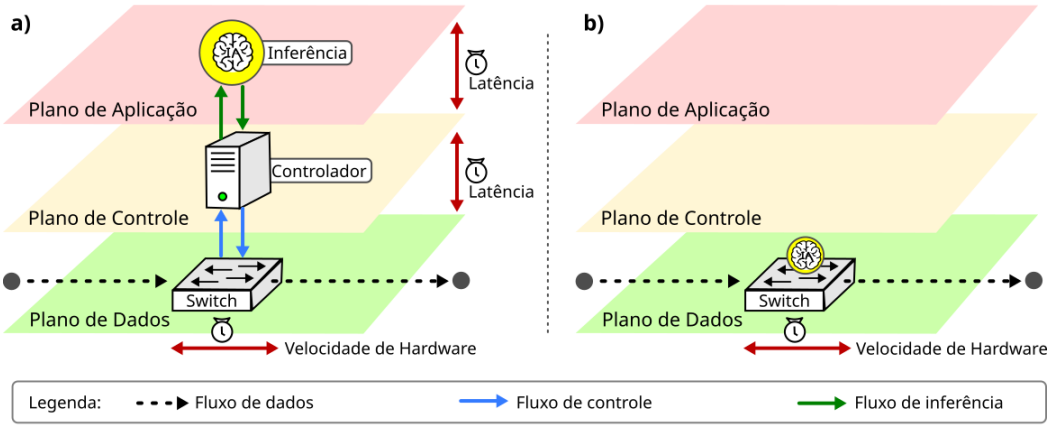

Figura 2. Diferença entre ML convencional e In-Network ML

A Figura 2 ilustra essa transição ao comparar os dois modos de operação. No paradigma convencional, Figura 2a, os pacotes precisam ser espelhados ou exportados para um servidor externo, no qual a inferência é executada, para depois retornar ao domínio da rede com a ação de controle; esse ciclo alonga o caminho de processamento, aumenta a

latência e amplia a dependência do plano de controle. Já no cenário de *In-Network ML*, Figura 2b, a extração de atributos e a inferência são incorporadas ao próprio pipeline de encaminhamento, permitindo classificar e reagir ao tráfego em tempo real, com menor sobrecarga de comunicação e maior capacidade de resposta a eventos dinâmicos.

Apesar dos ganhos com a diminuição da latência e melhoria na responsividade discutidos anteriormente, a materialização de modelos de ML em dispositivos originalmente projetados para encaminhamento de pacotes ainda impõe desafios técnicos relevantes. Em especial, limitações de memória, quantidade de estágios do pipeline, tipos de dados suportados e orçamento temporal por pacote exigem decisões de projeto criteriosas, além de adaptações no modelo e na engenharia de implementação para preservar viabilidade e desempenho [Xiong and Zilberman 2019].

De um modo geral, têm-se observado que pesquisadores se apoiam na abstração de *Match-Action Tables* (MATs), que oferece um mecanismo prático para mapear decisões inferenciais em operações compatíveis com o plano de dados. Embora esse mapeamento exija simplificações e ajustes no desenho do modelo, as MATs tornam viável a execução de inferência com baixa latência e alta vazão, aproximando o aprendizado de máquina das exigências de processamento em tempo real na infraestrutura de rede.

3. Trabalhos Relacionados

Ao analisar a literatura recente, percebe-se que o conhecimento encontra-se fragmentado entre vários trabalhos, que sugerem múltiplos casos de uso, modelos de ML, técnicas e soluções em inferência. Neste sentido, esta seção reúne uma análise individual e geral dos artigos, tentando posicionar o nosso estudo perante os trabalhos já existentes.

O conhecimento disperso e fragmentado também aparece na discussão do Planter [Zheng et al. 2024b], um *framework* voltado à automação da prototipação e do desenvolvimento de *In-Network Machine Learning*. A solução reúne três metodologias gerais de mapeamento, cobrindo múltiplos modelos e diferentes *targets* de inferência. Em linha com a proposta deste artigo, o Planter percorre o pipeline completo, do treinamento à conversão para P4 e à implantação em múltiplos *targets*, focando, contudo, em prover automaticamente o produto final, ao invés de enfatizar o processo.

Ainda no contexto de implementação concreta, o INQ-MLT [Zhang et al. 2024] constitui um conjunto de ferramentas focado em treinamento quantizado para viabilizar modelos mais complexos sob as restrições do P4. O INQ-MLT converte representações em ponto flutuante para inteiros de baixa precisão e estrutura um fluxo prático de implantação no plano de dados programável. Assim como o Planter, prioriza a entrega de ferramenta executável, com ênfase específica na etapa de quantização. Sua limitação é usar apenas o BMv2 como *target* P4.

Em paralelo, um framework para detecção de anomalias é proposto em [Nguyen et al. 2024], com trajetória clara entre treinamento *offline* e inferência no *switch*. O trabalho documenta de forma consistente as etapas de extração de dados via PCAP, treinamento do modelo, conversão para MATs e implementação da lógica do *switch* em P4. Essa abordagem reforça a importância de pipelines completos, embora permaneça centrada em um caso de uso específico.

De forma semelhante, o NetBeacon [Zhou et al. 2023] é um *framework* que busca aumentar a eficiência do desenho e da implantação de modelos com foco em features de nível de fluxo. O NetBeacon combina um componente de construção de modelos cientes do plano de dados com um componente de implantação otimizada, cobrindo etapas relevantes do pipeline de *In-Network ML*. Ainda assim, seu escopo permanece restrito a Decision Trees e Random Forests, além do suporte direcionado à arquitetura PISA, o que limita a generalização para outros modelos e *targets*.

Conforme evidenciado na Tabela 1, embora a literatura recente tenha avançado na proposição de pipelines completos para *In-Network ML*, nota-se uma carência de abordagens que sejam simultaneamente generalistas e independentes de *hardware*. Trabalhos anteriores tendem a fornecer ferramentas de implementação engessadas, limitando-se a modelos específicos (como modelos baseados em árvore de decisão) ou arquiteturas singulares (como a PISA). É exatamente nesta lacuna que a presente pesquisa se posiciona, propondo entregar um roteiro conceitual metodológico, capaz de orientar e democratizar a implantação de múltiplos modelos de aprendizado de máquina em uma diversidade de *targets* P4.

Tabela 1. Comparação das abordagens de implantação de In-Network ML, evidenciando as lacunas de generalização preenchidas por este trabalho.

Trabalho	Pipeline completo	Múltiplos modelos	Múltiplos targets P4	Roteiro metodológico
[Zheng et al. 2024b]	✓	✓	✓	×
[Zhang et al. 2024]	✓	✓	×	×
[Nguyen et al. 2024]	✓	×	×	×
[Zhou et al. 2023]	✓	×	×	×
Este artigo	✓	✓	✓	✓

4. Materializando modelos de ML no plano de dados programável

Esta seção apresenta o fluxo de materialização de modelos de ML no plano de dados, partindo da representação do modelo treinado até sua tradução para componentes executáveis no pipeline de um switch programável. A discussão é organizada em etapas para evidenciar como decisões de modelagem, estrutura de MATs e a relação sobre como os planos de controle e dados se combinam para viabilizar a inferência em tempo de *hardware*.

A Figura 3 apresenta uma visão geral do *framework* proposto neste estudo, organizado em três módulos distintos. Em termos gerais, o Módulo de Entrada (Figura 3a) é responsável pelo ciclo de treinamento: ele recebe um arquivo CSV, que representa o conjunto de dados de treino, e um arquivo de configuração com o modelo selecionado e seus hiperparâmetros; em seguida, executa o treinamento *offline* a partir de um notebook Python customizável gerado pelo módulo, produz resultados parciais contendo métricas e gráficos, como *loss*, acurácia e matrizes de confusão, e exporta o modelo em um formato intermediário.

Em seguida, o Módulo de Conversão (Figura 3b) consome esse modelo exportado e aplica recursos de adaptação, tais como a seleção e codificação de features,

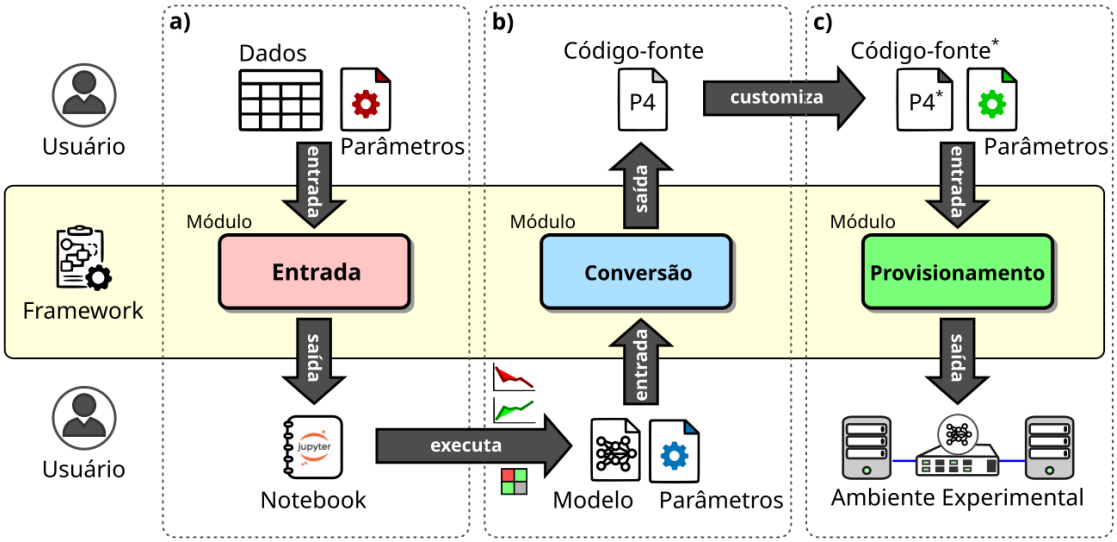

Figura 3. Visão geral do framework metodológico proposto

discretização/quantização de valores contínuos, transformação de limiares e regras de decisão em chaves e ações para mapear a inferência treinada em tabelas MAT e, então, gerar o código P4 pronto para execução no *target*. Neste ponto, há uma oportunidade para customização do código P4 por parte do usuário. Por fim, o Módulo de Provisionamento (Figura 3c) recebe o programa P4 e as configurações de ambiente para gerar automaticamente a infraestrutura de experimentação, fazendo uso extensivo de artefatos Ansible, Vagrant e contêineres, deixando o ambiente experimental preparado para execução e avaliação de tráfego.

Neste contexto, o *framework* proposto apresenta todas as etapas necessárias para tradução de um modelo de ML em uma representação compatível com as restrições de um switch P4 programável. Essa tradução varia conforme o algoritmo, mas segue uma lógica comum e generalista: identificar quais atributos do tráfego serão utilizados, como os parâmetros do modelo serão codificados e qual regra de decisão será aplicada no pipeline. Em árvores de decisão, por exemplo, a materialização costuma envolver features, limiares e caminhos de decisão; em modelos probabilísticos, envolve parâmetros de cálculo de pontuação/probabilidade; em métodos baseados em distância, envolve referência a centróides e funções de comparação. Apesar das diferenças matemáticas, todos os casos passam pelo mesmo princípio de engenharia: converter a função de inferência em operações determinísticas e executáveis no plano de dados.

De forma didática, a ideia central é decompor a inferência em etapas simples: (i) extração dos campos necessários do pacote, (ii) normalização/quantização quando necessário, (iii) avaliação de condições intermediárias e (iv) composição da decisão final. Essa decomposição permite distribuir a lógica em uma ou múltiplas MATs, conforme a complexidade do modelo e os limites do *hardware*. Assim, a proposta mantém caráter generalista, ou seja, não impõe um único algoritmo, uma única arquitetura ou um único caso de uso, mas oferece um roteiro reprodutível para transformar modelos treinados *offline* em inferência embarcada no plano de dados de switches programáveis.

Conforme descrito anteriormente de forma breve, as MATs são componentes utili-

zados para tomar decisões com base na correspondência de chaves. Cada MAT é definida por um conjunto de campos empregados no processo de correspondência, pelo tipo de correspondência aplicado a cada campo, que, em conjunto, formam a chave de busca, e por uma lista de possíveis ações que são executadas quando há correspondência. As entradas da tabela consistem em uma chave utilizada na correspondência, na ação a ser executada e os seus parâmetros.

Complementarmente, uma ação é uma estrutura sintaticamente semelhante a uma função da linguagem C, mas sem valor de retorno. Assim como em uma função, ela consiste em um bloco de código executado durante o processamento e que pode receber parâmetros quando é chamada. Na definição de uma ação, esses parâmetros podem ter direções que indicam se seus valores são de entrada, de saída ou de entrada e saída.

A definição dessas ações está diretamente conectada ao modo como as tabelas são populadas. O preenchimento das entradas pode ser realizado de forma estática no código ou de forma dinâmica pelo plano de controle. Quando definido no próprio código P4, esse preenchimento ocorre em tempo de compilação, de modo que as entradas permanecem fixas durante toda a execução e não podem ser alteradas sem recompilar e reinstalar o programa no dispositivo. Em contrapartida, quando o preenchimento é feito pelo plano de controle, torna-se possível inserir, remover ou atualizar entradas em tempo de execução, o que oferece maior flexibilidade para adaptar o comportamento da inferência a mudanças no tráfego, a novos parâmetros do modelo ou a ajustes da política operacional, sem interromper o processamento no plano de dados.

Neste sentido, o modelo de ML escolhido molda a estrutura final do código-fonte em P4, pois a lógica de inferência define quais etapas precisam ser materializadas no pipeline e quais features devem estar disponíveis em cada estágio. Em modelos baseados em limiares, por exemplo, a implementação tende a priorizar comparações discretas e o encadeamento de decisões em tabelas; já em modelos com operações matemáticas mais custosas, torna-se necessário reorganizar o fluxo para preservar desempenho e viabilidade no plano de dados. Como o plano de dados não oferece suporte nativo a tais operações, exigidas por determinados modelos, o plano de controle pode pré-computá-las e enviá-las às MATs, onde passam a ser consultadas durante a inferência. Dessa forma, a divisão de responsabilidades entre plano de controle e plano de dados permite manter a inferência eficiente sem comprometer as limitações arquiteturais do *target*.

Com base nessas definições, a inferência no plano de dados inicia com a coleta dos dados que serão utilizados como *features*, por meio da extração de cabeçalhos de pacotes ou do cálculo de métricas derivadas do tráfego observado (ex.: tempo de chegadas entre pacotes). Em seguida, quando necessário, esses dados passam por etapas de tratamento e transformação, como normalização, discretização e adequação de tipos, para que fiquem compatíveis com as limitações do *target* e com a lógica implementada nas MATs. Esse pré-processamento é fundamental para preservar consistência entre os dados usados no treinamento *offline* e os valores processados em tempo real no pipeline, reduzindo erros de classificação decorrentes de diferenças de representação ou escala.

Por fim, a inferência do modelo é dividida em etapas intermediárias, nas quais cada MAT executa uma parte específica da lógica de decisão e gera resultados parciais que alimentam as etapas seguintes do pipeline, conforme pode ser observado na Figura

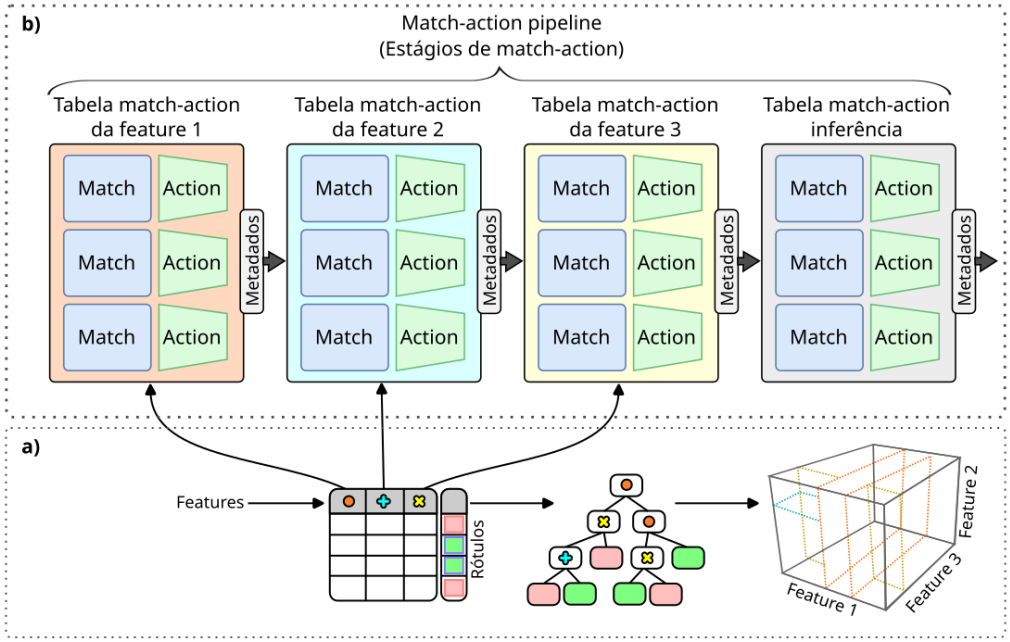

Figura 4. Árvores de Decisão: dos dados às MATs. (a) Dados de treinamento e Árvore de decisão resultante. (b) Estágios do pipeline que implementam a lógica do modelo

4. Essa decomposição permite organizar a inferência de forma incremental, reduzindo a complexidade de cada operação individual e facilitando a adaptação da implementação às restrições de memória, latência e número de estágios do *target*. Ao final desse encadeamento, os resultados intermediários são combinados para produzir a decisão final. Para viabilizar esse processo, as MATs utilizadas tanto nas etapas intermediárias quanto na classificação final são preenchidas pelo plano de controle com os parâmetros extraídos do modelo treinado, garantindo consistência entre o comportamento aprendido *offline* e a execução da inferência no plano de dados.

5. Caso de uso

O caso de uso apresentado nesta seção ilustra a execução prática do *framework* proposto em um ambiente de experimentação voltado para classificação de tráfego, no qual o modelo de ML é capaz de inferir diretamente no switch programável baseado no tráfego de entrada em tempo de execução.

O experimento foi conduzido a partir de um pipeline completo de *In-Network ML*, conforme descrito na seção anterior, e estruturado de forma sequencial e reprodutível. O fluxo experimental iniciou-se na aquisição e preparação dos dados e o treinamento do modelo através do Módulo de entrada, seguiu para a tradução da lógica de inferência para P4 com o Módulo de conversão e, por fim, passou pela configuração e implantação do ambiente através do Módulo de provisionamento.

Os dados de entrada (tráfego da rede) foram capturados através da ferramenta tcpdump, que permite a coleta de pacotes e seu armazenamento em formato de arquivos PCAP (*Packet CAPture*). Após a extração das features para um formato tabular, os dados se tornam viáveis para utilização no pré-processamento e treino do modelo no de entrada

com o uso do Scikit Learn[5].

Para este caso de uso foi utilizado a Árvore de Decisão devido a sua simplicidade e aderência à arquitetura do plano de dados. É importante esclarecer que outros modelos poderiam ser utilizados, tais como modelos probabilísticos, baseados em distância ou ainda redes neurais simplificadas. Para a etapa de conversão do modelo treinado para código P4, empregamos uma versão modificada e atualizada do framework de IISys[Xiong and Zilberman 2019].

O ambiente de experimentação foi construído sob uma infraestrutura de rede virtual composta por três nós: cliente, switch e servidor. O servidor disponibilizou diferentes classes de serviços, tais como: transmissão de vídeos adaptativos utilizando o padrão MPEG-DASH, transferência de arquivos em grande escala e acesso a páginas Web. O cliente consumiu esses serviços de forma automatizada, de maneira individual ou simultânea, gerando padrões de tráfego distintos. Com base nestas diferentes classes de tráfego, o switch é capaz de aplicar uma lógica de inferência e tomar decisões de classificação de acordo com o comportamento observado.

Foi utilizado o switch em software Behavioral Model v2 (BMv2)[6] devido ser a implementação de referência para fins didáticos da programabilidade do plano de dados com P4. Por meio da API do BMv2, a instância do switch é criada como processo do sistema operacional, com definição de interfaces físicas ou virtuais e carregamento das configurações em JSON, incluindo o código P4 compilado.

Para implementar o módulo de provisionamento, o Vagrant foi utilizado para descrever e gerenciar o ciclo de vida das máquinas virtuais por meio de arquivos em Ruby e comandos de linha. Para garantir que os dispositivos permaneçam operacionais e configurados de forma consistente, utilizou-se o Ansible como orquestrador, assegurando o estado desejado das máquinas em cenários de criação, inicialização, parada, reinicialização e destruição. Complementarmente, scripts em Bash foram utilizados para automatizar o consumo dos serviços pelo cliente, oferecendo controle preciso de tempo de execução e flexibilidade para compor diferentes padrões de carga.

Embora tenha sido realizada uma grande quantidade de experimentos neste caso de uso, esta seção destaca principalmente as métricas de desempenho do modelo de ML, como acurácia, recall, precisão e F1-Score. Ainda assim, a instrumentação experimental foi ampla e distribuída em diferentes camadas: no switch, por meio da API do BMv2, foram coletados indicadores como tamanho e atraso de fila, além de contadores e informações das tabelas; em nível de sistema, o uso de CPU e memória foi monitorado com o *top*; e o throughput da rede foi aferido com o *iperf*, com geração de tráfego de fundo para aumentar o realismo das condições avaliadas. Complementarmente, métricas específicas de cada serviço foram extraídas de logs e saídas padrão de servidores e clientes, conforme o mecanismo de instrumentação adotado.

Os resultados de desempenho do modelo, resumidos na Tabela 2, indicam que a implementação em *In-Network ML* manteve desempenho competitivo em relação ao modo convencional (validação *offline*), com ganhos em acurácia (86,05% versus 83,57%), precisão (98,72% versus 83,81%) e F1-Score (82,17% versus 83,68%), embora com

[5]`https://scikit-learn.org/`

[6]`https://github.com/p4lang/behavioral-model`

redução em recall (70,37% versus 83,57%). Esse comportamento evidencia um *trade-off*: o ganho de acurácia está associado a menor incidência de falsos positivos (maior precisão), ao custo de maior incidência de falsos negativos (menor recall). Ainda assim, os resultados reforçam a viabilidade da inferência no plano de dados com qualidade preditiva competitiva.

Tabela 2. Comparação do desempenho do modelo no modo convencional (validação *offline*) e em *In-Network ML*.

Métrica	Convencional (%)	In-Network ML (%)
Acurácia	83,57	86,05
Precisão	83,81	98,72
Recall	83,57	70,37
F1-Score	83,68	82,17

Como análise complementar ao desempenho preditivo do modelo, também foram observadas métricas de sistema e rede para caracterizar o impacto operacional da execução de *In-Network ML*. Em comparação ao pipeline convencional de P4, verificou-se aumento moderado no uso de CPU (média de 7,3%), enquanto o consumo de memória variou conforme o perfil de tráfego, com maior impacto em serviços web e transferência de arquivos e efeito mais discreto no streaming de vídeo. De forma semelhante, métricas de fila e vazão apresentaram comportamento dependente da carga, indicando que os efeitos da inferência embarcada estão mais associados ao padrão de tráfego do que a um custo fixo e uniforme da abordagem.

Ainda nesse contexto, a variação do número de features da Árvore de Decisão mostrou impacto limitado nas médias globais de CPU, memória e throughput, mas com sinais de maior sensibilidade em condições de pico, especialmente em percentis altos de atraso de fila e eventos de *stall*. Esses resultados reforçam a interpretação de que as métricas de infraestrutura devem ser lidas como evidências de viabilidade operacional do ambiente. Entretanto, cabe destacar que o objetivo central deste trabalho permanece a avaliação do desempenho do modelo de ML e da sua materialização no plano de dados, sendo as métricas de sistema e rede um suporte complementar para contextualizar a implantação.

6. Considerações finais

A integração de modelos de aprendizado de máquina diretamente no plano de dados programável representa um avanço relevante para a automação e a inteligência das redes, ao aproximar a tomada de decisão do caminho percorrido pelos pacotes. Neste trabalho, apresentamos e validamos um *framework* metodológico para materialização de pipelines de ML em switches programáveis, com foco na tradução da inferência para estruturas compatíveis com MATs e com as restrições operacionais do *target*.

Diferentemente de abordagens restritas a algoritmos ou arquiteturas específicas, a principal contribuição desta pesquisa é oferecer um roteiro generalista e reprodutível que conecta todas as fases do processo, da preparação dos dados à implantação em *targets* P4. A validação foi conduzida por meio de um caso de uso de classificação de tráfego com Árvore de Decisão em ambiente virtualizado com BMv2, mantendo o mesmo recorte de análise adotado na Seção 5.

Em termos de desempenho do modelo, os resultados mostraram comportamento competitivo entre o modo convencional e a execução em *In-Network ML*, com ganhos em acurácia e precisão e redução em recall, caracterizando um *trade-off* consistente com o perfil da inferência embarcada. Como análise complementar, as métricas de infraestrutura indicaram impacto operacional moderado: aumento médio de CPU na ordem de 7,3%, variações de memória dependentes do tipo de serviço e maior sensibilidade de atraso de fila em cenários de pico. Esses resultados reforçam a viabilidade prática da abordagem sem deslocar o foco principal da avaliação, que permanece no desempenho preditivo do modelo.

Como trabalhos futuros, planeja-se ampliar a validação do roteiro em hardwares físicos de alto desempenho, como switches com ASIC Tofino e SmartNICs, para analisar limites de memória e estágios de pipeline em ambientes reais. Também se pretende estender o suporte para modelos matematicamente mais complexos, como Máquinas de Vetores de Suporte e Redes Neurais, investigando estratégias de quantização e discretização compatíveis com P4. Por fim, pretende-se evoluir mecanismos de atualização dinâmica via plano de controle, viabilizando retreinamento e ajuste em tempo de execução sem necessidade de recompilação do plano de dados.

Referências

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Boutaba, R., Salahuddin, M., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo Rendon, O. (2018). A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, page 1–12, New York, NY, USA. Association for Computing Machinery.

Gebara, N., Ghobadi, M., and Costa, P. (2021). In-network aggregation for shared machine learning clusters. In Smola, A., Dimakis, A., and Stoica, I., editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 829–844.

Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., Frank, R., and Menth, M. (2021). A survey on data plane programming with p4: Fundamentals, advances, and applied research.

Kianpisheh, S. and Taleb, T. (2023). A survey on in-network computing: Programmable data plane and technology specific applications. *IEEE Communications Surveys & Tutorials*.

Kurose, J. F. and Ross, K. W. (2016). *Computer Networking: A Top-Down Approach*. Pearson, Boston, MA, 7 edition.

Lao, C., Le, Y., Mahajan, K., Chen, Y., Wu, W., Akella, A., and Swift, M. (2021). ATP: In-network aggregation for multi-tenant learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 741–761. USENIX Association.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Nguyen, H. N., Nguyen, M.-D., and Montes de Oca, E. (2024). A framework for in-network inference using p4. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ARES '24, New York, NY, USA. Association for Computing Machinery.

Sada, M. F., Graham, J., Tatineni, M., Mishin, D., DeFanti, T., and Würthwein, F. (2025). Real-time in-network machine learning on p4-programmable fpga smartnics with fixed-point arithmetic and taylor approximations. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*, PEARC '25, New York, NY, USA. Association for Computing Machinery.

Sapio, A., Canini, M., Ho, C.-Y., Nelson, J., Kalnis, P., Kim, C., Krishnamurthy, A., Moshref, M., Ports, D., and Richtarik, P. (2021). Scaling distributed machine learning with In-Network aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 785–808. USENIX Association.

Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., and Chan, K. (2020). When machine learning meets networking: A survey. *IEEE Communications Surveys & Tutorials*, 22(3):1609–1626.

Xiong, Z. and Zilberman, N. (2019). Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 25–33, New York, NY, USA. Association for Computing Machinery.

Zhang, K., Samaan, N., and Karmouch, A. (2024). A machine learning-based toolbox for p4 programmable data-planes. *IEEE Transactions on Network and Service Management*, 21(4):4450–4465.

Zhang, K., Zheng, C., Samaan, N., Karmouch, A., and Zilberman, N. (2026). Design, implementation, and deployment of multi-task neural networks in programmable data-planes. *IEEE Transactions on Network and Service Management*, 23:740–755.

Zheng, C., Hong, X., Ding, D., Vargaftik, S., Ben-Itzhak, Y., and Zilberman, N. (2024a). In-network machine learning using programmable network devices: A survey. *IEEE Communications Surveys & Tutorials*, 26(2):1171–1200.

Zheng, C., Zang, M., Hong, X., Perreault, L., Bensoussane, R., Vargaftik, S., Ben-Itzhak, Y., and Zilberman, N. (2024b). Planter: Rapid prototyping of in-network machine learning inference. *SIGCOMM Comput. Commun. Rev.*, 54(1):2–21.

Zhou, G., Liu, Z., Fu, C., Li, Q., and Xu, K. (2023). An efficient design of intelligent network data plane. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA. USENIX Association.