Um Framework para Realizar Aprendizado Federado em Ambientes com Recursos de Hardware Limitados

Igor L. Tomich[1] **e Guilherme Maia**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte – MG – Brazil

{igorlt96,jgmm}@dcc.ufmg.br

Abstract. *This paper presents a Federated Learning Framework adapted for resource-constrained environments, focusing on IoT devices. This is the first framework that enables federated training directly on microcontrollers. The framework demonstrates the autonomy of federated nodes, validating the feasibility of training models directly on microcontrollers. Two experiments were performed, showing promising results, despite challenges inherent to the environment, such as computational limitations, communicability, and scalability. Comparisons with related frameworks, such as TensorFlow-Federated, highlight the efficiency and dynamism of the proposed solution. The paper also discusses practical insights and improvements, contributing to the advancement of decentralized learning and the evolution of the TinyML scenario.*

Resumo. *Este artigo apresenta um framework de Aprendizado Federado adaptado para ambientes com recursos limitados, com foco em dispositivos IoT. Este é o primeiro framework que permite realizar treinamento federado diretamente em microcontroladores. O framework demonstra a autonomia dos nós federados, validando a viabilidade de treinar modelos diretamente em microcontroladores. Dois experimentos foram realizados, mostrando resultados promissores, apesar de desafios inerentes ao ambiente, tais como limitações computacionais, comunicabilidade e escalabilidade. Comparações com frameworks estabelecidos, como TensorFlow-Federated, destacam a eficiência e o dinamismo da solução proposta. O trabalho também discute ideias práticas e melhorias, contribuindo para o avanço do aprendizado descentralizado e para a evolução do cenário TinyML.*

1. Introdução

A Internet das Coisas (IoT) [Čolaković and Hadžialić 2018] redefiniu a interação entre dispositivos e sistemas com seu grande número de inclusões diárias. Essa revolução permitiu avanços e soluções inteligentes em vários contextos sociais, de áreas urbanas a setores agrícolas e industriais. Muitos desses dispositivos operam em um ambiente de computação de borda, desempenhando um papel vital na habilitação e amplificação da capacidade de processamento distribuído [Li et al. 2018]. De acordo com um estudo da International Data Corporation (IDC), estima-se que os dados globais atinjam 180 zettabytes (ZB), com 70% dos dados gerados pela IoT esperados para serem processados na borda da rede até 2025 [Shi et al. 2019]. Esse cenário exige abordagens inovadoras,

Pesquisa parcialmente financiada pela FAPEMIG, processos #APQ-02510-18 e #APQ-00426-22.

particularmente no reino do controle descentralizado capaz de operar vários dispositivos de borda juntos.

Paralelamente, avanços em técnicas de aprendizado de máquina (ML) vem crescendo significativamente. ML consiste no desenvolvimento de algoritmos e modelos que permitem reconhecer padrões para prever e tomar decisões com base no treinamento dos parâmetros de modelos por meio de uma grande quantidade de dados complexos [Wang et al. 2017]. No cenário de computação de borda, a crescente descentralização do ML se faz necessária, pois as soluções carecem de escalabilidade, possibilitando o aumento das capacidades de processamento com a adição de mais dispositivos. [Verbraeken et al. 2020]. Outro ponto importante é garantir a segurança e privacidade dos dados que podem ser processados localmente ou estão sob alguma regulamentação de proteção de dados. Os modelos de treinamento a partir de dispositivos IoT geralmente apresentam problemas de latência devido ao tempo de resposta do servidor central [Verbraeken et al. 2020]. Outro problema é a largura de banda, que pode causar gargalos na comunicação de dados com o servidor central ou até mesmo sobrecarregar a rede, principalmente em redes com conectividade limitada, impossibilitando, portanto, a utilização de ML centralizado em alguns cenários.

Como a computação de borda envolve a produção contínua de uma grande quantidade de dados, o Aprendizado Federado (FL) [Zhang et al. 2021] surge como uma abordagem inovadora no campo do aprendizado de máquina, especialmente adaptada para ambientes descentralizados. Nesse paradigma, o treinamento do modelo ocorre localmente em dispositivos distribuídos, preservando a privacidade dos dados, pois não é necessário transferir dados locais para um servidor central. Localmente, algoritmos de aprendizado específicos são empregados para ajustar os parâmetros do modelo. A colaboração entre dispositivos ocorre de forma federada, onde cada dispositivo contribui para o treinamento global, promovendo assim a convergência entre modelos locais. Essa abordagem pode oferecer vantagens significativas em ambientes com recursos computacionais limitados, pois o treinamento do modelo é totalmente distribuído, otimizando a eficiência da comunicação e minimizando a sobrecarga de dados entre dispositivos federados. No entanto, a implementação eficaz de técnicas de aprendizado de máquina em dispositivos com recursos limitados enfrenta desafios significativos. Termos como TinyML [Abadade et al. 2023] surgiram como referências para modelos de treinamento especificamente para dispositivos de borda como esses.

Pode-se destacar vários desafios ao tentar aplicar sistemas de FL em dispositivos com grande restrição computacional. Por exemplo, como reduzir o uso de memória do dispositivo ou mesmo diminuir a latência na execução da etapa de treinamento no dispositivo, pois os dispositivos têm memória flash e memória de acesso aleatório (RAM) restrita e normalmente têm um núcleo de processamento com arquiteturas de hardware simples, operando em frequências mais baixas em comparação aos computadores convencionais. Logo, o problema em estudo nesse trabalho é como realizar e escalar o treinamento de modelos de ML de forma federada em ambientes com dispositivos com grandes limitações de hardware, garantindo a convergência do treinamento e obtendo desempenho satisfatório para um conjunto de aplicações.

Dado o cenário acima, este trabalho propõe um framework para realizar o treinamento de modelos de ML de forma federada diretamente em microcontroladores, respei-

tando as limitações de recursos computacionais desses dispositivos. Diferente de soluções existentes na literatura, o framework proposto é o único que permite realizar a etapa de treinamento federado diretamente nos microcontroladores. A solução proposta abstrai diversos aspectos do processo de aprendizado federado, além de oferecer uma interface simples para os desenvolvedores. Desse modo, o framework demanda apenas a definição de alguns hiperparâmetros, como o número de dispositivos clientes, configuração de épocas por dispositivos e configurações de sincronização com o servidor de FL. De forma a demonstrar a viabilidade de se realizar treinamento federado diretamente em sistemas embarcados, uma comparação com o estabelecido *TensorFlow-Federated* foi realizada, destacando a eficiência e o dinamismo da solução proposta.

Este trabalho é organizado da seguinte forma: a Seção 2 apresenta trabalhos relacionados ao tema de aprendizado de máquina com dispositivos embarcados. Em seguida, a Seção 3 descreve em detalhes o desenvolvimento da arquitetura da estrutura de aprendizado federado para o problema proposto. Na Seção 4, um caso de uso é apresentado, que usa o conjunto de dados da flor iris para simular o ambiente FL e comparar a solução proposta com o *TensorFlow-Federated*. Finalmente, a Seção 5 fornece as conclusões obtidas ao longo do trabalho e propostas para trabalhos futuros.

2. Trabalhos Relacionados

A literatura foi revisada para encontrar trabalhos que forneçam suporte para o desenvolvimento de soluções de ML em um ambiente descentralizado que opera em pequenos dispositivos para treinamento de nós de rede federados.

O trabalho de Llisterri Gimenez et al.˜[Llisterri Giménez et al. 2022] levanta pontos importantes na implementação de aprendizado distribuído para reconhecimento de palavras-chave. Usa-se uma *placa Arduino Nano 33 BLE Sense*. A arquitetura do sistema é projetada para que o treinamento seja totalmente supervisionado com o objetivo principal de reconhecer duas palavras pelo usuário. No entanto, o trabalho tem uma aplicação única e é dedicado a um único problema, pois não possui uma estrutura de ferramentas que suporte o treinamento como um framework dedicado.

O trabalho de Ficco et al. [Ficco et al. 2024] cria outra abordagem geral para o treinamento em microcontroladores. A ideia principal é montar um sistema de aprendizado federado para pequenos dispositivos heterogêneos, cada um com suas próprias características, limitações e restrições. Um dos principais recursos explorados no trabalho é a quantidade de RAM, que está diretamente relacionada à complexidade do modelo que está sendo treinado e à latência de aprendizagem que está atrelada ao poder computacional do dispositivo. O trabalho avalia como diferentes hardwares se comportam com diferentes métricas de aprendizagem em cenários com e sem FL. Pontos fundamentais são levantados, especialmente quanto à aplicação da técnica proposta em cenários do mundo real para avaliar seu desempenho em vários domínios.

Além do escopo do treinamento e das restrições do microcontrolador, alguns estudos abordam a etapa de treinamento federado. O trabalho de [Wulfert et al. 2023] traz um novo conceito chamado TinyFL. Por meio da integração do barramento de comunicação de circuito integrado (I^2C), o TinyFL usa um protocolo mestre/escravo híbrido onde o MCU mestre é responsável pela comunicação e agregação. Usando o reconhecimento de gestos como estudo de caso, a abordagem deste estudo atingiu uma velocidade 11,5%

maior do que o treinamento centralizado. No entanto, além do barramento (I^2C) ter um alcance relativamente curto, ele pode sofrer gargalos quando estressado em um sistema multimestre hierárquico de forma assíncrona, comprometendo todo o sistema federado.

A maioria das soluções TinyML não fornece um sistema robusto para configurar o aprendizado federado, especialmente em relação a hiperparâmetros e parâmetros de redes neurais. Existem abordagens interessantes, como usar servidores em nuvem para treinamento em dispositivos restringidos computacionalmente ou usar plataformas bem conhecidas como TensorFlow Lite para habilitar a inferência de modelos nos dispositivos. No entanto, a integração dessas soluções pode tornar a implementação cara e o processo de treinamento complexo. Este projeto apresenta uma estrutura de aprendizado federado com opções de configuração para hiperparâmetros e parâmetros de rede neural para microcontroladores, como será mostrado nas seções a seguir.

3. Desenvolvimento do Framework

Esta seção descreve o desenvolvimento do framework FL inicialmente para o microcontrolador ESP32 [Rai and Rehman 2019] para analisar e estudar o framework em comparação com a plataforma *TensorFlow-Federate*. O Framework consiste em um servidor que deve ser capaz de compilar e instanciar um modelo global para disponibilizá-lo aos dispositivos clientes, onde eles treinarão o modelo global instanciado a partir de um conjunto de dados local, simulando a capacidade de ser autônomo e descentralizado. O servidor responsável pela aplicação deve receber os modelos locais treinados pelos microcontroladores em cada rodada federada e agregá-los ao modelo global para classificar todas as classes disponíveis do conjunto de dados escolhido para o treinamento. Após cada rodada predefinida, o servidor apresenta as métricas de validação do aprendizado federado. As subseções a seguir descreverão com mais detalhes a funcionalidade do servidor e cliente FL, algoritmos, configurações de hardware e fluxo de trabalho do framework.

3.1. Arquitetura do Aprendizado Federado

Uma arquitetura de aprendizado federado é implementada para executar qualquer tamanho de modelo, inicialmente compatível com a restrição do módulo ESP32 em estudo. Essa arquitetura consiste em um servidor central que controla vários módulos como um nó federado em uma rede *Wi-Fi* por meio de um roteador local.

A estrutura da arquitetura ilustrada na Figura 1 consiste em três subestruturas fundamentais vinculadas chamadas *FederatedLearning*, *NodeControl* e *NeuralNetwork*. A estrutura *FederatedLearning* representa a estrutura principal da arquitetura responsável por indicar o status do modelo global e armazenar a instância das estruturas *NeuralNetwork* e *Nodecontrol*. A estrutura *Neuralnetwork* possui os parâmetros necessários para a configuração e treinamento da rede neural profunda, na qual possui ponteiros para montar modelos de camadas e pesos de neurônios. Por fim, a estrutura *NodeControl* é responsável por gerenciar os nós clientes registrados na rede federada e controlar a agregação dos modelos locais treinados no modelo global por meio de outra instância da estrutura *NeuralNetwork*.

Os tipos de neurônios implementados até então para o *framework* foram os neurônios de entrada, saída e ocultos, necessários para a configuração de uma rede neural profunda. As funções de ativação implementadas para os neurônios ocultos são *ReLU*,

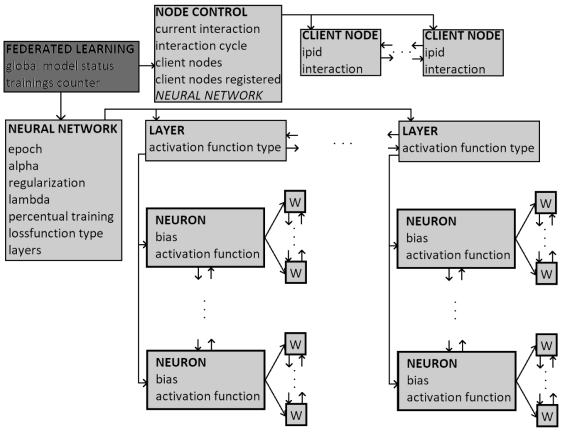

Figura 1. Estrutura Geral do Framework de Aprendizado Federado

Sigmoid e *Perceptron*. Para os neurônios de saída, a função *Softmax* está disponível, o que é essencial para redes de classificação multiclasse. Até o dado momento, o *framework* fornece duas funções de perda, a saber, Categorical cross entropy e minimal mean square. Além disso, também há duas funções de regularização disponíveis, Lasso e Ridge, para minimizar o efeito de overfitting. Com essas implementações, é possível modelar diversos modelos, pois as estruturas *Layer*, *Neuron* e *Weight* têm ponteiros para si mesmas, permitindo a alocação dinâmica de tamanho n limitado pela RAM do dispositivo e também são duplamente encadeadas para facilitar o processo de *feed-forward* e *back-propagation* da rede.

3.1.1. Servidor do Aprendizado Federado

O servidor de arquitetura foi implementado em linguagem C. Dessa forma, há um maior controle sobre onde ele pode ser executado em um computador convencional ou mesmo em outro dispositivo com recursos restritos, visando, por exemplo, uma futura implementação horizontal ou em camadas da rede federada. O servidor utiliza o padrão de criação Singleton na estrutura FL supracitada, garantindo uma única instância global acessível. Para proteger o acesso concorrente, cada vez que é feita uma tentativa de acesso à estrutura Singleton, uma chave mutex é utilizada, evitando a corrupção de dados durante as operações de leitura e escrita. Ele também é responsável por gerenciar todo o controle dos nós federados através da subestrutura *NodeControl*.

Além disso, o servidor é responsável por prover um serviço *HTTP* para consumo da API, no qual o microcontrolador pode solicitar o modelo global, verificar a disponibilidade do modelo global e se registrar como um nó válido. E para que o cliente possa enviar modelos locais, o servidor disponibiliza um serviço *Websocket* com um sistema de controle de *buffer* de mensagens para o controle de múltiplos nós clientes simultâneos na rede.

Para este projeto, o Federated Averaging (FedAVG) [Sun et al. 2023] é escolhido como o método de algoritmo de agregação. Ele é usado para combinar modelos treinados localmente de dispositivos de borda em um modelo global em um servidor central. Ele opera em rodadas iterativas, onde cada rodada envolve o treinamento do modelo local

seguido pela agregação de atualizações do modelo. *FedAVG* emprega a média ponderada para garantir uma representação equilibrada no modelo global, facilitando a colaboração eficaz, preservando a privacidade dos dados e minimizando a sobrecarga de comunicação. Essa abordagem torna o *FedAVG* um algoritmo fundamental para sistemas de ML descentralizados. Finalmente, para avaliar o modelo, o servidor executa as funções de acurácia, precisão, revocação, especificidade e *F1-Score* implementadas para essa finalidade em cada rodada federada.

3.1.2. Cliente do Aprendizado Federado

Como o cliente tem recursos computacionais limitados, ele tem o mínimo de tarefas possíveis. Inicialmente, ele configura a comunicação serial *UART*, estabelece a comunicação de rede *Wi-Fi*, formata e particiona o sistema de arquivos e, finalmente, configura os pinos de entrada e saída. O nó cliente tem implementações de comunicação *http* para requisitar o modelo global, registrar-se no servidor e verificar a disponibilidade do modelo global. Há também a configuração da comunicação *Websocket* com o servidor apenas para enviar o modelo local treinado.

3.2. Configuração do Hardware

Para interagir com a aplicação, é necessária uma etapa de preparação do hardware. A aplicação é implantada em módulos *ESP32-WROOM-32*, que já integram vários dos componentes. Uma etapa importante é configurar as partições do projeto, pois a memória é um recurso limitado, sendo de no mínimo 4 MB. Quatro partições foram configuradas, a primeira *NVS* de 16 KB, *PHYINIT* de 4 KB, *FACTORY* de 1 MB e o restante na partição *STORAGE*. Elas são usadas respectivamente para armazenar as configurações não voláteis do sistema, inicialização física da camada de rede, armazenar dados do sistema e código embarcado e finalmente armazenar os dados que serão usados na simulação de treinamento mostrada na Tabela 1.

Tabela de Partições				
Nome	Tipo	SubTipo	Offset	Tamanho
NVS	data	nvs	0x9000	0x4000
PGYINIT	data	phy	0xf000	0x1000
FACTORY	app	factory	0x10000	1M
STORAGE	data	spiffs	0x110000	0x100000

Tabela 1. Tabela de representação do sistema de partição do dispositivo ESP32.

Para melhor visualização do funcionamento do sistema e para ter uma melhor forma de depuração, foram configurados um botão e três LEDs para indicar os passos de execução do algoritmo, como pode ser visto na Figura 2

O botão é usado para inicializar o aplicativo após a configuração base inicial do dispositivo. O LED verde tem dois estados, em que o estado piscando indica que está esperando o botão ser pressionado para inicializar o aplicativo, enquanto o estado ligado indica que o aplicativo está em operação. O LED azul sinaliza qualquer conexão com o servidor via solicitação *HTTP* ou usando o canal *WebSocket* para enviar o modelo local.

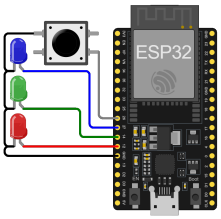

Figura 2. Configuração do Hardware.

Por fim, o LED vermelho indica falhas do sistema, em que uma piscada indica falha de inicialização, duas piscadas indicam falha de comunicação com o servidor e três piscadas indicam falha de treinamento.

3.3. Funcionamento Dinâmico do Sistema

Inicialmente, o servidor inicializa e compila o modelo FL, que consiste basicamente na construção dinâmica das três estruturas descritas na Seção 3.1, onde o número de nós clientes e o número de rodadas federadas são definidos para o controle dos nós. Quanto ao treinamento da rede, o número de camadas, a função de ativação para os neurônios da camada, o número de neurônios por camada, o tipo de regularização, a taxa de regularização, o número de épocas por rodada federada e a função de perda podem ser definidos. Logo após, os serviços HTTP e Websocket são inicializados em threads diferentes para que os nós clientes possam fazer as requisições necessárias.

Após a configuração inicial, o servidor aguarda o registro de todos os nós definidos no momento da compilação, armazenados na estrutura *NodeControl*. Quando todos os registros foram feitos, o servidor libera o modelo global para que os nós clientes possam solicitá-los. Agora o servidor entra em um estado de loop aguardando o modelo local de todos os nós registrados. Quando um modelo local é enviado ao servidor, o algoritmo FedAvg é executado em uma cópia da rede instanciada na estrutura *NodeControl*. Dessa forma, se um nó na rede perder o treinamento por algum motivo, ele pode solicitar o modelo global da rodada atual novamente. Assim que todos os nós enviam seus modelos, o servidor sai do loop de espera, copia a rede neural da estrutura *NodeControl* para a rede neural principal e executa as métricas de desempenho. Finalmente, o servidor entra em outro loop no qual verifica se todas as rodadas foram executadas. Caso contrário, o servidor libera o modelo global novamente e repete o processo de rodada federada. Então, após todas as rodadas federadas, a aplicação é encerrada. Uma melhor visualização do fluxo de trabalho do servidor pode ser vista na Figura 3a.

As tarefas nos microcontroladores são mínimas devido às suas restrições computacionais. Inicialmente, o nó cliente inicia suas configurações base iniciais mencionadas na Seção 3.1.2 para ter as condições necessárias para executar o aplicativo. Então, o nó faz uma solicitação *HTTP* para se registrar como um nó operacional na rede federada. Então, ele faz outra solicitação *HTTP* para verificar se o modelo global está disponível. Se não estiver disponível, um loop é inicializado para verificar a disponibilidade do modelo até que ele esteja disponível. Quando o modelo global está finalmente pronto para o treinamento, o loop é encerrado, continuando o fluxo do aplicativo, que subsequentemente faz uma nova solicitação à estrutura de aprendizado federado descrita na Seção 3.1. Com o modelo instanciado no dispositivo, o treinamento local inicializa com os da-

dos disponíveis na partição *STORAGE*, apresentada na tabela 1, e com as configurações predefinidas na compilação do modelo. Após a conclusão do treinamento, o dispositivo abre uma conexão *WebSocket* para enviar o modelo local ao servidor. Por fim, o processo retorna à etapa de verificação da disponibilidade do modelo global para uma nova rodada federada. A Figura 3b ilustra o fluxo de operação do nó cliente conforme descrito acima.

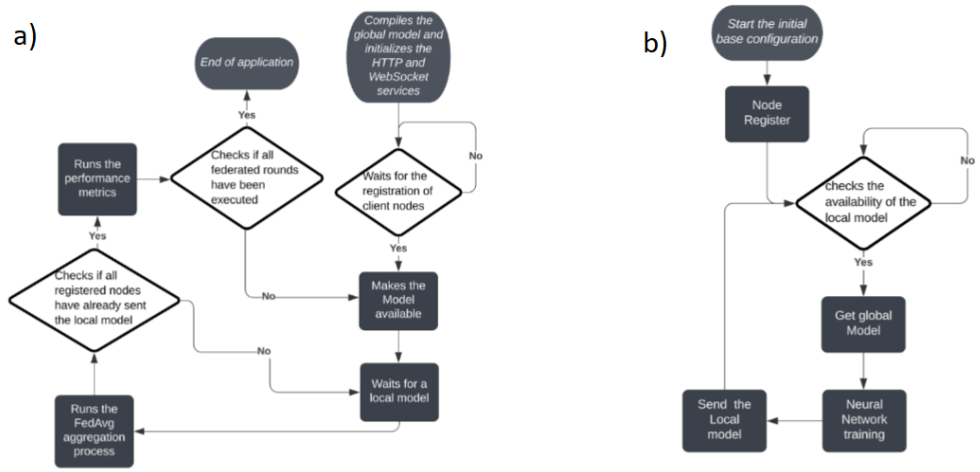

Figura 3. Fluxograma do funcionamento do sistema dinâmico do *Framework*.

4. Caso de Uso

Nesta seção, apresentamos primeiramente o estudo de caso envolvendo a descrição do conjunto de dados e sua aplicação do *framework* apresentado na seção 3. Além disso, apresentamos o modelo e os hiperparâmetros escolhidos para o treinamento federado. Dois experimentos foram conduzidos, nos quais o primeiro experimento executa o conjunto de dados de forma federada em três microcontrolador ESP32 como nós cliente da rede. O segundo experimento apresenta a execução do *framework* TFF em comparação com o *framework* desenvolvido neste trabalho para demonstrar o desempenho do treinamento federado.

4.1. Conjunto de Dados

Para analisar o desempenho e a operação do framework, escolhemos o conjunto de dados iris, que abrange três tipos da flor iris: Setosa, Versicolor e Virginica. Cada uma caracterizada por quatro características: comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala. Este conjunto de dados tem 150 amostras, das quais há 3 classes representando a classificação dos tipos de flores de íris. Cada classe tem a mesma quantidade de amostras, totalizando um terço para cada classe.

4.2. Treinamento do modelo de rede neural

É importante construir um modelo ideal para o conjunto de dados escolhido para poder avaliar o desempenho do framework e se ele realmente opera corretamente. É utilizada uma rede neural totalmente conectada composta por quatro camadas. A primeira camada de entrada é composta por quatro nós, responsáveis por receber as quatro características da amostra da flor de íris. Seguidas por duas camadas ocultas com três neurônios cada, ambas com funções de ativação *ReLU* (Rectified Linear Unit). O uso da função de ativação

ReLU se deve à mitigação do problema do gradiente de fuga, uma vez que suas derivadas são constantes para entradas positivas, facilitando a propagação do gradiente em redes profundas. Além disso, *ReLU* é computacionalmente simples e promove a escassez na ativação, melhorando a generalização do modelo e tornando-o menos propenso a overfitting. Por fim, a última camada, chamada de camada de saída, possui três neurônios porque as amostras são distribuídas em três classes. Portanto, ele usa a função de ativação *Softmax* para classificação, que é ideal para transformar os valores de saída do modelo em uma distribuição de probabilidade, onde a soma das probabilidades é 1, tornando mais fácil interpretar as previsões do modelo como probabilidades de pertencer a cada classe. O modelo pode ser melhor visualizado na Figura 4

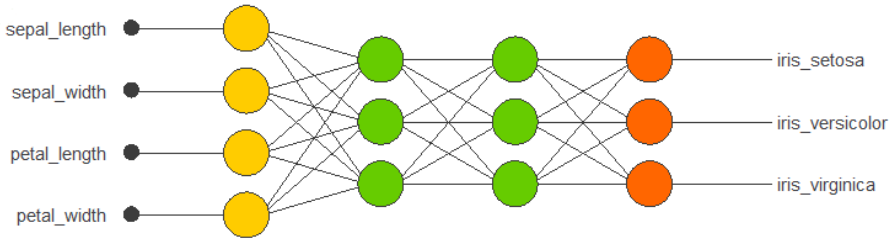

Figura 4. O modelo usado no projeto onde as células amarelas representam os neurônios de entrada, as células verdes representam os neurônios ocultos e as células laranja representam as células de saída

A função de perda *Entropia Cruzada Categórica* é amplamente usada em tarefas de classificação em Deep Learning porque mede a divergência entre a distribuição de probabilidade prevista pelo modelo e a distribuição real das classes. É ideal para problemas de classificação multiclasse porque penaliza fortemente previsões que atribuem alta probabilidade às classes erradas, incentivando o modelo a ser mais preciso. Ao minimizar a entropia cruzada categórica, o modelo ajusta seus pesos para aumentar a probabilidade das classes corretas, resultando em previsões mais precisas e confiáveis.

Para otimizar o treinamento do modelo, é possível ajustar os hiperparâmetros, sendo a taxa de aprendizado um dos mais importantes. Essa taxa controla o quanto o modelo é atualizado em resposta ao erro estimado em cada época de treinamento. Escolher a taxa de aprendizado correta é um desafio considerável. Um valor muito pequeno pode resultar em um longo período de treinamento, enquanto um valor muito alto pode tornar o processo de treinamento instável. A taxa de aprendizado padrão usada em nossa aplicação é 0,001. A ideia principal é permitir que os nós treinem por um período suficiente para que nenhum deles tenha um impacto significativo no momento da agregação pelo servidor.

Além da taxa de aprendizado, outro hiperparâmetro crucial é definir o número de épocas que cada nó treinará antes de enviar seu modelo local para o servidor e solicitar uma atualização. Essa escolha é vital para equilibrar o tempo de treinamento local e a frequência de atualização do modelo. Se o número de épocas for muito baixo, o modelo local pode não ter tempo suficiente para aprender padrões significativos nos dados. Por outro lado, um número excessivamente alto de épocas pode levar a um desperdício de recursos computacionais e possível ajuste excessivo aos dados locais. Portanto, encontrar o equilíbrio ideal para o número de épocas é essencial para garantir um treinamento eficaz e uma colaboração suave entre os nós e o servidor no sistema de aprendizado federado.

Para finalizar a configuração do modelo, a regularização *L2* (Regressão de Ridge) foi adicionada, onde a função de perda é aumentada com a soma de todos os pesos ao quadrado. Em geral, ele evita o ajuste excessivo ao penalizar pesos que tiveram crescimento significativo em comparação a outros pesos.

4.3. Execução do Framework

Para analisar o desempenho do framework em um ambiente federado, há 3 módulos do ESP32 na rede como nós federados. Geralmente, um desempenho pior é esperado quando um modelo é treinado de forma federada, quando comparados com um treinamento centralizado. Neste experimento, as 120 amostras disponíveis foram divididas igualmente entre os 3 nós da rede, totalizando 40 amostras distintas para cada um. A Figura 5 mostra a média da função de perda do modelo local entre os 3 dispositivos.

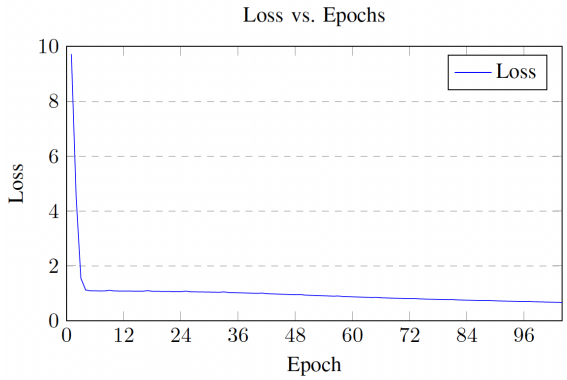

Figura 5. Perda vs. épocas durante o processo de treinamento nos três dispositivos.

Na Tabela 2 podemos ver a avaliação do modelo calculada pelo servidor através de métricas de desempenho para 3 nós federados. Esses indicadores são cruciais para uma análise abrangente e precisa da capacidade do modelo de fazer previsões corretas e identificar corretamente as classes de interesse.

Ao distribuir o treinamento em 3 nós, cada um processando 40 amostras distintas, o experimento destaca o desafio da diversidade de dados reduzida por nó, o que poderia afetar potencialmente o desempenho dos modelos individuais. No entanto, o objetivo principal dessa abordagem era demonstrar a capacidade da estrutura de treinar modelos de rede neural profunda em dispositivos de baixa computação de forma federada. Os resultados confirmam a viabilidade de combinar treinamento no dispositivo em microcontroladores com aprendizado federado em uma estrutura dedicada, apesar dos desafios relacionados à comunicação, escalabilidade e recursos computacionais limitados. Essas descobertas ressaltam a capacidade da estrutura de agregar eficientemente modelos treinados em subconjuntos distribuídos de dados, tornando-a uma solução viável para aprendizado profundo baseado em computação de borda.

4.4. Comparando com o Framework Tensorflow-Federated

A comparação com um *framework* conhecido como o TensorFlow, aceito pela comunidade, é importante para avaliar o bom funcionamento e dinamismo do *framework* desenvolvido. O *TensorFlow* possui ferramentas de aprendizado federado chamadas

Métricas de desempenho por rodada					
Rodada	**Acurácia**	**Precisão**	**Recall**	**Especificidade**	**F1-Score**
1	0.76	0.27	1.00	0.73	0.42
2	0.76	0.30	0.90	0.74	0.45
3	0.77	0.33	0.91	0.75	0.49
4	0.78	0.43	0.81	0.77	0.57
5	0.82	0.57	0.85	0.81	0.68
6	0.81	0.57	0.81	0.81	0.67
7	0.81	0.57	0.81	0.81	0.67
8	0.81	0.57	0.81	0.81	0.67
9	0.82	0.57	0.85	0.81	0.68
10	0.82	0.57	0.85	0.81	0.68
11	0.87	0.70	0.88	0.86	0.78
12	0.92	0.87	0.90	0.93	0.88

Tabela 2. Métricas de desempenho para cada rodada durante o treinamento de aprendizado federado no ESP32.

TensorFlow-Federated (TFF), que foram utilizadas para comparar os desempenhos associados a cada *framework*. É muito difícil criar o mesmo modelo idêntico em *frameworks* diferentes e ao mesmo tempo obter os mesmos resultados, pois pode haver pequenas alterações nos algoritmos e diferentes formas de precisão e conversão de ponto flutuante de uma linguagem para outra. Outro ponto importante a ser avaliado é a forma como os parâmetros são aplicados nos diferentes *frameworks*, principalmente aqueles relacionados ao aprendizado federado. Dito isso, foi montado um modelo o mais fiel possível no TFF, seguindo a mesma ideia de um modelo de 4 camadas em que uma se refere à camada de entrada de 4 neurônios, seguida de 2 camadas intermediárias de 3 neurônios com ativação *Relu* e finalmente uma camada de saída com ativação *Softmax*. A configuração federada também consiste em 3 nós com a mesma taxa de aprendizado de 0,001, também usando *Entropia cruzada categórica* como a função de perda e também o mesmo otimizador de treinamento *Stochastic Gradient Descent*. É importante destacar que o TFF não permite o treinamento federado (permite apenas a fase de inferência) diretamente nos microcontroladores, logo, uma rede com os 3 nós foi simulada.

Os dados também foram divididos de forma que 20% das amostras foram usadas para avaliação do modelo e os 80% restantes foram divididos com a mesma quantidade de 40 amostras para os 3 nós de simulação da estrutura TFF. Após vários treinamentos com a mesma configuração, resultados semelhantes sempre foram obtidos, conforme mostrado na Figura 6 representando um desses treinamentos.

Uma similaridade notável na precisão entre a estrutura desenvolvida e o TFF também é observada. Os resultados demonstraram que ambas as estruturas convergiram para uma precisão comparável, como pode ser visto na Fig. 7, mostrando outra maneira de validar a eficácia da estrutura em relação ao TFF.

Em resumo, além da compatibilidade observada na função de perda entre as rodadas federadas, a precisão alcançada pelo *framework* proposto foi consistentemente alinhada com aquela obtida pelo TFF. Essa correspondência tanto em perda quanto em pre-

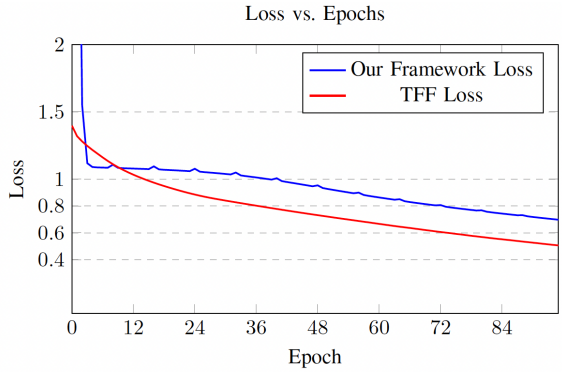

Figura 6. Perda vs. épocas durante o processo de treinamento.

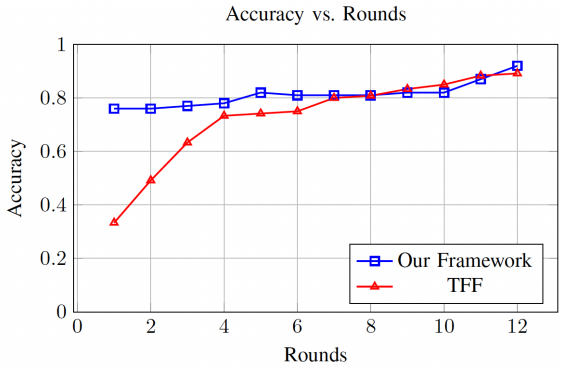

Figura 7. Precisão vs. rodadas federadas durante o processo de treinamento.

cisão valida a robustez e eficácia do *framework* desenvolvido, demonstrando que ele pode atingir resultados equivalentes aos de um *framework* amplamente reconhecido como o TFF. Assim, os resultados sugerem que o *framework* não é apenas uma alternativa viável, mas também uma ferramenta confiável e competitiva para a implementação de sistemas de aprendizado federados em dispositivos de recursos restritos e computação de borda.

4.5. Avaliação da Comunicabilidade na Rede

Foi o utilizado a ferramenta *WireShark* para análise do tráfego de pacotes na rede. O projeto utiliza um roteador ASUS RT-AC1200 operando na banda de 2,4 GHz para a conexão *wi-fi* dos microcontroladores.

Os dados da Tabela 3 indicam que as operações CheckGlobalModel e GetGlobalModel, ambas do tipo GET, possuem pacotes pequenos e latências relativamente baixas, sendo CheckGlobalModel ainda mais leve. Apesar de GetGlobalModel e SendGlobalModel transmitirem modelos de tamanho equivalente na teoria (o mesmo modelo com pesos ajustados após o treinamento), observa-se uma diferença significativa nas latências. Enquanto GetGlobalModel apresenta picos ocasionais, PostGlobalModel, realizada via WebSocket, possui latências consistentemente mais altas, em torno de 1 segundo, o que pode ser atribuído ao overhead de envio ou processamento no servidor. Essa diferença sugere que fatores como estado da conexão WebSocket ou carga do servidor podem impactar o desempenho.

Round	CheckGlobalModel		GetGlobalModel		SendGlobalModel	
	Pacotes	Latência	Pacotes	Latência	Pacotes	Latência
1	5	26ms	7	36ms	8	1044ms
2	5	20ms	7	29ms	9	1034ms
3	5	17ms	7	24ms	9	1021ms
4	5	26ms	7	55ms	9	1026ms
5	5	20ms	7	63ms	8	1024ms
6	5	16ms	7	56ms	8	1029ms
7	5	16ms	7	57ms	9	1036ms
8	5	24ms	7	45ms	8	1032ms
9	5	16ms	7	61ms	8	1028ms
10	5	32ms	7	80ms	6	1036ms
11	5	21ms	7	27ms	8	1023ms
12	5	18ms	7	75ms	8	1033ms

Tabela 3. Tabela com dados de pacotes e latências das operações CheckGlobalModel, GetGlobalModel e PostGlobalModel.

4.6. Métricas de Consumo de Energia

O módulo INA219 foi integrado via barramento I2C para medir o consumo de energia. A tensão, conforme mostrado na Tabela 4, representa a tensão na carga, calculada como a soma da tensão de derivação (queda de tensão no resistor) e da tensão do barramento (tensão da fonte). Tais resultados mostram que diferentes soluções de alimentação de energia podem ser empregadas durante a utilização do framework, como por exemplo, pequenos painéis solares integrados ou bateria de lítio.

State	Load Voltage (V)	Current (mA)	Power (mW)
Idle	5.03	68.10	340.00
Training	5.02	83.10	410.00

Tabela 4. Métricas de consumo de energia durante as fases de inatividade e treinamento, capturadas pelo módulo INA219.

5. Conclusão e Trabalhos Futuros

Foi apresentado um *framework* de aprendizado federado, que realiza treinamento no dispositivo de um modelo de rede neural usando módulos *ESP32-WROON-32*. Um caso de uso descreveu como o aplicativo treina rótulos de flores de íris em cada dispositivo de forma federada, demonstrando a autonomia dos nós na rede. Dois experimentos foram realizados: um descreveu a aplicação do *framework* com três nós de forma federada e o outro experimento foi a comparação do *framework* desenvolvido com o *Tensorflow-Federated*. Os experimentos também mostraram uma tendência decrescente nas perdas conforme as rodadas de treinamento aumentaram, conforme esperado. Os resultados de ambos os experimentos indicam a viabilidade de combinar treinamento no dispositivo em microcontroladores com aprendizado federado em um *framework* dedicado, apesar dos desafios de comunicabilidade, escalabilidade e escassez de recursos computacionais por parte dos microcontroladores.

Dentre as melhorias que podem ser feitas no *framework* pode-se destacar, a implementação de mais funções de ativação, funções de perda e outros métodos de otimização. Um outro ponto interessante seria implementar outros meios de comunicação mais adequados para microcontroladores, como comunicação em rede *LoRa* ou serial bus.

Referências

Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., and Hafid, A. S. (2023). A comprehensive survey on tinyml. *IEEE Access*, 11:96892–96922.

Ficco, M., Guerriero, A., Milite, E., Palmieri, F., Pietrantuono, R., and Russo, S. (2024). Federated learning for iot devices: Enhancing tinyml with on-board training. *Information Fusion*, 104:102189.

Li, H., Ota, K., and Dong, M. (2018). Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE Network*, 32(1):96–101.

Llisterri Giménez, N., Monfort Grau, M., Pueyo Centelles, R., and Freitag, F. (2022). On-device training of machine learning models on microcontrollers with federated learning. *Electronics*, 11(4).

Rai, P. and Rehman, M. (2019). Esp32 based smart surveillance system. In *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–3.

Shi, W., Pallis, G., and Xu, Z. (2019). Edge computing [scanning the issue]. *Proceedings of the IEEE*, 107(8):1474–1481.

Sun, T., Li, D., and Wang, B. (2023). Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4289–4301.

Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., and Rellermeyer, J. S. (2020). A survey on distributed machine learning. *ACM Computing Surveys*, 53(2):1–33.

Wang, P., Li, Y., and Reddy, C. K. (2017). Machine learning for survival analysis: A survey.

Wulfert, L., Wiede, C., and Grabmaier, A. (2023). Tinyfl: On-device training, communication and aggregation on a microcontroller for federated learning. In *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*, pages 1–5.

Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, 216:106775.

Čolaković, A. and Hadžialić, M. (2018). Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144:17–39.