MiniNetFED: Uma Ferramenta para Emulação e Análise de Aprendizado Federado com Dispositivos Heterogêneos

Johann Jakob Schmitz Bastos, Eduardo Montagner de Moraes Sarmento, Rodolfo da Silva Villaça, Vinícius F. S. Mota

[1]Departamento de Informática – Universidade Federal do Espírito Santo (Ufes)
Vitória/ES – Brasil

{johann.bastos, eduardo.sarmento}@edu.ufes.br
{rodolfo.villaca, vinicius.mota}@inf.ufes.br

Abstract. *This paper presents the MiniNetFed, an emulation tool for analyzing federated learning algorithms. The MiniNetFed enables the emulation of an environment with heterogeneous devices, differing in processing power, memory, and network capability, for conducting federated learning experiments. The tool also allows users to define: i) data partition formats among devices; ii) client selection policies; iii) model aggregation functions; iv) pre-trained models for major datasets used in the literature; and v) graphical visualizations of key performance metrics. Finally, besides its educational potential, the MiniNetFed is designed so that new algorithms and models can be easily extended, allowing researchers to implement and evaluate their proposals.*

Resumo. *Este artigo apresenta o MiniNetFed, uma ferramenta de emulação de um ambiente com dispositivos heterogêneos para análise de algoritmos de aprendizado federado. A ferramenta permite aos usuários definirem: i) modo de divisão do conjunto de dados entre os dispositivos; ii) políticas de seleção de clientes; iii) função de agregação de modelos; iv) utilizar modelos para os principais datasets disponíveis; e v) visualizações gráficas das principais métricas de desempenho. Por fim, além do potencial educacional, o MiniNetFed foi projetado de forma que novos algoritmos e modelos sejam facilmente estendidos, permitindo que pesquisadores implementem e avaliem suas propostas.*

1. Introdução

Em nosso cotidiano, interagimos com uma série de dispositivos inteligentes e conectados, que formam a Internet das Coisas (IoT) [Minerva et al. 2015]. Entretanto, muitas vezes os dados sensoreados contém informações sensíveis, sendo comumente transmitidos, armazenados e processados na nuvem. Neste sentido, o aprendizado federado surge como um mecanismo distribuído em que os próprios dispositivos treinam e geram modelos usando algoritmos de inteligência artificial, localmente, sem a necessidade de compartilhar dados com a nuvem [McMahan et al. 2017]. Cada dispositivo treina um modelo apenas com os dados que possui. Em seguida, envia o modelo gerado para um servidor de agregação, responsável por unificar os modelos por meio de algum algoritmo de consenso.

Dentre os principais problemas abordados pelos pesquisadores em aprendizado federado, tais como em [Chen et al. 2022] e [Hosseinzadeh et al. 2022], destaca-se: se todos os dispositivos gerarem e enviarem seus modelos locais, além de sobrecarregar a

rede, alguns dispositivos podem piorar o modelo global. Neste caso, deve-se propor algoritmos que selecionem um subconjunto de dispositivos capazes de melhorar o modelo, gerando economia na transmissão de dados.

Adicionalmente a estes problemas, as pesquisas que utilizam ambientes simulados ou emulados geralmente usam conjuntos de dados na literatura e simulam como estes dados são modelados por cada dispositivo, podendo considerar, ou não, a heterogeneidade dos dispositivos em relação à sua capacidade de processamento, memória e rede. Todos esses fatores podem afetar diretamente no tempo e qualidade dos modelos resultantes de uma aplicação de aprendizado federado [Tran et al. 2019].

Para permitir que os pesquisadores possam propor, experimentar e analisar algoritmos e novos modelos de aprendizado federado, este trabalho apresenta uma ferramenta chamada *MiniNetFed*. O *MiniNetFed* é baseado no conceito de contêineres e permite emular um ambiente para execução de experimentos de aprendizado federado, permitindo ao usuário que defina capacidade de processamento, memória e comunicação (largura de banda, perda e atraso) para grupos de dispositivos heterogêneos. Além disso, o *MiniNetFed* possui diversos algoritmos já implementados que, por meio de um arquivo de configuração, permite ao usuário definir: i) como um conjunto de dados será dividido entre os dispositivos; ii) o algoritmo de seleção de clientes; iii) função de agregação de modelos; e iv) modelos para os principais *datasets* utilizados na literatura.

Após configurado, o *MiniNetFed* executa os experimentos e retorna um conjunto de *logs* e visualizações gráficas das principais métricas para análises. O *MiniNetFed* foi projetado para ser extensível. Desta forma, pesquisadores podem desenvolver seus próprios algoritmos e compará-los com os já implementados. A ferramenta desenvolvida também tem uma proposta educacional, visto que permite a apresentação e uso prático dos principais conceitos e desafios na implementação do aprendizado federado.

2. Arquitetura do MiniNetFed

A emulação de dispositivos de redes é uma ferramenta importante para avaliação de desempenho de protocolos e tecnologias de redes com características mais reais do que as encontradas em simulação. Neste sentido, o Mininet[1] se tornou um dos emuladores mais populares entre pesquisadores por emular redes virtuais realísticas. Em [Peuster et al. 2016], foi proposta uma nova versão do Mininet, chamada ContainerNet, usada pelo *MiniNetFed*, que permite que cada dispositivo seja emulado como um contêiner *Docker*. Dessa forma é possível configurar parâmetros de hardware, como processamento e memória utilizada por cada dispositivo.

Uma visão geral da arquitetura do *MiniNetFed* é apresentada na Figura 1. Por meio de um único arquivo de configurações, o *MiniNetFed* define múltiplos aspectos de hardware dos contêineres que representarão os dispositivos que fazem parte do experimento. A arquitetura proposta consiste em pelo menos um servidor de agregação, um conjunto de clientes, e uma interface de comunicação, realizada por um *broker* MQTT (*Message Queuing Telemetry Transport*). A escolha por um *broker* permite que um modelo seja publicado pelo servidor e os clientes o recebam de forma assíncrona. Isto também difere o *MiniNetFed* de ferramentas comumente usadas para simular aprendizado federado, como

[1] https://mininet.org/

o *Flower*[2], que utiliza comunicação gRPC síncrona entre clientes e servidor.

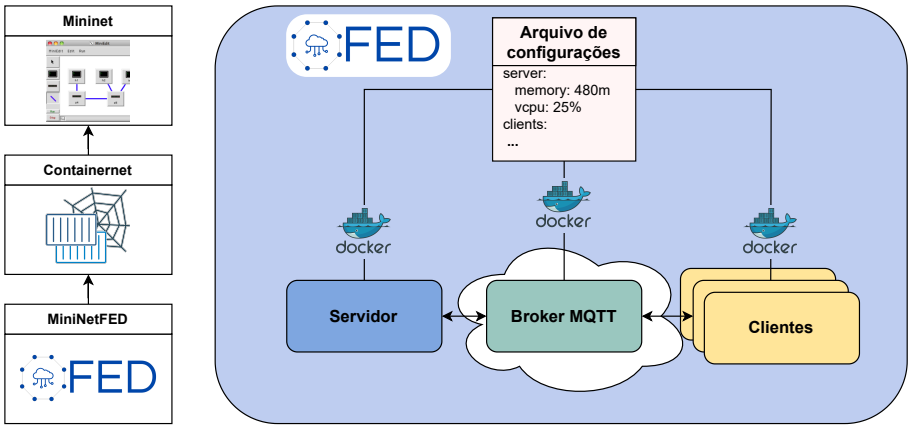


Figura 1. Visão Geral da Arquitetura do *MiniNetFed*.

O *MiniNetFed* permite configurar um ambiente de clientes com capacidades de hardware distintas por meio de um arquivo de configurações YAML. Neste arquivo são definidos: i) a quantidade total de vcpu utilizada pelo experimento, ii) o hardware do servidor, e iii) grupos de clientes. Para cada grupo é definido a quantidade de clientes, memória, vcpu, probabilidade de perda de pacotes e largura de banda. Adicionalmente, também são configurados os parâmetros globais, tais como: i) conjunto de dados que será utilizado, ii) os modelos que serão utilizados no treinamento, iii) configurações específicas de algoritmos de aprendizado profundo, e iv) imagens utilizadas em cada entidade da rede.

Ao iniciar a emulação de um experimento são instanciados: um contêiner para cada cliente conforme pré-definido nos grupos; um contêiner para o servidor; e um contêiner para o *broker* MQTT. O conjunto de dados é divido entre cada cliente baseado em uma configuração pré-definida (random, iid, por identificador). Foram definidos um conjunto de tópicos para disseminação de mensagens entre o servidor de agregação e os dispositivos clientes via interface MQTT. Foram definidos tópicos para *registro* de clientes, *seleção* de clientes que treinaram um modelo, *pré-agregação*, *pós-agregação*, anúncio de *métricas* e *parada* de treinamento, detalhados a seguir.

Para participar do processo de aprendizado federado, o cliente publica uma mensagem *registro*, enviando seu identificador único e aguarda o retorno com todos os parâmetros de configuração dos algoritmos usados no treinamento. Quando o número mínimo de clientes se registram, o processo de treinamento se inicia. Usando o algoritmo de seleção de clientes, definido no início do processo, o servidor de agregação determina quais clientes irão fazer parte do treinamento. Os clientes escolhidos treinam seus modelos locais, enquanto os não escolhidos entram em estado de espera.

Em seguida, os clientes ”treinadores”enviam um vetor com os pesos do modelo após o treinamento para servidor de agregação. O servidor usa algum algoritmo de agregação, também pré-definido, para agregar os pesos locais, assim compondo um modelo global. Este modelo global é publicado como uma mensagem de *pós-agregação* no *broker*. Após receber o modelo global, os clientes atualizam seus modelos locais com

[2]https://flower.ai/

o modelo global. Por fim, os clientes testam o modelo global com seus dados locais, calculando uma métrica de qualidade do modelo global. Esta métrica é publicada no tópico *métricas*, usada como condição de parada do processo, com o número máximo de rounds escolhido na configuração. Durante todo o experimento, o *MiniNetFed* gera um conjunto de *logs* que permite aos pesquisadores analisarem o processo. Adicionalmente, nesta versão já são geradas visualizações dos *logs* para as métricas comumente analisadas em aprendizado federado.

3. Avaliação de algoritmos de Aprendizado Federado com o MininetFed

O arquivo de configuração define como o Containernet irá instanciar os dispositivos, e o *MiniNetFed* também é responsável por controlar e orquestrar todo o experimento em aprendizado federado. A Figura 2 apresenta uma visão arquitetural dos algoritmos que podem ser implementados e executados. O experimentador pode definir (ou implementar um novo) algoritmos para a seleção de clientes, agregação do vetor de pesos e modelos de treinamento para cada cliente. Para isto, foram definidas classes padrões que devem ser especializados para cada proposta. O restante desta seção explica os algoritmos implementados e demonstra a flexibilidade da ferramenta para implementação novas propostas.

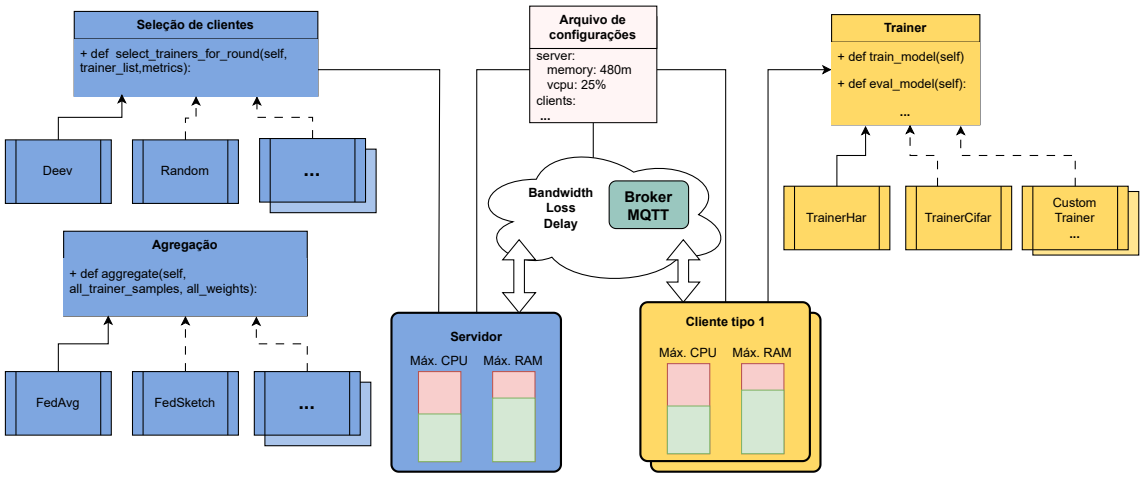


Figura 2. Algoritmos de aprendizado federado implementados no *MiniNetFed*.

3.1. Treinadores (*Trainers*)

No aprendizado federado, os clientes executam um algoritmo de aprendizado profundo, sob um conjunto de dados, visando resolver problemas de classificação ou predição de novos dados. Um experimentador precisa definir qual o conjunto de dados e quais técnicas de aprendizado profundo serão utilizados pelos dispositivos treinadores. No *MiniNetFed* isto é responsabilidade da classe *Trainer*.

Um *Trainer* é uma classe que define o conjunto de dados, como estes dados serão distribuídos entre os clientes, e qual algoritmo de treinamento de modelos sera executado nos clientes. O *Trainer* também atualiza os pesos recebidos do servidor de agregação e, após realizar o treinamento baseado nos dados da sua instância, gera as métricas de avaliação de desempenho. O *MiniNetFed* tem suporte para os principais conjuntos de dados utilizados para avaliação de algoritmos, sendo dois de classificação de imagens e dois de classificação de atividades humanas a partir de séries temporais:

MNIST: Conjunto composto por imagens de dígitos manuscritos monocromáticos de tamanho 28x28 pixeis.

CIFAR-10: Consiste em 60.000 imagens coloridas 32x32 pixeis em 10 classes, com 6.000 imagens por classe.

Human-Activity-Recognition (HAR): composto por cerca de 20 mil leituras de sensores de 6 participantes realizando 5 ações diferentes. Cada leitura consiste nas medidas acelerômetro, giroscópio e magnetômetro. [hum 2013].

Motion Sense: séries temporais coletadas de 24 participantes, realizando 6 atividades diferentes: subindo escadas, descendo escadas, andando, correndo, sentando e ficando em pé. [Malekzadeh et al. 2018].

A classe *Trainer* foi especializada para cada um desses conjuntos de dados. Para o treinamento, o *MiniNetFed* utiliza o pacote *TensorFlow*. Logo, qualquer arquitetura de aprendizado existente no TensorFlow pode ser utilizada para treinamento. A Tabela 1 sumariza as especializações de *Trainers* e configurações de arquiteturas utilizadas para treinamento nos clientes.

Tabela 1. Conjunto de dados e arquiteturas suportados (*Trainers*)

Trainer	**Conjunto de dados**	**Modelo de aprendizado**
TrainerMNIST	MNIST	Convolutional Neural Network. Seleção aleatória dos dados para cada cliente.
TrainerCIFAR	CIFAR-10	Variação da CNN anterior. Três modos de seleção de dados para cada cliente: aleatório, por classe, todos.
TrainerHar	HAR	MLP com 3 camadas ocultas e 256 unidades por camada. Três modos de seleção de dados: aleatório, por id, todos.
TraineMotionSense	MotionSense	Arquitetura *Time Series Transformer* para a classificação de séries temporais. Seleção de 50% dos clientes aleatórios.

3.2. Funções de políticas de seleção de clientes

No *MiniNetFed*, o servidor de agregação é responsável por implementar uma política de seleção de clientes, a qual define os clientes que irão participar da próxima rodada (*round*) de treinamento. Alguns algoritmos de seleção de clientes necessitam de informações e/ou métricas de cada cliente participante do treinamento. Para prover estas informações, a política de seleção deve indicar quais métricas devem ser publicadas após cada rodada. O servidor recebe estas informações e o resultado é a lista de clientes que serão "treinadores" na próxima rodada. O *MiniNetFed* implementa as seguintes políticas de seleção:

All: Seleciona todos os clientes. Utilizada como base de comparação nos experimentos.

Random: Seleciona aleatoriamente k clientes para participar da rodada.

FedSecPer: Calcula a acurácia de todos os clientes e seleciona todos que têm a acurácia abaixo da média [Souza et al. 2023]

DEEV: Variação do FedSecPer, mas aplica um fator de decaimento para selecionar uma porcentagem decrescente dos clientes [Souza et al. 2023].

Para o usuário do *MiniNetFed* implementar uma nova política basta que especialize a classe, tendo a liberdade para definir a política de seleção de clientes mais adequada ao experimento a ser emulado.

3.3. Funções de agregação

O *MiniNetFed* implementa o *FedAvg*, um algoritmo clássico de aprendizado federado que calcula uma média dos pesos ponderada pela quantidade de dados de treino de cada cliente. Outras funções de agregação podem ser implementadas no *MiniNetFed* a partir da Classe *FedAvg*.

Em um trabalho recente, foi proposto o uso de *Sketches*[3] para comprimir o vetor de pesos, diminuindo o volume de dados transmitidos no processo de aprendizado federado [Sarmento et al. 2024]. Para este caso, o *MiniNetFed* implementa o algoritmo de agregação *FedSketch*, que calcula uma média simples dos *sketches* das atualizações dos pesos pelo número de clientes.

3.4. Métricas Monitoradas pelo *MiniNetFed*

Durante a execução, o *MiniNetFed* registra uma série de métricas armazenadas em arquivos de *log*. Após a execução, os logs são processados para extração das seguintes métricas: i) acurácia global, por round; ii) perda (*loss*) por round; iii) duração de cada round; iv) quantidade de rounds; e v) quantidade de Bytes transmitidos por unidade de tempo. Estas métricas são salvas em um arquivo CSV, onde cada linha representa um round e cada coluna uma métrica.

Em seguida, um *script* para visualização de dados, personalizado para o experimento, gera os gráficos das métricas supracitadas. Por fim, os logs, os dados pré-processados para as métricas e os *scripts* geradores de gráficos são disponibilizados em uma pasta específica para cada experimento. Desta forma, os usuários podem personalizar a forma de visualização dos resultados, sem a necessidade de re-execução do experimento. Ainda é possível implementar outras métricas e enviá-las com as métricas já existentes. O pré-processamento dos dados irá tratar e inserir novas métricas automaticamente como uma coluna no arquivo CSV.

4. Casos de Uso do MininetFed

Foram propostos três casos de uso para demonstrar as funcionalidades do *MiniNetFed*.

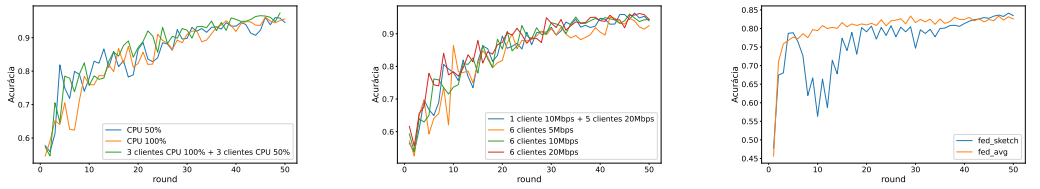
1. **Caso de Uso 1 - Dispositivos com processamento heterogêneo** - Avalia o impacto da heterogeneidade de capacidade de processamento dos dispositivos. Foram avaliados três cenários de aprendizado de atividade física (*TrainerHar*) com seis clientes: i) todos com 100% da capacidade de processamento; ii) todos com apenas 50% da capacidade de processamento; e iii) três clientes com 50% de capacidade de processamento e três com 100%. Foi utilizada a função de seleção *Random* e a função de agregação *FedAvg* em todos os cenários avaliados.
2. **Caso de Uso 2 - Dispositivos com largura de bandas heterogêneas** - Avalia o impacto da largura de banda dos dispositivos. Mesmas configurações do Caso de uso 1. Três cenários com largura de banda variando entre 5, 10 e 20 Mbps e um quarto cenário no qual 1 cliente possui 10 Mbps e os demais 20 Mbps.

[3] Sketches são estruturas de dados probabilísticas capazes de comprimir vetores de alta dimensionalidade para dimensões menores com garantias de erro de estimativa já bem estudadas.

3. **Caso de Uso 3 - Avaliando novos Algoritmos para aprendizado federado** - Neste experimento foi utilizado o algoritmo de aprendizagem *FedSketch* [Sarmento et al. 2024] com seleção de clientes aleatória e com o algoritmo *FedAvg*. No *FedSketch* as atualizações dos modelos são comprimidas em estruturas de dados probabilísticas (*sketches*) antes de serem enviadas para diminuir a quantidade de dados e, ao mesmo tempo, melhorar a privacidade do processo. Foi utilizado o *dataset* MotionSense, com 24 clientes.

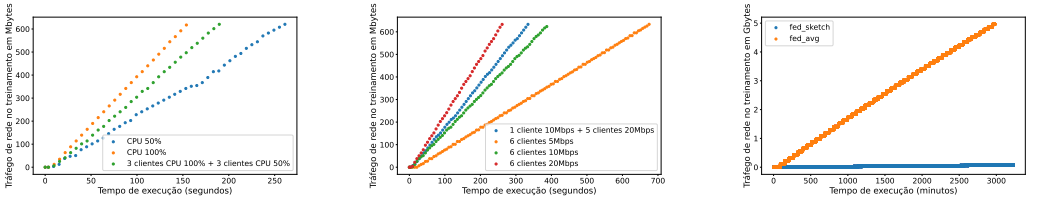
Para todos casos de uso, o *MiniNetFed* foi configurado para registrar a acurácia média e tráfego de dados ao longo do tempo do experimento. Ao final dos experimentos a ferramenta provê a visualização dos resultados e o *script* gerador, permitindo ao pesquisador personalizar *labels*, fontes e cores dos gráficos.

As Figuras 3 e 4 apresentam a acurácia média e o tráfego de dados para todos os casos de uso. Observa-se que a acurácia chega ao seu valor máximo com maior disponibilidade de CPU e largura de banda (Fig. 3(a) e Fig. 3(b)). Enquanto o tráfego de dados é igual para todos os cenários nos Casos de Uso 1 e 2 ao final do processo. Por outro lado, a proposta *Fedsketch* demora um tempo maior para convergir e alcançar a mesma acurácia do clássico *FedAVG* (Fig. 3(c), mas a compressão dos pesos em sketches levou a uma diminuição expressiva do tráfego de dados durante o processo (Fig. 4(c)).



(a) CPU (b) Largura de banda (c) Comparação algoritmos

Figura 3. Acurácia média para os três casos de uso.



(a) CPU (b) Largura de banda (c) Comparação algoritmos

Figura 4. Tráfego de dados (Mb) por tempo nos três casos de uso.

5. Divulgação e Documentação

Documentação e Github: `github.com/lprm-ufes/MininetFed`

Vídeo de Instalação: `youtu.be/G8hIIM3Xmr4`

Vídeo Tutorial de Utilização: `youtu.be/ZyunOvp50IQ`

6. Conclusões

Esse trabalho apresentou o *MiniNetFed*, uma ferramenta que se propõe a oferecer uma série de recursos de emulação de clientes e rede, além de oferecer uma interface para implementar algoritmos de seleção e agregação, modelos, e tratamento de dados em um contexto de aprendizado federado. Para avaliar as capacidades da ferramenta, foram implementados alguns casos de uso, funções de agregação e seleção de clientes e testados em experimentos realizados na própria ferramenta. Os resultados desses experimentos puderam ser coletados e analisados, mostrando que a ferramenta consegue cumprir o ciclo completo desde o desenvolvimento até a execução de diferentes cenários.

Agradecimentos

Este trabalho possui financiamento parcial de: CNPq, CAPES (Finance Code #001), Fapes (#2023/RWXSZ, #2022/ZQX6) e Fapesp/MCTI/CGI.br (#2020/ 05182-3 e #2023/00148-0).

Referências

(2013). Wearable computing: Classification of body postures and movements (puc-rio). `https://github.com/enceladus3/humanActivityRecognition`. Data Set Source: Pontifical Catholic University of Rio de Janeiro (PUC-Rio).

Chen, H., Huang, S., Zhang, D., Xiao, M., Skoglund, M., and Poor, H. V. (2022). Federated learning over wireless iot networks with optimized communication and resources. *IEEE Internet of Things Journal*, pages 1–1.

Hosseinzadeh, M., Hudson, N., Heshmati, S., and Khamfroush, H. (2022). Communication-loss trade-off in federated learning: A distributed client selection algorithm. In *IEEE Consumer Communications & Networking Conf.*, pages 1–6.

Malekzadeh, M., Clegg, R. G., Cavallaro, A., and Haddadi, H. (2018). Protecting sensory data against sensitive inferences. In *Privacy by Design in Distributed Systems*. ACM.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR.

Minerva, R., Biru, A., and Rotondi, D. (2015). Towards a definition of the internet of things (iot). *IEEE Internet Initiative*, 1(1):1–86.

Peuster, M., Karl, H., and van Rossem, S. (2016). Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *IEEE Conference on Network Function Virtualization and Software Defined Networks*, pages 148–153.

Sarmento, E. M. M. et al. (2024). Privacidade e comunicação eficiente em aprendizado federado: Uma abordagem utilizando estruturas de dados probabilísticas e seleção de clientes. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Souza, A. et al. (2023). Dispositivos, eu escolho vocês: Seleção de clientes adaptativa para comunicação eficiente em aprendizado federado. In *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 1–14. SBC.

Tran, N. H., Bao, W., Zomaya, A., Nguyen, M. N. H., and Hong, C. S. (2019). Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1387–1395.