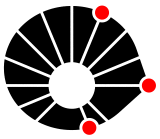

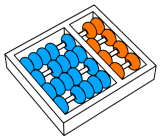

Otimização Energética em Aprendizado Federado via Computação Colaborativa

C. P. Libardi *L. L. Favacho* *R. G. de Souza*
L. F. Bittencourt *F. M. Roberto*

Relatório Técnico - IC-PFG-25-39
Projeto Final de Graduação
2025 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Otimização Energética em Aprendizado Federado via Computação Colaborativa

Cicero Pizzol Libardi Leonardo Luca Favacho Rafael Gregori de Souza
Luiz Fernando Bittencourt Filipe Maciel Roberto

Resumo

O Aprendizado Federado (Federated Learning - FL) consiste em uma abordagem de aprendizado de máquina descentralizada que garante a privacidade dos dados, permitindo o treinamento de modelos de Inteligência Artificial sem que os dados brutos precisem sair dos dispositivos dos usuários. No entanto, a implementação do FL em ambientes móveis enfrenta desafios críticos, majoritariamente devido ao alto consumo de energia e a heterogeneidade dos dispositivos participantes do treinamento. As diferenças significativas na capacidade computacional, memória e níveis de bateria entre os participantes podem levar a disparidades de desempenho, longos tempos de treinamento e até mesmo à desistência de cliente, comprometendo a qualidade final e a convergência do modelo.

Para endereçar essas limitações, este trabalho apresenta a implementação do EAFL+, um sistema de otimização de energia projetado para gerenciar o consumo de bateria em dispositivos com recursos limitados. A abordagem avaliada é baseada na computação colaborativa, integrando recursos da nuvem, da borda e dos próprios dispositivos terminais. Utilizando o framework de simulação Flower, o sistema seleciona, de forma eficiente, um alvo para onde as tarefas computacionais mais pesadas do treinamento podem ser realocadas. Esta metodologia busca o equilíbrio ideal entre custo e qualidade.

A avaliação do EAFL+, realizada em um ambiente de FL emulado com datasets reais, visa demonstrar sua eficácia em conservar a energia dos dispositivos, resultando em um aumento nas taxas de participação e contribuição dos clientes. Consequentemente, espera-se que a abordagem leve a uma melhoria na acurácia do modelo e a uma convergência mais rápida, sendo capaz de mitigar a desistência de clientes. Comparado aos métodos existentes, como o EAFL original e o Oort, o trabalho busca consolidar o EAFL+ como solução robusta para viabilizar o Aprendizado Federado em cenários reais e heterogêneos.

1 Introdução

O aprendizado de máquina consolidou-se como uma das áreas mais impactantes da inteligência artificial, possibilitando tomar decisões mais fundamentadas em áreas como finanças, transporte e saúde. Nesse contexto, o Aprendizado Federado surge como uma metodologia que permite o treinamento colaborativo de modelos, priorizando a privacidade

dos dados (MCMAHAN et al., 2017). Dessa forma, cada dispositivo participa do treinamento de forma independente, o que pressupõe desafios técnicos como a heterogeneidade entre os participantes, a instabilidade de conexões de rede e o desbalanceamento de carga entre clientes.

Diante desses empecilhos, estratégias de seleção de participantes ganharam destaque. O algoritmo Oort (LAI et al., 2021), precursor nesse contexto, aplica políticas de amostragem baseadas em utilidade para equilibrar o desempenho do treinamento e o custo de comunicação. Embora eficiente em reduzir o tempo de convergência, o Oort não foi projetado para lidar com restrições críticas de bateria, o que pode levar à exaustão prematura de dispositivos em cenários de IoT. Para preencher essa lacuna, foi proposto o EAFL, uma abordagem derivada do Oort que introduz mecanismos de consciência energética e de borda, visando reduzir latências e garantir a longevidade dos dispositivos durante o processo.

Neste trabalho, é avaliada a implementação do EAFL+, um novo sistema que aprimora os conceitos estabelecidos pelo EAFL (e, consequentemente, pelo Oort) ao incorporar estratégias adicionais de seleção, balanceamento de carga e priorização. A inovação central do EAFL+ emerge da sua arquitetura de computação colaborativa em três camadas (nuvem-borda-terminal) que implementa o descarregamento de computação (AROUJ; ABDELMONIEM, 2024). Em vez de descartar um dispositivo com bateria fraca, o EAFL+ permite que este delegue suas tarefas de treinamento para um alvo quase ideal — seja um servidor de borda, a nuvem ou outro dispositivo par. Ao fazer isso, o sistema busca conservar a energia dos participantes, eliminar as desistências e, consequentemente, melhorar a acurácia, a velocidade de convergência e a justiça do modelo federado.

1.1 Objetivos

O objetivo geral deste trabalho é realizar a implementação do algoritmo EAFL+ (Energy-Aware Federated Learning Plus) utilizando o framework de simulação Flower (BEUTEL et al., 2020). O projeto visa validar a eficácia do sistema na redução do consumo de energia e, então, propor e avaliar alterações que otimizem ainda mais o desempenho em cenários de IoT.

Os objetivos específicos incluem:

- Estudar os conceitos de computação em nuvem e borda (Mobile Edge Computing - MEC) aplicados ao Aprendizado Federado.

- Dominar o uso da ferramenta de simulação Flower para orquestração de cenários de aprendizado federado.

- Utilizar como base uma implementação existente do algoritmo Oort no ambiente Flower e estendê-la para contemplar as funcionalidades do EAFL+ (offloading e colaboração).

- Analisar o desempenho comparativo (acurácia e energia) para alguns hiperparâmetros de execução, discutidos posteriormente.

2 Fundamentação Teórica

Este capítulo estabelece as bases conceituais necessárias para a compreensão do algoritmo a ser avaliado, abordando o funcionamento do Aprendizado Federado, a integração com a Computação em Borda e os desafios inerentes aos ambientes móveis heterogêneos.

2.1 Aprendizado Federado (Federated Learning)

O Aprendizado Federado (FL) é um paradigma de aprendizado de máquina distribuído que permite que múltiplos dispositivos (como smartphones e dispositivos IoT) treinem um modelo global colaborativo sem compartilhar seus dados brutos. Em uma configuração típica de FL, um servidor central (agregador) coordena o processo. A cada rodada de treinamento, o servidor seleciona um subconjunto de clientes disponíveis e envia a eles o modelo global atual.

Cada cliente selecionado realiza o treinamento localmente usando seus próprios dados privados, computando atualizações de gradiente (*Stochastic Gradient Descent* - SGD). Após o cálculo, apenas as atualizações dos pesos do modelo são enviadas de volta ao servidor, onde são agregadas (geralmente através da média, como no algoritmo FedAvg (MCMAHAN et al., 2017)) para formar um novo modelo global. Esse ciclo se repete até que o modelo atinja a acurácia desejada.

As principais vantagens do FL incluem a privacidade dos dados, já que as informações sensíveis nunca saem do dispositivo do usuário, e a eficiência de comunicação, pois apenas os parâmetros do modelo são transmitidos, economizando largura de banda em comparação ao envio de grandes volumes de dados brutos para a nuvem.

2.2 Computação em Borda e Nuvem Móvel

Com o crescimento exponencial de dados gerados por dispositivos móveis, surgiu o conceito de *Mobile Cloud Computing* (MCC), que aloca recursos de computação sob demanda de infraestruturas de nuvem centralizadas (LIM et al., 2020). Embora o MCC ofereça grande poder de processamento para aplicações pesadas, ele enfrenta problemas de alta latência devido à distância física entre os dispositivos e os datacenters remotos.

Para mitigar a latência, a Computação em Borda Móvel (*Mobile Edge Computing* - MEC) aproxima os serviços de nuvem da borda da rede, localizando servidores nas estações base celulares ou pontos de acesso próximos aos usuários (LIM et al., 2020). O MEC oferece latência ultra-baixa e reduz o tráfego na rede principal (*core network*), permitindo que dispositivos móveis descarreguem (*offload*) tarefas computacionais pesadas para a borda. Assim, enquanto a nuvem mitiga as limitações de armazenamento e processamento dos dispositivos terminais, a borda aproxima a funcionalidade do usuário, resolvendo problemas de atraso e segurança. Essa arquitetura é fundamental para o EAFL+, que explora a diversidade de recursos disponíveis nessas camadas.

2.3 Desafios em Ambientes Móveis: Heterogeneidade e Energia

A implementação prática do FL em redes móveis enfrenta o desafio crítico da heterogeneidade de dispositivos. Os participantes variam drasticamente em termos de:

- **Capacidade de Computação:** Diferenças em CPU/GPU afetam o tempo que cada cliente leva para treinar o modelo local (LI et al., 2020).
- **Conectividade:** Variações em largura de banda (Wi-Fi, 4G, 5G) impactam o tempo de transmissão dos modelos (LI et al., 2020).
- **Recursos de Energia:** A restrição de bateria é talvez o maior gargalo.

O treinamento de modelos de *Deep Learning* é intensivo e consome muita energia (LIM et al., 2020). Dispositivos com bateria fraca ou hardware limitado podem não conseguir completar o treinamento a tempo ou desligar no meio do processo. Isso gera o problema dos *Dropouts* (desistências). Ao contrário dos *stragglers* (clientes lentos que apenas atrasam a rodada), os *dropouts* são incapazes de enviar suas atualizações, o que pode enviesar o modelo global e desperdiçar os recursos de rede já utilizados. A maioria das abordagens atuais ignora a otimização de energia ou assume que os dispositivos estão conectados à tomada, o que não é realista para IoT e smartphones.

2.4 Trabalhos Relacionados

Diversas estratégias foram propostas para otimizar o FL. O algoritmo Oort (LAI et al., 2021), por exemplo, foca na eficiência do sistema, selecionando clientes com base em uma utilidade que equilibra a qualidade dos dados e a velocidade do sistema. No entanto, o Oort tende a negligenciar o consumo de energia, o que pode levar à exaustão rápida da bateria dos dispositivos selecionados.

Já o EAFL original propôs uma abordagem consciente de energia, priorizando a seleção de clientes com maior carga de bateria restante para garantir a conclusão das tarefas. Contudo, embora o EAFL evite selecionar dispositivos fracos, ele não atua na redução do consumo de energia dos dispositivos selecionados, nem oferece mecanismos para auxiliar aqueles com poucos recursos a participarem, limitando a diversidade de dados do modelo.

3 O Sistema EAFL+

O EAFL+ (*Energy-Aware Federated Learning Plus*) é a solução estudada neste trabalho para superar as limitações de energia e heterogeneidade discutidas anteriormente. Ele é projetado como um sistema colaborativo que integra as camadas de Nuvem, Borda e Terminal (dispositivos) para otimizar o consumo energético e viabilizar a participação de dispositivos com recursos limitados.

3.1 Arquitetura Colaborativa Nuvem-Borda-Terminal

A inovação central do EAFL+ reside na sua arquitetura de três camadas. Em vez de processar todo o treinamento localmente no dispositivo (como no FL padrão) ou enviar tudo para a nuvem (centralizado), o EAFL+ explora a diversidade de recursos disponíveis no ambiente (AROUJ; ABDELMONIEM, 2024):

- **Camada Terminal (Clientes):** Dispositivos IoT e smartphones que possuem os dados. Eles podem colaborar entre si via conexões peer-to-peer (P2P) de baixa latência.
- **Camada de Borda (Edge):** Servidores MEC próximos aos usuários, que oferecem poder computacional moderado com baixa latência.
- **Camada de Nuvem (Cloud):** Servidores remotos com recursos computacionais praticamente ilimitados, mas com maior custo de latência de comunicação.

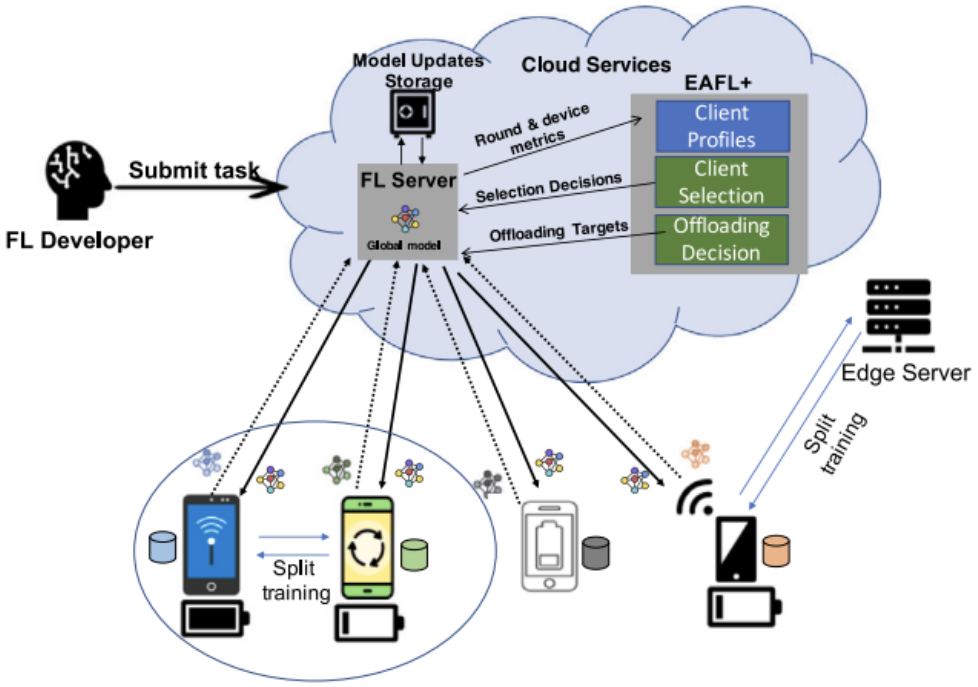

Figura 1: Arquitetura do sistema EAFL+ com computação colaborativa Nuvem-Borda-Terminal.

3.2 Mecanismo de Offloading e Split Learning

Para evitar o esgotamento da bateria e os *dropouts*, o EAFL+ utiliza a técnica de *Split Learning* (Aprendizado Dividido) (VEPAKOMMA et al., 2018). Quando um dispositivo

selecionado para treinamento possui pouca bateria ou baixa capacidade de processamento ("cliente fraco"), o sistema pode decidir "quebrar"o modelo de rede neural em duas partes. A primeira parte (geralmente as camadas iniciais) é processada localmente pelo dispositivo. A segunda parte (camadas mais pesadas) é descarregada (*offloaded*) para um "alvo de offloading".

O sistema identifica e seleciona o alvo de offloading ideal entre três opções:

- **Vizinhos Próximos:** Outros dispositivos na mesma rede local que possuam bateria suficiente e possam ajudar.
- **Servidor de Borda:** Se não houver vizinhos, a tarefa é enviada para a borda.
- **Servidor de Nuvem:** Como último recurso, utiliza-se a nuvem.

3.3 Modelo de Decisão

A decisão de para onde descarregar a computação é tomada com base em um *trade-off* entre custo (energia/tempo) e qualidade. O EAFL+ realiza um perfilamento online (*online profiling*) dos clientes, monitorando níveis de bateria, capacidade de RAM e condições de rede. Com esses dados, o algoritmo busca minimizar o consumo de energia do dispositivo fraco, garantindo ao mesmo tempo que o tempo total de treinamento e comunicação não exceda os limites que prejudicariam a eficiência do sistema global. Dessa forma, o EAFL+ consegue manter a taxa de participação alta (reduzindo *dropouts* a zero em experimentos) e aumentar a acurácia final do modelo ao incluir dados de dispositivos que, de outra forma, seriam descartados pelo sistema.

3.4 Estratégia de Implementação

A metodologia de desenvolvimento deste trabalho segue uma abordagem incremental baseada na reutilização e extensão de código no ambiente Flower:

- **Validação do Cenário Base (Oort):** Inicialmente, será instanciado o ambiente de simulação no Flower utilizando uma implementação pré-existente do algoritmo Oort. Esta etapa serve para estabelecer uma linha de base de desempenho e validar o funcionamento correto da comunicação federada.
- **Implementação do EAFL+:** O código base será refatorado para incluir os módulos de decisão de *offloading* e colaboração Nuvem-Borda-Terminal, características centrais do EAFL+.

3.5 Algoritmo

A implementação do algoritmo EAFL+, uma evolução do EAFL, fundamenta-se na modificação do critério de seleção de participantes e na gestão de recursos. Em contraste com a abordagem aleatória, o algoritmo proposto atua como um orquestrador que avalia métricas históricas e o estado instantâneo da bateria.

A lógica do algoritmo divide-se em componentes fundamentais: o cálculo do peso, o mecanismo de amostragem (explotação e exploração) e o controle dinâmico de ritmo. Abaixo, detalha-se a implementação de cada etapa.

3.5.1 Cálculo de Pesos (W_k)

Na etapa de configuração da rodada, o servidor atribui a cada cliente candidato (k) um peso de seleção. Esta métrica é uma adaptação heurística do modelo de otimização proposto em (AROUJ; ABDELMONIEM, 2024), combinando a utilidade estatística normalizada e a razão de bateria, ponderadas pelo hiperparâmetro γ (`eafl_weight`). O código abaixo demonstra a implementação da função `calc_eafl_weight`:

```
def calc_eafl_weight(self, cid):
        if not (0 <= self.eafl_weight <= 1):
            raise ValueError("w deve estar entre 0 e 1")

        # Protege contra lista vazia
        if len(self.client_utils) == 0:
            max_U = 1.0
            min_U = 0.0
        else:
            max_U = max(self.client_utils)
            min_U = min(self.client_utils)
            max_U = max_U if max_U != 0 else 1.0

        # Protege para utility do cliente
        client_utility = self.profiles[cid].get('utility', 0.0)

        # Normalize client utility
        normalized_utility = (client_utility - min_U) / \
            (max_U - min_U) if max_U > min_U else 0.0

        # Calcula peso EAFL
        if self.use_battery:
            battery_ratio = (self.profiles[cid]["initial_battery_mJ"] /
                             self.profiles[cid]['max_battery_mJ'])
            return self.eafl_weight * battery_ratio + (1 - self.eafl_weight)
    * normalized_utility
        else:
            # Se nao usar bateria, retornar apenas a utilidade normalizada
            return normalized_utility
```

Listing 1: Cálculo do peso de seleção EAFL+

3.5.2 Seleção: Explotação (Exploitation)

A seleção final utiliza um mecanismo balanceado. Na fase de *exploitation*, o algoritmo ordena os clientes pelo peso W_k. Aplica-se um filtro de corte (*cut-off*) onde clientes com peso muito inferior são descartados. A escolha final entre os candidatos aptos é feita via amostragem probabilística proporcional a W_k.

```python
available_cids.sort(key=lambda x: x[1], reverse=True)

sorted_by_eafl_weights = [cid for cid, eafl_weight in available_cids]

# Calcula a utilidade de corte
cut_off_util = (
    self.profiles[sorted_by_eafl_weights[exploited_clients_count - 1]]['eafl_weight'] * self.cut_off
)

# Inclui clientes com utilidade maior que o corte
exploited_clients = []
for client_id in sorted_by_eafl_weights:
    if (
        self.profiles[client_id]['eafl_weight'] > cut_off_util
        and client_id not in self.blacklist
    ):
        exploited_clients.append(client_id)

# Amostra clientes com base em suas utilidades
total_eafl_weights = float(
    sum(self.profiles[client_id]['eafl_weight'] for client_id in exploited_clients)
)

probabilities = [
    self.profiles[client_id]['eafl_weight'] / total_eafl_weights
    for client_id in exploited_clients
]

if len(probabilities) > 0 and exploited_clients_count > 0:
    selected_clients = np.random.choice(
        exploited_clients,
        min(len(exploited_clients), exploited_clients_count),
        p=probabilities,
        replace=False,
    )
    selected_clients = selected_clients.tolist()

last_index = (
    sorted_by_eafl_weights.index(exploited_clients[-1])
    if exploited_clients
    else 0
)
```

Listing 2: Fase de Explotação na Seleção

3.5.3 Seleção: Exploração (Exploration)

Paralelamente, na fase de *exploration*, uma fração dos slots é reservada para clientes inéditos (que nunca participaram), garantindo que o algoritmo descubra novos dados e evite ficar preso a um ótimo local de dispositivos conhecidos.

```python
# Seleciona clientes inexplorados aleatoriamente
unexplored_size = len(self.unexplored_clients)
if unexplored_size > 0:
    selected_unexplore_clients = random.sample(
        self.unexplored_clients,
        min(unexplored_size, sample_size - len(selected_clients))
    )
else:
    selected_unexplore_clients = []

# Atualiza listas de controle
self.explored_clients += selected_unexplore_clients
for client_id in selected_unexplore_clients:
    self.unexplored_clients.remove(client_id)

selected_clients += selected_unexplore_clients
```

Listing 3: Fase de Exploração na Seleção

3.5.4 Decisão de Offloading e Infraestrutura

O sistema identifica a infraestrutura disponível, mapeando quais clientes possuem acesso a servidores de borda para realizar o *offloading* computacional. Embora a execução do offloading ocorra no cliente, o servidor mantém o estado da topologia (Edges e Clientes) durante a inicialização para coordenar as rodadas.

```python
# Distribui Servidores de Borda entre os clientes
self.edge_servers = [True] * self.num_edge_servers + [False] * (self.
    num_clients - self.num_edge_servers)
random.shuffle(self.edge_servers)

# Atualiza perfil dos clientes
for cid, profile in self.profiles.items():
    self.profiles[cid]['has_edge'] = self.edge_servers[int(cid)]
```

Listing 4: Inicialização da Topologia de Offloading

3.5.5 Controle Dinâmico de Ritmo (Pacer)

Para lidar com a heterogeneidade, aplica-se um controle dinâmico de ritmo (*Pacer*). O algoritmo monitora a utilidade agregada do sistema em janelas de tempo (`step_window`). Se a utilidade estagnar ou cair, o sistema relaxa a restrição de tempo (`desired_duration`), permitindo a participação de dispositivos mais lentos.

```python
# Armazena hist rico de utilidade
self.util_history.append(round_utility)

if server_round >= 2 * self.step_window:
    # Compara a soma da utilidade da janela anterior com a atual
    last_pacer_rounds = sum(
        self.util_history[-2 * self.step_window: -self.step_window]
    )
```

```
    current_pacer_rounds = sum(self.util_history[-self.step_window:])

    # Se a utilidade caiu, aumenta o tempo limite (relaxa o sistema)
    if last_pacer_rounds > current_pacer_rounds:
        self.desired_duration += self.pacer_step
```

Listing 5: Controle Dinâmico de Ritmo (Pacer)

4 Metodologia

Neste trabalho, o framework Flower (BEUTEL et al., 2020) foi utilizado devido à sua capacidade de gerenciamento da comunicação entre servidor e clientes em um ambiente federado simulado de forma escalável, sem a necessidade de possuir dispositivos físicos. Essa ferramenta suporta heterogeneidade de sistemas e integração com bibliotecas populares de ML como PyTorch e TensorFlow. Sua arquitetura modular facilita a implementação de estratégias customizadas de agregação e seleção de clientes, sendo ideal para testar as abordagens de offloading a serem avaliadas neste trabalho.

4.1 Dataset

Os experimentos utilizam o dataset CIFAR-10 (KRIZHEVSKY; HINTON et al., 2009), composto por 60.000 imagens coloridas de 32x32 pixels, divididas em 10 classes. O conjunto é separado em 50.000 imagens para treinamento e 10.000 para teste.

Para simular a natureza Non-IID (Não Independente e Identicamente Distribuída) típica de cenários federados, a partição dos dados entre os clientes segue uma Distribuição de Dirichlet Latente Dir(α). O hiperparâmetro α regula a heterogeneidade dos clientes. Quanto mais próximo de 0, mais heterogêneo é o cenário testado. Assim, os valores usados estão descritos a seguir:

- **Heterogeneidade Extrema ($\alpha = 0.1$):** simula um cenário onde cada cliente possui dados de pouquíssimas classes, criando um viés estatístico severo.
- **Heterogeneidade Moderada ($\alpha = 1.0$):** os dados são distribuídos de forma mais equilibrada, embora ainda não uniforme, facilitando a convergência do modelo global.

4.2 Modelo

Foi utilizado o *ShuffleNet v2 (0.5x)* (MA et al., 2018), um perfil de modelo de aprendizado profundo para representar diferentes complexidades computacionais adequadas para dispositivos móveis. Este perfil oferece um equilíbrio entre acurácia e custo computacional.

4.3 Simulação de hardware

A heterogeneidade de hardware é modelada através de perfis de dispositivos gerados estaticamente. Cada cliente federado possui uma capacidade de bateria, um valor finito de

energia que decresce a cada rodada de treinamento local se o dispositivo for selecionado. Essa bateria é usada para treinamento e comunicação do dispositivo.

Nos cenários de teste, utilizou-se um parâmetro para ativar o módulo de clientes críticos. Caso a bateria de um cliente selecionado seja insuficiente para completar o treinamento local e a comunicação com o servidor, o cliente é considerado como inoperante (dropout), não contribuindo para a agregação global naquela rodada.

4.4 Experimentos

Os testes foram realizados em duas situações diferentes de acordo com a estratégia de seleção de participantes. No primeiro cenário, utiliza-se uma estratégia de seleção aleatória (*random*). O objetivo é estabelecer um ponto de comparação padrão, onde o servidor escolhe participantes sem considerar suas restrições energéticas ou latência. Embora o consumo de bateria seja contabilizado para fins de monitoramento e registro de falhas (*dropouts*), ele não influencia a decisão de seleção.

No cenário experimental, onde foi utilizado o EAFL+, avaliou-se a capacidade do sistema de realizar o *offloading* computacional para servidores de borda (*Edge Servers*) e selecionar clientes com base em sua aptidão energética.

4.5 Parâmetros

Para a análise das simulações, foram utilizados os parâmetros conforme a tabela 1, com o objetivo de comparar a performance de ambos os algoritmos.

Tabela 1: Parâmetros da Simulação

Parâmetro	**Valor / Configuração**	**Descrição**
Rodadas	150	Número total de ciclos de agregação (num-rounds).
Semente (Seed)	42	Garante a reprodutibilidade da inicialização dos pesos e distribuição dos dados.
Estratégia de Seleção	random vs eaflplus	Variável independente principal do estudo.
Pesos EAFL	{0.25, 0.75}	Peso dado ao componente de energia na seleção (eafl-weight).
Servidores de Borda	{25, 50, 100}	Variação da densidade da infraestrutura (num-edge-servers).

5 Resultados

Nesta seção, são apresentados os resultados obtidos através das simulações, avaliando o desempenho do algoritmo EAFL+ sob diferentes configurações de infraestrutura e parâmetros de seleção. A análise inicial foca na eficiência energética, um dos pilares do sistema proposto, monitorando a quantidade acumulada de clientes que esgotaram totalmente suas

baterias e se tornaram inoperantes (*dropouts*) ao longo das rodadas de treinamento. A Figura 2 apresenta as curvas de esgotamento para diferentes combinações de peso energético (w) e densidade de servidores de borda (*edges*), onde destacamos o comportamento das curvas marrom, vermelha e verde, que ilustram de forma coerente as premissas teóricas do descarregamento computacional.

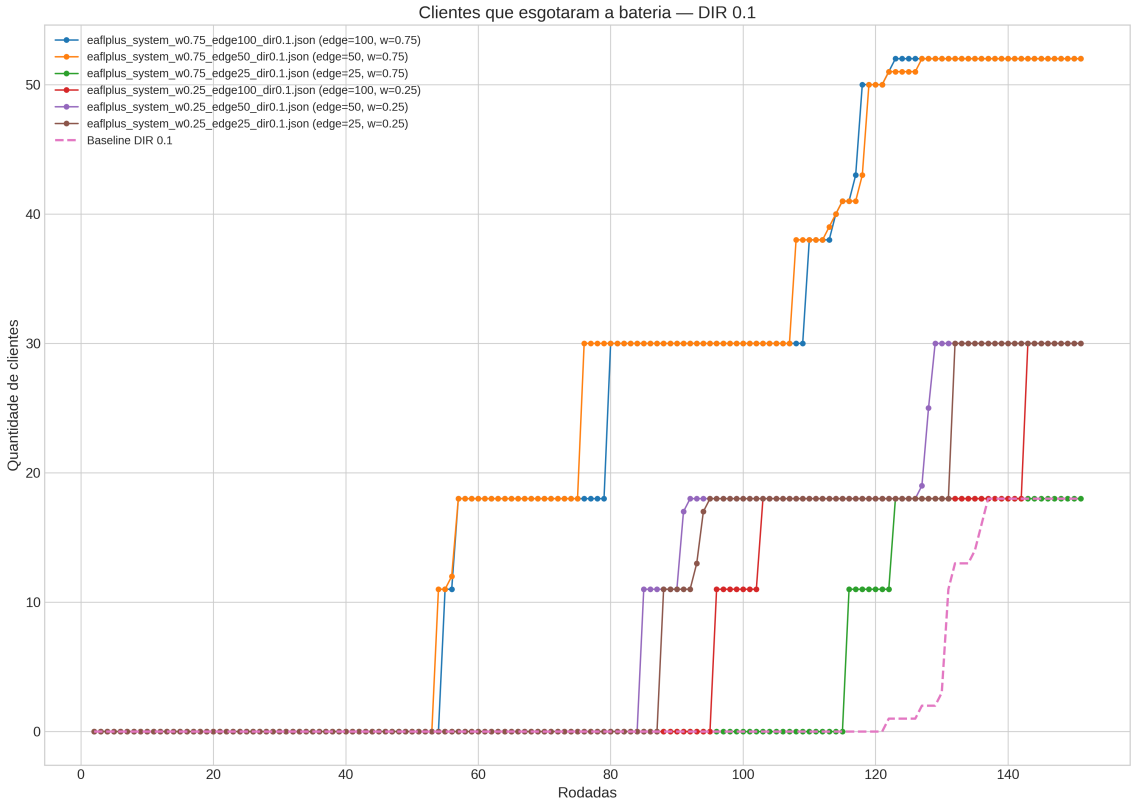

Figura 2: Quantidade acumulada de clientes com bateria esgotada ao longo das rodadas.

Ao fixar o peso de seleção de energia em um valor moderado ($w = 0.25$), é possível isolar o impacto da infraestrutura comparando a curva marrom (25 servidores) com a curva vermelha (100 servidores). Como esperado, a configuração com mais servidores (curva vermelha) resultou em um retardo significativo no esgotamento das baterias. Enquanto o cenário marrom apresentou falhas mais cedo, o cenário vermelho conseguiu manter os dispositivos ativos por mais tempo, pois a maior densidade de servidores de borda aumenta a probabilidade de sucesso no *offloading*, poupando a bateria local dos clientes.

Além da infraestrutura, a influência do algoritmo de seleção é evidenciada na curva verde ($w = 0.75$, 25 servidores). Mesmo operando com a mesma infraestrutura limitada da curva marrom, a curva verde obteve o melhor desempenho de conservação entre os cenários analisados. Isso ocorre porque o aumento do peso w torna o critério de seleção mais rigoroso, priorizando agressivamente dispositivos com maior carga restante. Dessa forma, demonstra-se que a inteligência na seleção pode compensar a escassez de recursos físicos, evitando a

exaustão prematura e garantindo a perenidade da rede federada.

Esse comportamento de conservação é corroborado pela análise dos clientes que atingiram o limiar crítico de energia (definido como 10% de carga restante). A Figura 3 ilustra o momento em que os dispositivos cruzam essa linha de segurança, antecedendo a falha total.

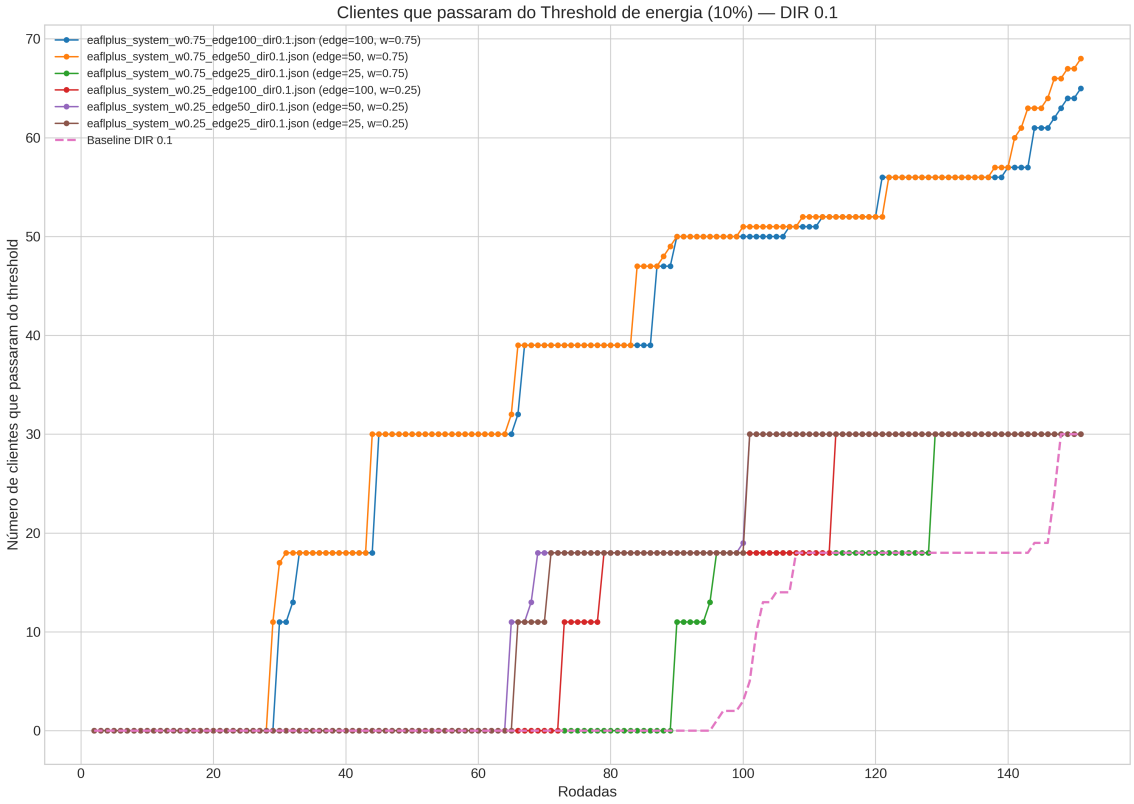

Figura 3: Quantidade acumulada de clientes que ultrapassaram o limiar crítico de energia (10%).

As tendências observadas no esgotamento repetem-se consistentemente na entrada em estado crítico. Comparando-se novamente as curvas com peso fixo ($w = 0.25$), a curva vermelha (100 servidores) apresenta um atraso significativo no aumento de clientes críticos em relação à curva marrom (25 servidores). Isso confirma que a maior disponibilidade de alvos para *offloading* não apenas evita a morte do dispositivo, mas desacelera a degradação da bateria muito antes do fim, mantendo os dispositivos em zonas seguras de operação por mais tempo.

De modo análogo, a eficácia da política de seleção é validada pela curva verde ($w = 0.75$, 25 servidores). Mesmo operando com a infraestrutura limitada (idêntica à do cenário marrom), esta configuração foi capaz de adiar a entrada dos clientes no estado crítico de forma mais eficiente do que as demais. Isso indica que um peso w elevado atua preventivamente: ao penalizar a escolha de dispositivos que já consumiram parte de sua energia, o algoritmo distribui o desgaste de forma mais equilibrada, evitando que clientes específicos sejam dre-

nados até o limiar de 10% precocemente.

Além da análise individual da saúde das baterias, é fundamental avaliar a eficiência do sistema sob uma perspectiva macroscópica, contabilizando a energia total consumida por todos os participantes (selecionados e não selecionados) ao longo do treinamento. A Figura 4 ilustra esse consumo acumulado, permitindo observar o custo energético global da operação federada.

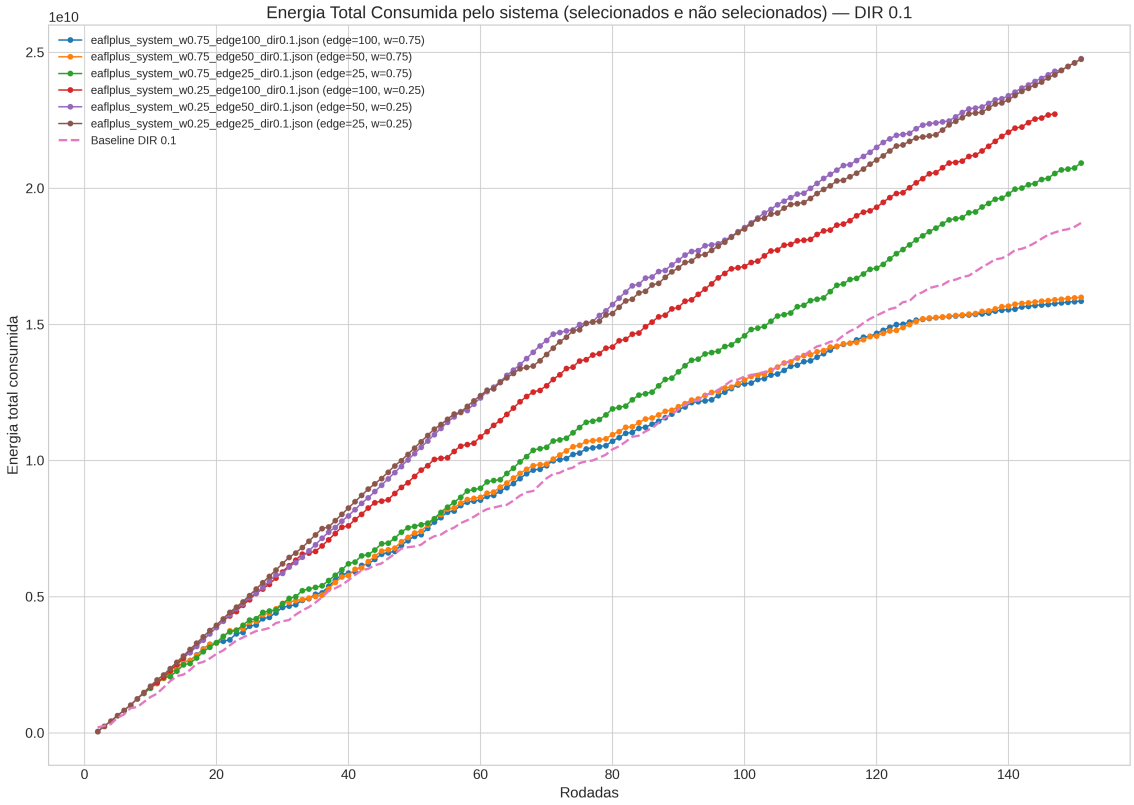

Figura 4: Energia total consumida pelo sistema ao longo das rodadas.

A coerência observada nas métricas de bateria mantém-se na análise de consumo global entre as curvas marrom, vermelha e verde. A configuração marrom ($w = 0.25$, 25 servidores) apresenta a curva de crescimento mais acentuada, indicando o maior gasto energético entre as três. A introdução de mais infraestrutura na configuração vermelha ($w = 0.25$, 100 servidores) reduz a inclinação da curva, comprovando que o *offloading* diminui o custo computacional agregado. Já a curva verde ($w = 0.75$, 25 servidores) demonstra que a priorização algorítmica de dispositivos aptos atua de forma ainda mais eficaz na contenção do gasto energético total.

Entretanto, ao analisar o desempenho interno das variantes do EAFL+, os melhores resultados são observados nas curvas azul ($w = 0.75$, 100 servidores) e laranja ($w = 0.75$, 50 servidores). Estes cenários ilustram a sinergia ideal entre infraestrutura e algoritmo: ao combinar uma alta disponibilidade de servidores de borda (50 ou 100) com um peso de

seleção elevado ($w = 0.75$), o sistema maximiza o uso do *offloading* — reduzindo drasticamente o custo local — enquanto evita desperdícios com clientes ineficientes. O resultado é um sistema que atinge seus objetivos de aprendizado consumindo a menor quantidade de energia acumulada possível.

Paralelamente à eficiência energética, avaliou-se o impacto da estratégia proposta na qualidade do modelo global. A Figura 5 apresenta a curva de acurácia ao longo do treinamento, comparando o baseline (seleção aleatória) com dois cenários distintos do EAFL+: um cenário de menor capacidade ("fraco", com $w = 0.25$ e 25 servidores) e um cenário de alta capacidade ("forte", com $w = 0.75$ e 100 servidores).

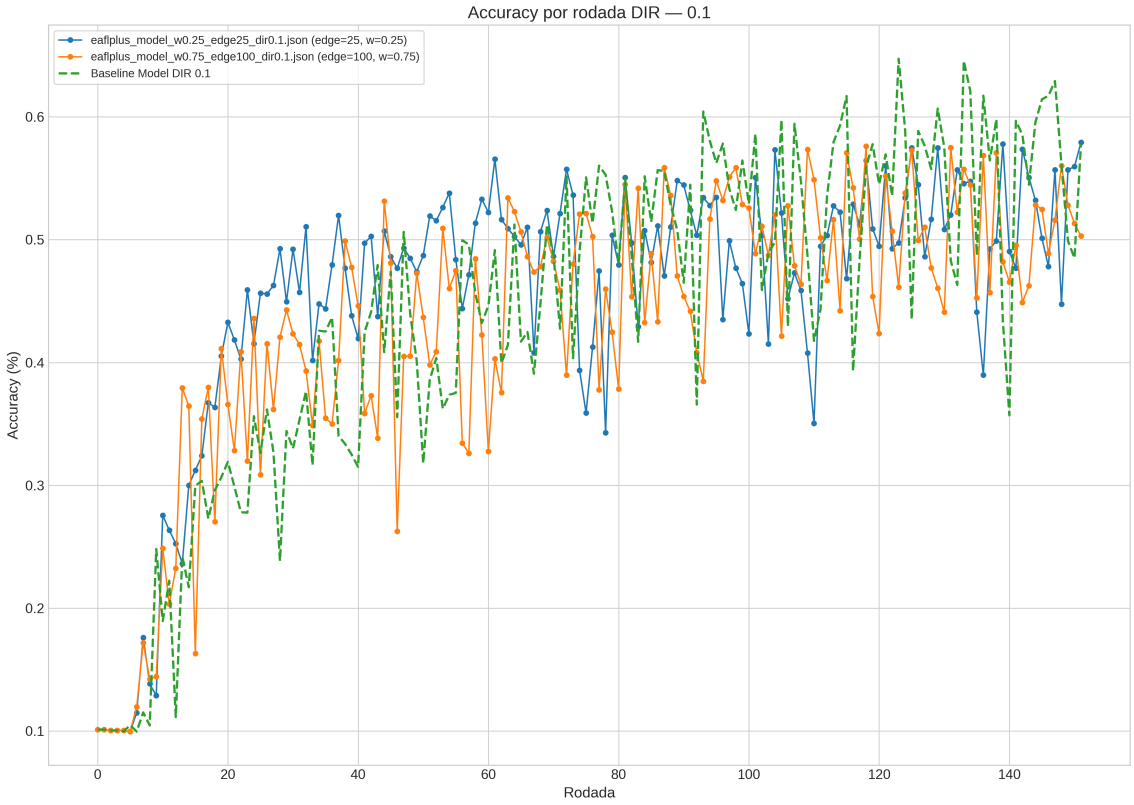

Figura 5: Comparação da evolução da acurácia entre baseline e configurações extremas do EAFL+.

A análise das curvas demonstra que, ao final das 150 rodadas, todas as abordagens convergiram para patamares de acurácia similares, indicando que as restrições impostas pelo EAFL+ não prejudicaram o aprendizado final do modelo. No entanto, destaca-se um comportamento divergente nas fases iniciais do processo: até aproximadamente a rodada 35, ambos os modelos EAFL+ (tanto o cenário fraco quanto o forte) apresentaram uma taxa de crescimento de acurácia superior à do baseline. Esse ganho de velocidade inicial sugere que a seleção guiada por utilidade e bateria permite identificar, logo no início, participantes mais aptos e confiáveis, acelerando a convergência preliminar antes de o sistema estabilizar

no mesmo nível da abordagem aleatória.

Por fim, a análise estende-se ao cenário de heterogeneidade moderada de dados (DIR 1.0), onde a distribuição das classes entre os clientes é mais equilibrada. A Figura 6 ilustra a evolução da acurácia neste contexto.

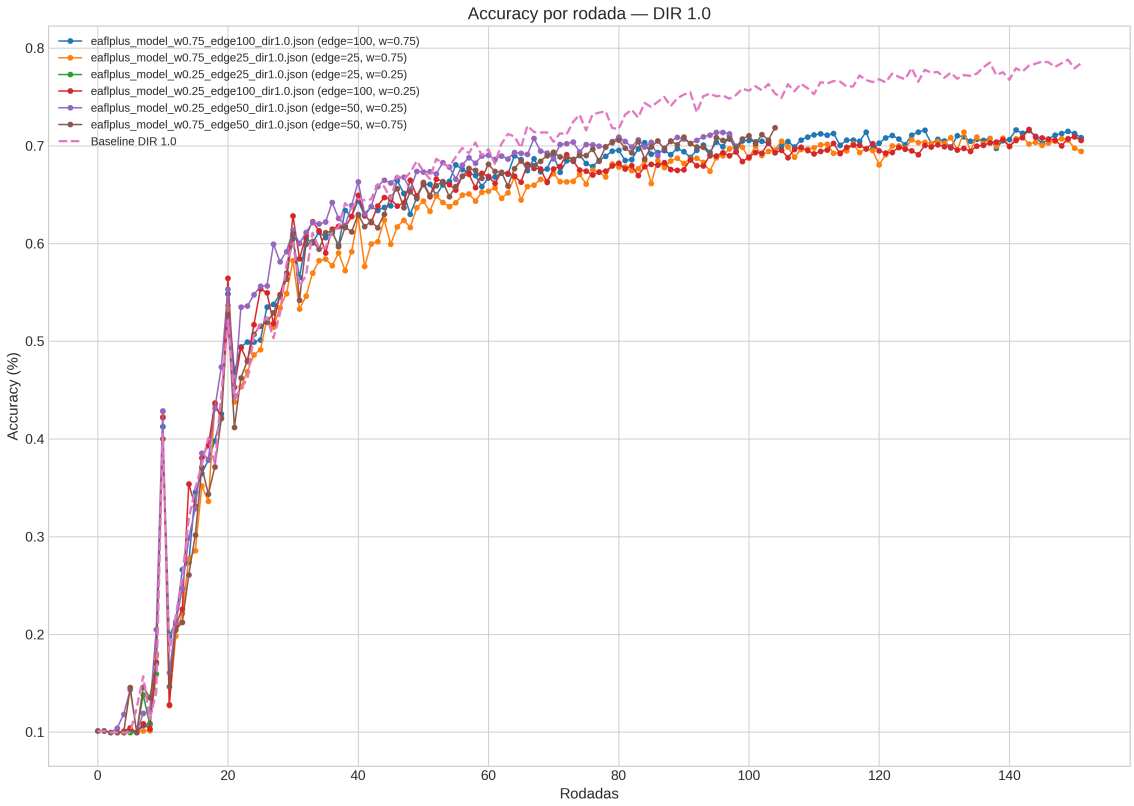

Figura 6: Evolução da acurácia em cenário de heterogeneidade moderada (DIR 1.0).

Diferente do cenário de alta heterogeneidade (DIR 0.1), observa-se aqui um comportamento distinto onde o baseline (linha tracejada) supera as abordagens do EAFL+ em termos absolutos de acurácia final, atingindo picos próximos a 80%, enquanto as variantes do EAFL+ estabilizam-se em torno de 70%. Este fenômeno evidencia o *trade-off* inerente ao sistema proposto: em ambientes onde os dados são mais uniformes, a seleção aleatória beneficia-se da exploração irrestrita de todos os clientes disponíveis, maximizando o aprendizado ao custo de exaurir os recursos dos participantes.

O EAFL+, por sua vez, ao impor restrições de conservação de energia e priorizar o *offloading*, acaba limitando o conjunto de participantes elegíveis. Embora isso resulte em uma acurácia final ligeiramente inferior, as curvas agrupadas das variantes EAFL+ demonstram a robustez do método: independentemente da configuração de infraestrutura ou peso, o sistema mantém uma performance estável e consistente. Portanto, no cenário DIR 1.0, a "perda"de acurácia do EAFL+ deve ser interpretada como o custo operacional necessário para garantir a longevidade da rede, uma troca aceitável para aplicações de IoT onde a

perenidade dos dispositivos é prioritária sobre o desempenho máximo instantâneo.

Por fim, a avaliação do consumo energético total no cenário DIR 1.0 revela nuances importantes sobre o custo operacional do sistema, conforme ilustrado na Figura 7.

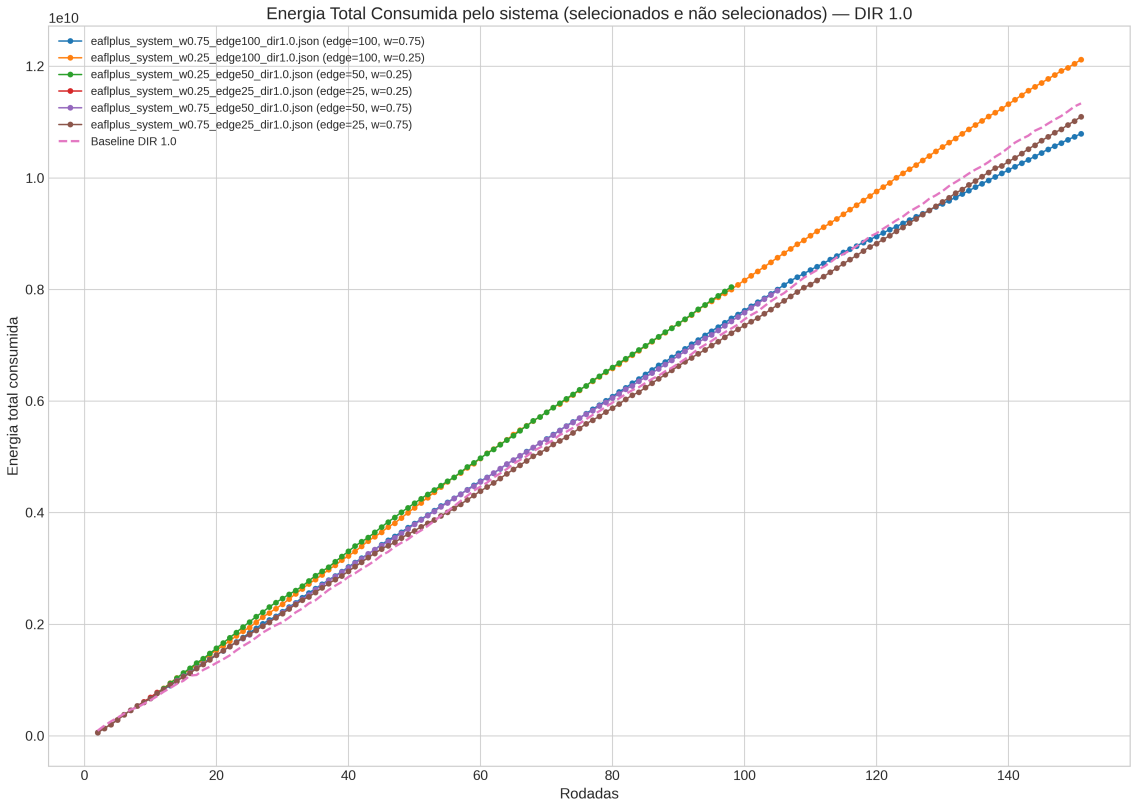

Figura 7: Energia total consumida pelo sistema em cenário de heterogeneidade moderada (DIR 1.0).

A análise das curvas destaca comportamentos contrastantes que evidenciam o impacto dos custos de comunicação. Observa-se que a configuração com peso baixo e alta infraestrutura ($w = 0.25$, 100 servidores) apresentou um consumo total superior ao próprio baseline. Este fenômeno sugere que, em cenários onde a seleção é menos restritiva (w baixo), o custo energético adicional para transmitir os dados e modelos para os numerosos servidores de borda pode superar a economia gerada pelo processamento remoto. Isso confirma a hipótese de que o *overhead* de comunicação do *offloading* não é desprezível e deve ser balanceado.

Em contrapartida, as configurações mais equilibradas, representadas pelas curvas marrom ($w = 0.25$, 25 servidores) e azul ($w = 0.75$, 100 servidores), demonstraram eficiência superior ao baseline. Notavelmente, a curva azul posicionou-se abaixo da marrom, confirmando que, quando o peso de seleção é ajustado corretamente ($w = 0.75$) para priorizar a eficiência, o aumento da infraestrutura (100 servidores) cumpre seu papel de reduzir o consumo global, validando a eficácia do EAFL+ quando seus hiperparâmetros são sintonizados para mitigar o custo de transmissão.

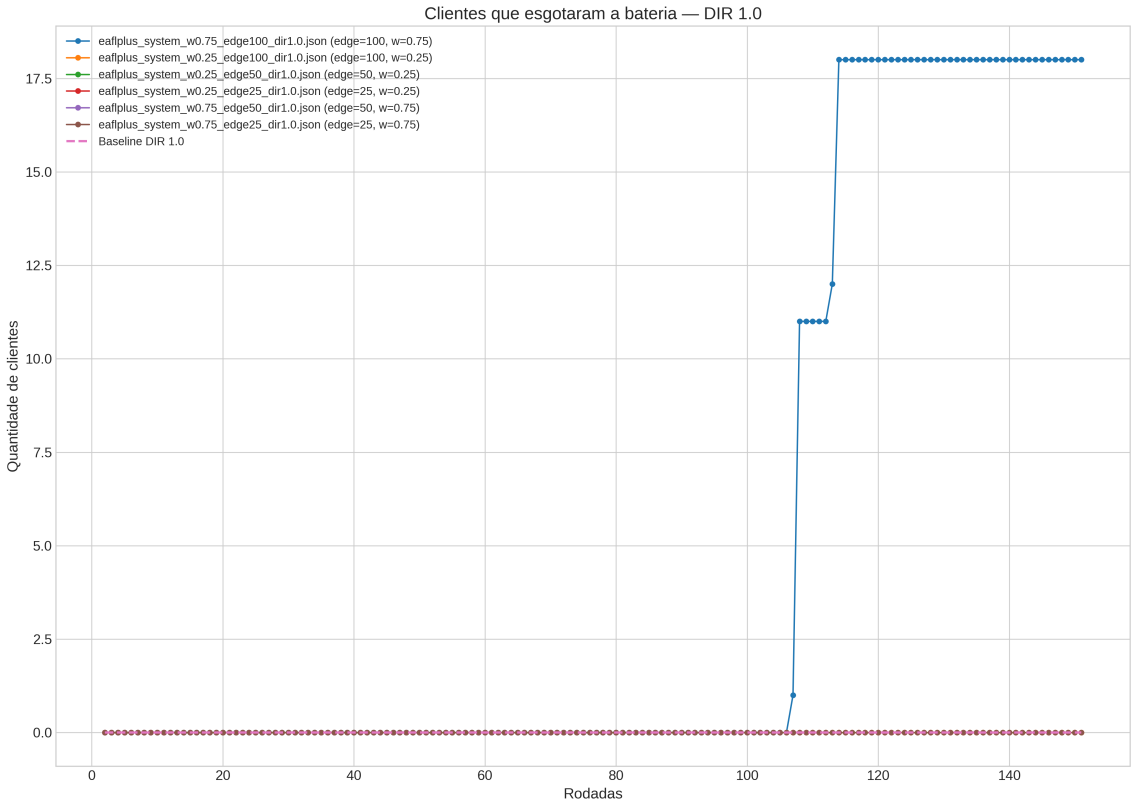

Figura 8: Quantidade acumulada de clientes com bateria esgotada (Dropouts) no cenário DIR 1.0.

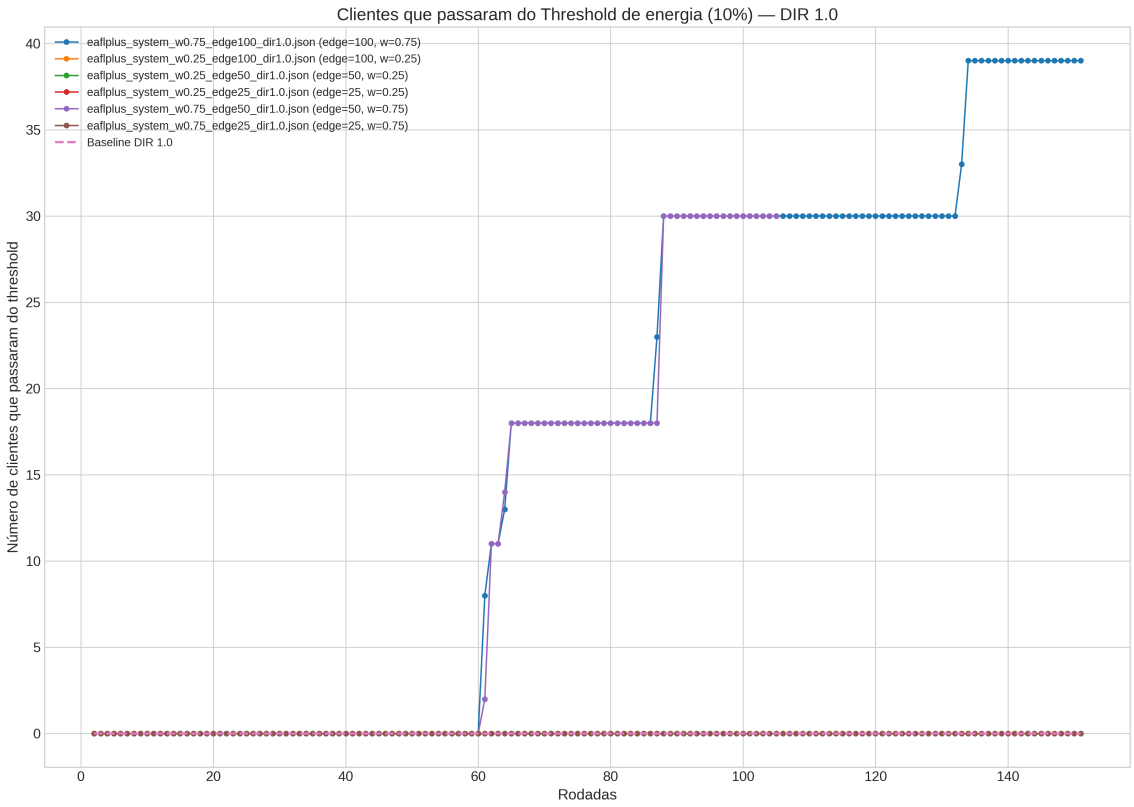

Figura 9: Quantidade acumulada de clientes no limiar crítico (10%) no cenário DIR 1.0.

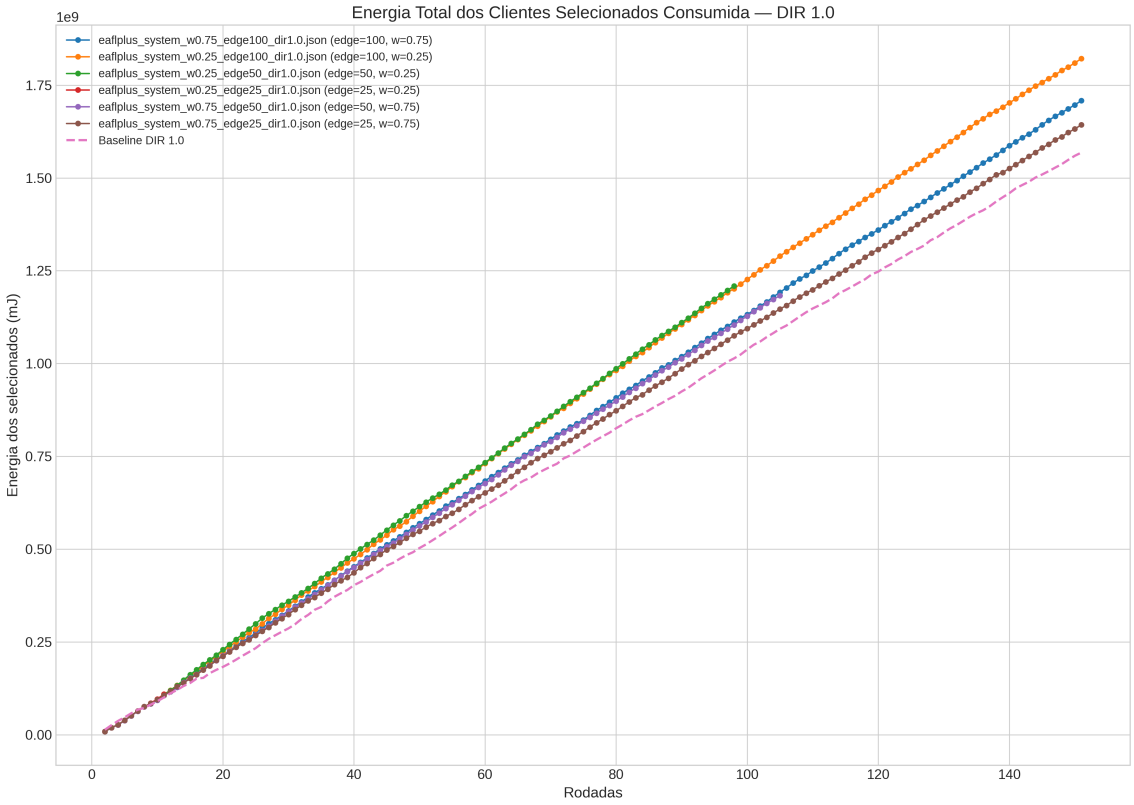

Figura 10: Energia total consumida pelos clientes selecionados no cenário DIR 1.0.

As Figuras 8, 9 e 10 apresentam o comportamento energético do sistema no cenário de heterogeneidade moderada (DIR 1.0). Ao contrário do observado no cenário DIR 0.1, nota-se aqui um fenômeno onde o baseline apresentou, em diversas métricas, um desempenho superior ou equivalente às variantes do EAFL+.

Analisando especificamente o consumo de energia dos clientes selecionados (Figura 10), percebe-se que as curvas do EAFL+ posicionam-se acima da curva do baseline. Este comportamento contraintuitivo pode ser explicado pelo custo energético da comunicação. O mecanismo de *offloading* do EAFL+, embora poupe o processamento local, exige a transmissão intensa de dados (pesos do modelo ou ativações) para os servidores de borda.

No cenário DIR 1.0, onde os dados são mais equilibrados e a convergência local ocorre com mais facilidade, o custo energético de transmitir os dados para a borda (E_{com}) superou a economia gerada pelo descarregamento do processamento (E_{treino}). Isso pode ser demonstrado matematicamente comparando o consumo estimado para ambos os casos. No cenário base (sem *offloading*), o consumo de um dispositivo é dado por $E_{base} = E_{treino} + 2 \cdot E_{com} + E_{idle}$. Já no cenário com *offloading*, o consumo passa a ser predominantemente comunicativo: $E_{offload} = 4 \cdot E_{com} + E_{idle}$.

Portanto, o offloading só é energeticamente vantajoso se $E_{offload} < E_{base}$, o que implica na inequação:

$$4 \cdot E_{com} + E_{idle} < E_{treino} + 2 \cdot E_{com} + E_{idle} \tag{1}$$

Simplificando a inequação, tem-se que a condição $2 \cdot E_{com} < E_{treino}$ não foi satisfeita para este cenário específico, tornando "mais caro" transmitir a tarefa do que processá-la localmente. O baseline, por não realizar *offloading* e manter todo o processamento local, evitou esse *overhead* de transmissão extra.

Isso reflete-se também nas curvas de esgotamento e threshold (Figuras 8 e 9), onde o esforço adicional de comunicação imposto pelo EAFL+ aos clientes selecionados acelerou o consumo de bateria em configurações onde o peso w não foi suficientemente alto para restringir a seleção. Este resultado evidencia que a eficácia do EAFL+ depende criticamente da relação entre o custo de transmissão e o custo de computação; quando a comunicação é custosa e o problema local é simples, a estratégia de *offloading* pode gerar um custo operacional líquido negativo.

6 Conclusão

Este trabalho abordou o desafio crítico do consumo de energia em redes de Aprendizado Federado (FL), propondo e avaliando a implementação do algoritmo EAFL+ no ambiente de simulação Flower. A motivação central residiu na necessidade de viabilizar o treinamento de modelos de Inteligência Artificial em dispositivos móveis e IoT, onde a heterogeneidade de recursos e a limitação de bateria frequentemente inviabilizam a aplicação de abordagens convencionais como o FedAvg ou o Oort.

A implementação da arquitetura colaborativa Nuvem-Borda-Terminal permitiu validar a hipótese de que o descarregamento de computação (*offloading*), aliado a uma seleção de participantes consciente, é capaz de estender a vida útil da rede. Os experimentos realizados no cenário de alta heterogeneidade (DIR 0.1) demonstraram que o EAFL+ foi

capaz de reduzir a taxa de desistências (*dropouts*). A combinação de uma alta densidade de servidores de borda com um peso de seleção elevado ($w = 0.75$) provou ser a configuração mais robusta, protegendo dispositivos vulneráveis e garantindo a convergência do modelo em situações onde a abordagem aleatória falharia por exaustão da rede.

Entretanto, as simulações também revelaram nuances importantes sobre os limites da abordagem, evidenciando que o *offloading* não é uma solução universalmente benéfica. No cenário de dados mais equilibrados (DIR 1.0), observou-se que o custo energético de transmissão dos dados para a borda pode superar a economia obtida no processamento local. Conforme demonstrado analiticamente, a viabilidade do EAFL+ depende da inequação $2 \cdot E_{com} < E_{treino}$. Quando o custo de comunicação é predominante, o sistema incorre em um gasto energético líquido superior ao Baseline, além de apresentar uma leve penalidade na acurácia final devido à restrição conservadora na seleção de participantes.

Conclui-se, portanto, que o EAFL+ não se mostra benéfico como no artigo proposto (AROUJ; ABDELMONIEM, 2024), sendo necessária a realização de mais experimentos e avaliações mais profundas, identificando cenários favoráveis para esse algoritmo.

Referências

AROUJ, Amna; ABDELMONIEM, Ahmed M. Towards energy-aware federated learning via collaborative computing approach. **Computer Communications**, Elsevier, v. 221, p. 131–141, 2024.

BEUTEL, Daniel J et al. Flower: A friendly federated learning research framework. **arXiv preprint arXiv:2007.14390**, 2020.

KRIZHEVSKY, Alex; HINTON, Geoffrey et al. **Learning multiple layers of features from tiny images**. [S.l.], 2009.

LAI, Fan et al. Oort: Efficient Federated Learning via Guided Participant Selection. In: 15TH USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). [S.l.: s.n.], 2021. P. 19–35.

LI, Tian et al. Federated optimization in heterogeneous networks. In: PROCEEDINGS of Machine learning and systems. [S.l.: s.n.], 2020. v. 2, p. 429–450.

LIM, Wei Yang Bryan et al. Federated learning in mobile edge networks: A comprehensive survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 22, n. 3, p. 2031–2063, 2020.

MA, Ningning et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: PROCEEDINGS of the European conference on computer vision (ECCV). [S.l.: s.n.], 2018. P. 116–131.

MCMAHAN, Brendan et al. Communication-efficient learning of deep networks from decentralized data. In: PMLR. ARTIFICIAL intelligence and statistics. [S.l.: s.n.], 2017. P. 1273–1282.

VEPAKOMMA, Praneeth et al. Split learning for health: Distributed deep learning without sharing raw patient data. **arXiv preprint arXiv:1812.00564**, 2018.