CAIROS: Controle Adaptativo do Aprendizado Federado em Redes Sem Fio

Lucas Airam C. de Souza[1,2], **Nadjib Achir**[2], **Miguel Elias M. Campista**[1], **Luís Henrique M. K. Costa**[1]

[1]Universidade Federal do Rio de Janeiro (UFRJ)
[2]INRIA Saclay, École Polytechnique, France

{airam,miguel,luish}@gta.ufrj.br
nadjib.achir@inria.fr

Resumo. *O aprendizado federado veicular (Vehicular Federated Learning – VFL) é aplicado ao treinamento dos modelos de IA para assegurar a privacidade dos dados dos usuários. Entretanto, os clientes apresentam maior variação no canal de comunicação do que em cenários estáticos devido à alta mobilidade dos clientes, ultrapassando o tempo limite para o envio de resposta da rodada. Isso reduz o desempenho do sistema, pois as atualizações de clientes retardatários são descartadas se ultrapassarem o tempo limite de envio na rodada. Este trabalho propõe o CAIROS, uma estratégia para o treinamento de modelos no aprendizado veicular, que permite que cada cliente estime suas condições de rede e de computação por meio de um modelo LSTM. A partir da estimativa, o cliente decide se continua o treinamento ou antecipa o envio dos parâmetros calculados para evitar o estouro do tempo limite. Os resultados mostram que o CAIROS, comparado ao FedAvg, reduz a incidência de descarte de atualizações por estouro de temporizador no VFL em até 38%, aumentando a acurácia dos modelos treinados em até 25%.*

Abstract. *Vehicular Federated Learning (VFL) is applied to the training of AI models to ensure user data privacy. However, clients exhibit greater variation in the communication channel than in static scenarios due to high client mobility, exceeding the round response timeout. This reduces system performance, as updates from straggler clients are discarded if they exceed the transmission timeout for the round. This work proposes CAIROS, a strategy for model training in vehicular learning that allows each client to estimate its network and computing conditions through an LSTM model. Based on this estimate, the client decides whether to continue training or to send the calculated parameters early to avoid a timeout. The results show that CAIROS, compared to FedAvg, reduces the incidence of discarded updates due to timer expiration in VFL by up to 38%, increasing the accuracy of the trained models by up to 25%.*

O presente trabalho foi realizado com apoio do CNPq (310234/2025-5, 407304/2025-8, 402531/2025-6, 408255/2023-4, 405940/2022-0 e 309304/2021-0); da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Código de Financiamento 001 e 88887.954253/2024-00; da FAPERJ (E-26/200.380/2023, E-26/204.122/2024 e E-26/210.778/2025); da FAPESP (2023/00673-7 e 2023/00811-0) e da Fundação de Desenvolvimento da Pesquisa - Fundep - Rota 2030 em conjunto dos nossos parceiros Stellantis e Mobway.

1. Introdução

A inteligência artificial (IA) automatiza e aprimora diversas tarefas cotidianas, podendo automatizar a condução de veículos. Nesse contexto, o aprendizado federado veicular (*Vehicular Federated Learning* – VFL) é aplicado ao treinamento dos modelos para garantir a privacidade dos clientes e de seus dados. O aprendizado federado [McMahan et al. 2017, Thomaz et al. 2025] é uma forma de realizar o treinamento dos modelos de forma colaborativa, substituindo o envio de dados pelo envio de parâmetros atualizados do modelo.

No aprendizado federado síncrono, os clientes devem executar L épocas locais de atualização e enviar os parâmetros calculados ao servidor em um intervalo máximo de tempo $\Delta T_{timeout}$. Entretanto, o ambiente veicular é extremamente dinâmico, o que provoca modificações nas condições do canal de comunicação do cliente. Além disso, os clientes possuem diferentes recursos computacionais disponíveis, o que gera variações no atraso de computação. Assim, estabelecer um número fixo L de épocas locais para todos prejudica clientes que eventualmente estejam em regiões de baixa conexão ou possuam baixa disponibilidade de recursos computacionais.

Caso o intervalo de tempo necessário para executar as épocas locais seja maior, os parâmetros calculados pelo cliente são descartados da atualização do modelo global para aquela rodada. Isso gera problemas como o desperdício computacional do cliente, além da redução de parâmetros na agregação, reduzindo a acurácia do modelo global. Para ilustrar esse efeito, a Figura 1 demonstra que mesmo executando apenas uma época local, utilizar mais clientes aporta mais informação para o modelo global, o que resulta em uma maior acurácia. Nesse experimento, varia-se a quantidade de clientes selecionados, utilizando o algoritmo FedAvg. Os conjuntos de dados dos clientes são gerados por meio da distribuição de Dirichlet com $\alpha = 5.0$ com o conjunto de dados CIFAR-10. Portanto, pode-se verificar o impacto na acurácia do modelo da redução no número de clientes, que pode ocorrer como consequência do término do treinamento após o intervalo máximo de tempo $\Delta T_{timeout}$.

Observa-se, dessa forma, que o fluxo de execução do aprendizado federado para o contexto veicular deve ser aprimorado, a fim de aumentar a sua eficiência. Uma forma de reduzir o descarte de modelos devido ao excesso de tempo para envio da atualização é implementar o aprendizado assíncrono [Wang et al. 2022] ou semissíncrono [Nguyen et al. 2022]. No aprendizado federado assíncrono, os clientes enviam atualizações dos parâmetros ao servidor, que as agrega ao modelo global a cada nova atualização recebida. A agregação é ponderada pelo tempo de atraso de cada cliente, pois atrasos maiores significam que o cliente treinou em um modelo mais defasado do que o atual. Por outro lado, o aprendizado semissíncrono aguarda o recebimento de algumas atualizações antes de fazer a agregação, para reduzir a quantidade de atualizações no servidor e garantir um modelo mais consistente. Porém, ambas as propostas sofrem com o problema do gradiente defasado, que ocorre quando cliente retardatários enviam gradientes calculados para modelos muito antigos. Outra abordagem é adaptar a quantidade de computação que cada cliente realiza. O Controle de Temporização Adaptativo (*Adaptive Timing Control* - ATC) [Ono e Nakao 2025] permite que os clientes enviem os parâmetros de forma antecipada caso estejam saindo da região de conexão com a RSU (*Road Side Unit*). Entretanto, os autores propõem uma abordagem baseada apenas na

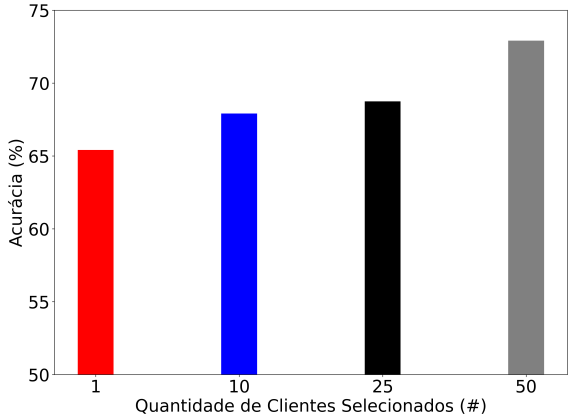


Figura 1. A acurácia do modelo resultante de um treinamento federado é dependente do número de clientes selecionados por rodada, considerando apenas uma época local e 50 rodadas utilizando o conjunto de dados CIFAR-10 e o modelo ResNet10.

posição geográfica do cliente, desconsiderando o contexto computacional e a vazão de rede disponível para transmissão dos dados do modelo quando o cliente está na zona de cobertura.

Este trabalho propõe o CAIROS (Controle Adaptativo do aprendIzado fedeRadO em redes Sem fio)[1], uma estratégia para reduzir perdas de atualizações no aprendizado federado síncrono causadas pelos atrasos de comunicação e computação dos clientes. O trabalho foca o cenário do aprendizado federado veicular, no qual os clientes possuem alta mobilidade e, consequentemente, enfrentam altas variações nas condições do canal de comunicação. A estratégia proposta prevê, de forma ativa, os atrasos de computação e comunicação de cada cliente. O CAIROS prevê dois momentos para verificar o atraso estimado, a cada lote (CAIROS-PB) ou a cada época local (CAIROS-PE). Assim, quando o cliente detecta que o tempo disponível é insuficiente para completar a atualização do modelo, ele antecipa o envio do seu modelo local ao servidor para evitar perdas. A estimativa dos atrasos de comunicação é realizada a partir dos atrasos observados anteriormente pelo cliente. Essa informação é utilizada por um modelo *Long Short-Term Memory* (LSTM) para estimar os atrasos futuros. Por outro lado, os atrasos de computação são fixados em um tempo de 40 milissegundos por lote. Considera-se o problema de classificação de imagens como a tarefa de aprendizado resolvida pelos clientes. Os resultados mostram que, para cenários com um intervalo máximo de tempo mais restrito, $\Delta T_{timeout}$ =10 segundos de acordo com as simulações, o CAIROS exibe uma acurácia 25% maior do que o treinamento com o FedAvg. Isso é possível, pois o CAIROS reduz em cerca de 50% as perdas causadas pelo envio dos modelos fora do intervalo de tempo limite. As principais contribuições deste trabalho são:

- A proposta de uma estratégia para reduzir o descarte de atualizações dos clientes no aprendizado federado. O CAIROS permite que o cliente identifique seu contexto e envie antecipadamente o seu modelo, caso seja necessário;
- A aplicação de um modelo LSTM para estimativa dinâmica dos atrasos dos clientes ao invés de considerar apenas informações geográficas do cliente.

[1]Disponível em `https://github.com/AiramL/cairos`.

- A implementação e disponibilização de uma ferramenta para análise da proposta com clientes usando padrões de mobilidade veicular, um fator pouco explorado no estado da arte.

2. Trabalhos Relacionados

Esta seção apresenta os principais trabalhos que visam reduzir a perda de atualizações dos clientes no aprendizado federado. Inicialmente, são apresentadas as propostas que relaxam as condições do tempo máximo por rodada. A seguir, discutem-se técnicas que utilizam diferentes modelos por cliente para determinar as tarefas a serem executadas com o tempo disponível. Por fim, apresentam-se propostas que determinam a quantidade de dados utilizados como forma de regular o tempo de processamento por cliente ao longo do treinamento, categoria na qual o CAIROS se insere.

2.1. Relaxamento do Tempo Limite

Uma forma de mitigar perdas no aprendizado causadas pelos clientes retardatários (*stragglers*) é utilizar abordagens que reduzem a dependência do tempo para realizar a agregação. Nesse contexto, surgem o aprendizado federado assíncrono e o semissíncrono [Xu et al. 2023]. Wang *et al.* propõem um algoritmo de agregação assíncrono para acelerar o treinamento federado sem perdas de cálculos dos clientes. A abordagem assíncrona elimina a noção de rodada, agregando os modelos sob demanda, para evitar a lentidão causada pelos clientes retardatários [Wang et al. 2022]. Cada vez que um cliente finaliza sua computação local, ele envia o modelo local ao servidor, que realiza a agregação de forma instantânea e envia o resultado ao cliente. Em AAFL (*Adaptive Asynchronous Federated Learning* [Liu et al. 2021] o servidor aguarda pela resposta de alguns clientes antes de realizar a agregação. A proposta integra um modelo de aprendizado por reforço que otimiza o tempo de espera para atualizar o modelo.

O SAFA (*Semi-Asynchronous Federated Averaging*) [Wu et al. 2020] é uma proposta semissíncrona que busca um equilíbrio entre a abordagem síncrona, que aguarda os clientes até o tempo limite; e a assíncrona, que atualiza o modelo global a cada modelo local recebido. Para isso, o servidor aguarda uma quantidade mínima de respostas dentro de uma janela de tempo. Os clientes retardatários, impossibilitados de enviar o modelo dentro da restrição de tempo definida, podem continuar a atualizar o modelo local. Uma vez que o cálculo termina, eles podem enviar o resultado ao servidor. O servidor não descarta as atualizações retardatárias e agrega o resultado do cliente na próxima rodada, computando a diferença entre o modelo local do cliente e o modelo global atual.

Um desafio presente tanto nas abordagens assíncronas quanto nas semissíncronas é a dificuldade de convergência, pois os clientes treinam o modelo com parâmetros distintos. Como os clientes recebem o modelo apenas quando enviam seus parâmetros locais, clientes retardatários ficam com modelos mais defasados do que os clientes rápidos. Assim, ao agregar as versões defasadas, o modelo global pode divergir [Nguyen et al. 2022, Xu et al. 2023]. Portanto, gerenciar a quantidade de computação que cada cliente executa é uma forma eficiente para controlar o atraso de cada um, mantendo o sincronismo e promovendo a convergência do modelo. Para isso, há duas abordagens: definir modelos diferentes para cada cliente ou ajustar a quantidade de dados utilizados por cada cliente para atualização do modelo local.

2.2. Definição de Modelo Diferente para cada Cliente

Outra estratégia possível para reduzir os efeitos negativos da heterogeneidade computacional é atribuir diferentes modelos aos clientes. Dessa forma, o Aprendizado Federado Dinâmico baseado em Níveis (*Dynamic Tiering-based FL* - DTFL) [Mahmoud et al. 2025] utiliza *split learning* para dividir o modelo em camadas distintas e treiná-las. Os clientes, em cada rodada, são classificados em níveis que consideram as capacidades de computação e comunicação. Portanto, clientes com condições mais limitadas treinam menos camadas. Apesar de empregar um estimador para determinar o tempo de treinamento, a técnica é inadequada para ambientes veiculares, onde as condições de conexão variam rapidamente [Clancy et al. 2024].

O FjORD [Horváth et al. 2021] resolve o problema de heterogeneidade de clientes reduzindo a quantidade de parâmetros treinados por cada cliente. Para isso, a estratégia agrupa os clientes conforme as capacidades computacionais e de comunicação, sendo que cada grupo atualiza um percentual da rede neural. A mesma técnica é realizada pelo Helios [Xu et al. 2021]. Os autores propõem uma poda adaptativa do modelo para reduzir a sobrecarga nos clientes retardatários. A principal contribuição da proposta é determinar quais são os clientes retardatários e variar qual parte do modelo é treinada por eles. Assim, a proposta garante que os clientes limitados explorem todo o espaço do modelo. Esta proposta, porém, desconsidera a dinamicidade dos clientes presentes em ambientes veiculares. Portanto, uma forma mais efetiva de controlar o tempo de resposta de cada cliente é através da variação da quantidade de dados usados por cada cliente ao longo do treinamento.

2.3. Adaptação dos Dados de Treinamento

Variar a quantidade de dados ou o número de atualizações realizadas por cada cliente é outra abordagem para evitar o descarte de modelos pelo servidor. O Algoritmo de Orquestração de Lotes (*Batch-Orchestration Algorithm* - BOA) [Yang et al. 2018] implementa no servidor do aprendizado federado um verificador de uso de recursos por dispositivo. Com essa informação, o servidor executa o algoritmo para ajustar o tamanho do lote de cada cliente para a próxima rodada. Entretanto, a abordagem foca o cenário estático em nuvem, onde os atrasos de comunicação são estáveis.

Ajustando Mini-Lotes e Época Local (*Adjusting Mini-Batch and Local Epoch* - AMBLE) [Park et al. 2022] é uma proposta para sincronizar os clientes considerando a heterogeneidade computacional. Assim, o tamanho dos lotes e a quantidade de épocas locais são ajustados conforme a capacidade computacional do cliente selecionado, para que todos finalizem o treinamento local ao mesmo tempo, previamente definido. Entretanto, a abordagem é inadequada para cenários em que há grandes variações de comunicação e computação em uma mesma rodada, pois o ajuste é realizado apenas no início de cada rodada. O Controle de Temporização Adaptativo (*Adaptive Timing Control* - ATC) [Ono e Nakao 2025] é uma proposta que considera a dinâmica dos ambientes veiculares. A principal contribuição dos autores é permitir que os clientes enviem o modelo local de forma antecipada, como no CAIROS, quando os clientes estão a ponto de se desconectar da RSU (*Road Side Unit*). Entretanto, a proposta apresenta pontos em aberto, como a desconsideração das cargas computacionais dos clientes e a falta de metodologia ao indicar como os clientes devem inferir que estão saindo da região de cobertura da RSU.

O FedAWT (*Federated learning with Adaptive Weighted Tree*) [Kim et al. 2025] reduz a quantidade de estouro do tempo de atualização ao longo do treinamento federado com dispositivos IoT. O sistema faz o ajuste dinâmico do número de épocas de cada cliente considerando tanto a heterogeneidade estatística dos dados quanto a relação com o atraso dos clientes. Entretanto, o ajuste das épocas locais é realizado por meio da observação passiva do número de estouro do tempo limite dos clientes ao longo do tempo, o que não reduz o gasto computacional desnecessário.

Diferentemente das propostas anteriores, nas quais o servidor define a quantidade de épocas locais do cliente, no CAIROS cada cliente decide o quanto pode realizar de computação antes que o tempo máximo de envio da atualização seja atingido. Isso é realizado de forma dinâmica pelo próprio cliente, que com base em suas informações atuais, prevê o tempo necessário para computar e enviar o modelo local ao servidor. Dessa forma, o CAIROS não é limitado apenas ao contexto de posição do cliente, como no ATC, sendo mais adequado aos ambientes veiculares dinâmicos. Além disso, a abordagem de treinamento síncrona permite gerar modelos mais acurados.

3. O CAIROS

Esta seção apresenta a proposta deste trabalho para o treinamento de modelos no aprendizado federado veicular. Inicialmente, discutem-se as hipóteses da proposta e descreve-se o cenário de atuação. A arquitetura da proposta e seu fluxo de execução são então apresentados.

3.1. Hipóteses e Descrição do Cenário

Este trabalho assume um cenário de aprendizado federado veicular síncrono, no qual os clientes possuem alto grau de mobilidade e experimentam condições de comunicação variáveis conforme o deslocamento. A atualização do modelo global é realizada ao final de cada rodada r, que dura no máximo o intervalo de tempo $\Delta T_{timeout} = t_{r_f} - t_{r_0}$, idêntico para todas as rodadas. Este intervalo de tempo é iniciado a partir do envio do modelo global ao início da rodada r, no instante t_{r_0} e finalizado no instante de tempo t_{r_f}. Os clientes que eventualmente iniciem o treinamento após o instante t_{r_0} e enviem o modelo em um tempo $t_r^{(k)} > t_{r_f}$, implicando em um $\Delta T_{r_t}^{(k)} > \Delta T_{timeout}$, não são contabilizados para a rodada atual. Entretanto, a ausência de participação do cliente em uma rodada não o impede de participar das rodadas posteriores. O servidor seleciona os clientes de forma aleatória para participar do treinamento em uma rodada e não armazena informações sobre eventuais falhas dos clientes. Dessa forma, todos os clientes possuem a mesma probabilidade de serem selecionados pelo servidor em uma rodada r para realizar o treinamento do modelo.

Assume-se que os clientes possuem capacidade computacional suficiente para treinar o modelo do aprendizado federado ω_m e para executar o modelo LSTM estimador de atrasos ω_t. O modelo ω_m é relacionado à tarefa de aprendizado, sendo que neste trabalho o caso de uso é a classificação de imagens. Dada uma imagem x, o classificador $\omega_m(\phi, x)$ retorna a classe mais provável para a imagem dado ϕ, o conjunto de parâmetros do modelo. Por outro lado, o estimador $\omega_t(D_{t_i}^{(k)})$ retorna o atraso estimado para continuar o treinamento do modelo e enviá-lo ao servidor em um instante de tempo t_i com base nos atrasos $D_{t_i}^{(k)}$ experimentados pelo cliente k anteriormente. Como o estimador não é

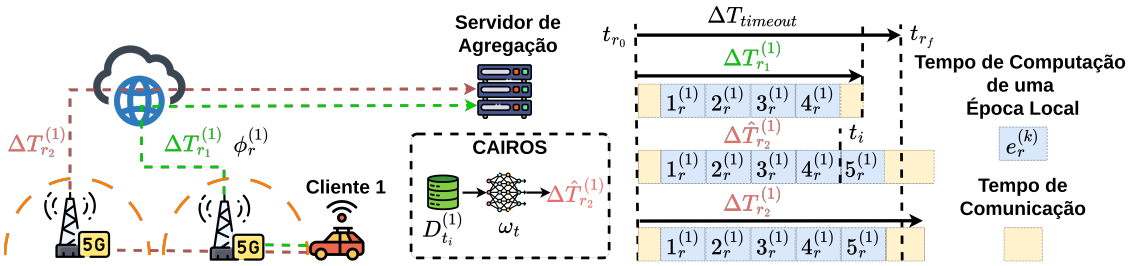


Figura 2. Dinâmica do aprendizado federado veicular tradicional em comparação com o CAIROS. O Cliente 1, após realizar o treinamento, deve enviar os parâmetros $\phi_r^{(1)}$ do modelo para o servidor. O arcabouço original determina que o cliente compute $L = 5$ épocas locais antes de enviar o modelo. Dessa forma, após a computação de todas as épocas locais, o cliente utiliza um intervalo de tempo $\Delta T_{r_2}^{(1)}$. Por outro lado, o CAIROS é flexível e permite que o cliente estime o atraso de computação restante e comunicação no instante de tempo t_i, $\Delta \hat{T}_{r_2}^{(1)}$. Ao verificar que esse valor é maior do que $\Delta T_{timeout}$, o cliente antecipa a transmissão para $\Delta T_{r_1}^{(1)}$. Isto é feito a partir do modelo ω_t, que estima os atrasos futuros. Assim, o CAIROS evita o descarte da atualização.

treinado ao longo do treinamento federado, os seus parâmetros são omitidos da notação. $D_{t_i}^{(k)}$, neste trabalho, é um vetor formado por uma janela deslizante, do k-ésimo cliente, composto pelos 10 últimos atrasos experimentados pelo cliente com o servidor anteriores ao instante de tempo t_i.

3.2. Arquitetura do Sistema Proposto e Fluxo de Execução

O CAIROS visa reduzir as perdas causadas pelo descumprimento do tempo limite de envio da atualização do modelo. As perdas são tanto de trabalho, pois os clientes computam atualizações que são descartadas pelo servidor, quanto de desempenho do modelo, pois com menos clientes treinando, são necessárias mais rodadas de treinamento. O problema é agravado quando o servidor configura um tempo limite para atualização muito restrito, o que impede a participação no treinamento de clientes que possuam dispositivos incapazes de treinar o modelo em tempo hábil. Mesmo existindo a possibilidade de que o servidor realize uma seleção de clientes para evitar longos atrasos e eventuais descartes de modelo [de Souza et al. 2025], no VFL os clientes são dinâmicos dentro de uma rodada. Assim, os clientes estão em constante movimento e os atrasos de comunicação apresentam alta variação, dificultando a seleção de clientes de forma eficiente. Assim, o cliente deve ser capaz de estimar suas condições de comunicação para verificar qual é a faixa de tempo disponível para enviar a atualização.

A Figura 2 exibe a dinâmica de um cliente no aprendizado federado. Inicialmente, o Cliente 1 é selecionado para treinar o modelo. Ao recebê-lo, o cliente inicia o treinamento local. No treinamento tradicional, com o FedAvg, o servidor determina os hiperparâmetros do modelo e compartilha com os clientes. Entre os hiperparâmetros está definido o número L de épocas locais que o cliente deve executar. Dessa forma, o Cliente 1 envia o modelo atualizado ao servidor no intervalo de tempo $\Delta T_{r_2}^{(1)}$. Entretanto, este intervalo de tempo pode ser maior do que o $\Delta T_{timeout}$, definido pelo servidor como

intervalo limite para aceitação de novas atualizações na rodada r.

O diferencial do CAIROS está no estimador de atrasos. O cliente coleta ativamente estatísticas locais de rede e computacionais. Essa informação é utilizada para treinar um estimador $\omega_t(D_{t_r^{(k)}})$, antes de iniciar o treinamento federado. Assim, o cliente, quando selecionado, estima periodicamente o intervalo de tempo $\Delta\hat{T}_{r_2}^{(k)}$ e verifica se este é maior do que $\Delta T_{timeout}$ antes de continuar a atualização do modelo local. Dessa forma, o cliente determina se a sua computação local será descartada ao enviar o modelo em um intervalo de tempo posterior. Assim, ele decide enviar o modelo no intervalo de tempo oportuno $\Delta T_{r_1}^{(k)}$. Isso permite ao cliente participar da rodada, controlando a quantidade de computação realizada e evitando o descarte do modelo.

Algoritmo 1: Algoritmo de Execução do Cliente pelo CAIROS

Entrada: $D_s^{(k)}, D_t^{(k)}, \omega_m, \phi_{r-1}^g, \eta, \omega_t, L, \Delta T_{timeout}$;
Saída: $\phi_r^{(k)}$;
while $e_r^{(k)} < L$ **do**
  for $b \in D_s^{(k)}$ **do**
    $\phi_r^{(k)} \leftarrow \phi_{r-1}^{(g)} - \eta \nabla \mathcal{L}(\phi_r^{(k)}, b)$;
    if *CAIROS-PB* **then**
      $\Delta\hat{T}_r^{(k)} \leftarrow \omega_t(D_{t_i}^{(k)})$;
      if $\Delta\hat{T}_r^{(k)} > \Delta T_{timeout}$ **then**
        EnviarParâmetros($\phi_r^{(k)}$);
        Parar;
      end
    end
  end
  if *CAIROS-PE* **then**
    $\Delta\hat{T}_r^{(k)} \leftarrow \omega_t(D_t^{(k)})$;
    if $\Delta\hat{T}_r^{(k)} > \Delta T_{timeout}$ **then**
      EnviarParâmetros($\phi_r^{(k)}$);
      Parar;
    end
  end
end

3.3. Dinâmica de Treinamento

O treinamento federado veicular segue a formulação proposta no algoritmo FedAvg [McMahan et al. 2017]. Assim, o servidor de agregação seleciona um subconjunto $\mathcal{N}_r \in \mathcal{K}$, onde $\mathcal{K}$ é o conjunto de clientes disponíveis para o treinamento, para realizar o ajuste do modelo na r-ésima rodada. Cada cliente selecionado recebe os parâmetros globais ϕ_{r-1}^g da rodada anterior para iniciar o treinamento local. A partir do conjunto de dados local e da função de perda, o cliente atualiza seus parâmetros locais. A função de perda $\mathcal{L}(\phi, b)$ utilizada é:

$$\mathcal{L}(\phi, b) = \frac{1}{|b|} \sum_{(x,y)\in b} CE(\omega_m(\phi, x), y), \tag{1}$$

onde ϕ representa o conjunto de parâmetros do modelo e b o lote de dados (*batch*). Cada lote é composto por amostras que contêm um conjunto de características x e seu rótulo y. O modelo do cliente $\omega_m(\phi, x)$ retorna o rótulo previsto $\hat{y}$ para o dado x. Esse valor é comparado ao rótulo original y para determinar o valor da função de perda. Neste trabalho, a comparação entre o rótulo original e o previsto, $CE(\hat{y}, y)$, é realizada com a função de entropia cruzada.

O CAIROS prevê que o cliente pare de atualizar o modelo e o envie ao servidor de agregação, caso estime que o tempo limite será atingido. O Algoritmo 1 define o comportamento descrito nas linhas 6-10 e 14-18. O CAIROS prevê dois momentos para verificar o atraso estimado, a cada lote (CAIROS-PB) ou a cada época local (CAIROS-PE). No caso em que o cliente verifica a cada lote, ele utiliza o seu estimador ω_t para prever o atraso $\Delta\hat{T}_i$ que representa o intervalo entre o início da rodada e o fim da transmissão do modelo ao final do próximo lote neste caso. Para o CAIROS-PE, $\Delta\hat{T}_i$ segue a mesma lógica, porém o intervalo de tempo é estimado para o final da próxima época local. O estimador prevê as condições de computação e comunicação, retornando o valor esperado do intervalo de tempo entre o início da rodada e o fim do envio do modelo caso o cliente continue o cálculo no próximo lote ou época. Caso o intervalo de tempo estimado seja insuficiente para realizar mais uma atualização do modelo, o cliente interrompe o seu treinamento e compartilha os parâmetros do modelo calculados até o momento com o servidor. O CAIROS, por outro lado, requer o uso de um estimador de atrasos futuros, sendo este uma limitação para clientes com baixo poder computacional. Além disso, a eficiência da proposta é dependente do desempenho do estimador. Esse algoritmo é avaliado na seção seguinte e comparado ao FedAvg.

4. Avaliação Experimental

Esta seção analisa o desempenho do CAIROS. Primeiramente, é descrito o ambiente e os parâmetros usados para execução dos experimentos. Em seguida, a metodologia, os experimentos executados e a discussão dos resultados obtidos são apresentados.

4.1. Ambiente de Execução e Metodologia Experimental

Este trabalho utiliza o modelo ResNet-10 [Gong et al. 2022] para a classificação de imagens do conjunto de dados CIFAR-10 [Krizhevsky et al. 2009]. No entanto, o objetivo é validar a proposta e compará-la com o estado da arte. O CAIROS é genérico para ser usado como critério de seleção de clientes para o treinamento de outros modelos, inclusive aqueles que oferecem melhor desempenho que o usado na avaliação. Outros modelos podem ser recomendados por outras estratégias [De Souza et al. 2024] e adicionados ao sistema.

O modelo de aprendizado profundo é implementado usando o framework PyTorch v2.3.0. O treinamento é executado no flower v1.18.0 [Beutel et al. 2020], um arcabouço de código aberto para simular o aprendizado federado. Antes de iniciar o treinamento, há uma fase de pré-processamento onde se normalizam as características e divide-se aleatoriamente o conjunto de dados de treino entre os clientes, utilizando a função de Dirichlet com $\alpha = 5.0$ para criar distribuições de dados não independentes e identicamente distribuídas (IID). Essa forma de criar a distribuição de dados simula um cenário mais realista, onde os dados dos clientes são únicos e personalizados de acordo com os hábitos do cliente.

O modelo de mobilidade veicular é implementado usando o SUMO v1.21.0, com o modelo de mobilidade de Manhattan [Patanè et al. 2024, de Souza et al. 2025]. O tráfego de veículos é gerado pela ferramenta **randomTrips** em uma região de 600×600 m^2, a uma velocidade constante de 50 km/h e em uma única faixa. A comunicação 5G dos clientes usa um modelo de canal que aloca igualmente a largura de banda de 10 Gbps e 20 Gbps para *upload* e *download*, respectivamente, para os usuários [Zhu et al. 2021, Chatzoulis et al. 2023]. O modelo de comunicação utilizado é o 3GPP TR 38.901 com frequência de portadora 5,9 GHz, potência de transmissão da estação rádio base igual a 46 dBm, enquanto a potência de transmissão dos clientes é igual a 23 dBm. A largura de banda de transmissão e recepção são configuradas para 20 e 10 MHz respectivamente. Uma única estação base é usada para todos os usuários, que se conectam a um servidor de agregação remoto.

Os clientes do aprendizado federado são simulados por meio de múltiplos processos em um servidor equipado com uma CPU AMD EPYC 7452 com 64 núcleos e 32 GB de RAM e duas GPUs NVIDIA Tesla V100S de 8 GB. Os resultados comparam três estratégias de treinamento: FedAvg [McMahan et al. 2017], o CAIROS executado a cada lote (CAIROS-PB) e executado a cada época local (CAIROS-PE). No FedAvg, os clientes executam exatamente L épocas locais antes de enviar o modelo ao servidor. Já o CAIROS-PB executa a estimativa dos atrasos futuros a cada lote e envia antecipadamente os resultados da atualização se identificar um atraso maior do que o determinado pelo servidor. Por outro lado, o CAIROS-PE realiza a mesma abordagem, porém ao final de cada época local. O estimador de atraso possui a arquitetura em um trabalho anterior [de Souza et al. 2025]. Este é treinado antes de inicializar o aprendizado federado, com um conjunto de dados com a vazão nominal de bytes por segundo do cliente, medida por meio do tempo de envio das mensagens anteriores. É utilizada uma janela deslizante de 10 amostras, cada amostra representando 100 milissegundos, para estimar os atrasos futuros. Uma vez que o modelo é treinado, os parâmetros são congelados para executar apenas previsões.

4.2. Experimentos Realizados

A avaliação da proposta considera dois aspectos: a eficiência do treinamento e o desempenho do modelo treinado. A eficiência é medida por meio da razão entre a quantidade total de atualizações recebidas sobre o total de atualizações esperadas no servidor de agregação, discutida na Seção 4.2.1. Já o desempenho do modelo é medido por meio da acurácia obtida no conjunto de testes e discutido na Seção 4.2.2.

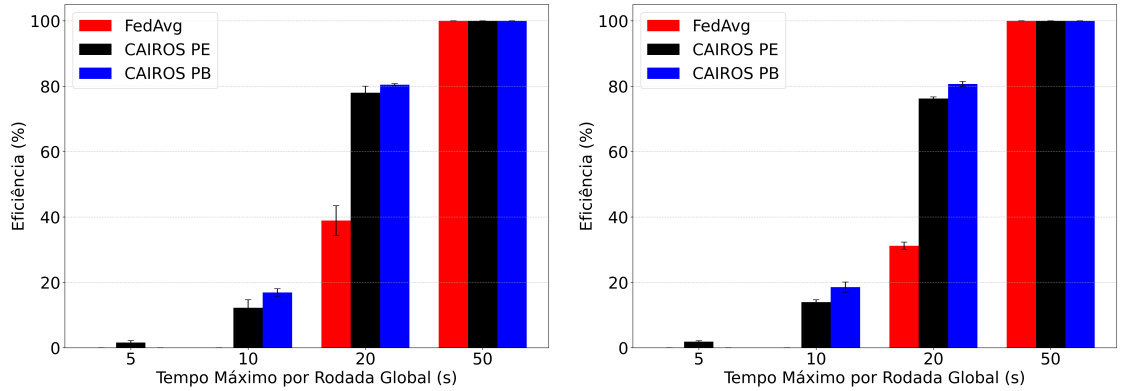
4.2.1. Eficiência do Treinamento

Para medir a eficiência do treinamento, é necessário definir a quantidade de atualizações recebidas e o total de atualizações esperadas. A quantidade total de atualizações recebidas P ao longo de todas as rodadas de treinamento é dada pela Equação 2.

$$P = \sum_{r=1}^{R} \sum_{k}^{\mathcal{N}_r} d_r^{(k)}, \text{ onde } d_r^{(k)} = \begin{cases} 1, & \text{se } \Delta T_{r_t}^{(k)} \leq \Delta T_{timeout} \\ 0, & \text{caso contrário,} \end{cases} \tag{2}$$

nela, são contados todos os clientes que responderam no intervalo de tempo máximo definido pelo servidor. O total de atualizações esperadas é dado pela multiplicação do número de rodadas R e da quantidade de clientes selecionados por rodada C constante ao longo das rodadas, ou seja, $R \cdot C$. Assim, caso todos os clientes respondam dentro do limite estabelecido, a eficiência do sistema é de 100%, e 0% quando nenhum cliente responder.

As Figuras 3(a) e 3(b) exibem o valor da eficiência $E = \frac{P}{R.C}$, ao variar o atraso máximo $\Delta T_{timeout}$, ao selecionar 10 e 30 clientes por rodada, respectivamente. Pode-se observar que para um tempo máximo muito restrito, menor do que 20 segundos, apenas as abordagens com o CAIROS evitam 100% das perdas. Para o cenário em que o atraso máximo é de 20 segundos, a eficiência do FedAvg é de 40%, enquanto o CAIROS oferece uma eficiência superior a 75%. O comportamento é independente da quantidade de clientes selecionados para realizar o treinamento federado.



(a) Eficiência ao selecionar 10 clientes por rodada. (b) Eficiência ao selecionar 30 clientes por rodada.

Figura 3. Comparação da eficiência entre as três abordagens avaliadas ao variar o tempo máximo de resposta e o número de clientes selecionados.

Assim, pode-se concluir que o CAIROS reduz o desperdício de trabalho dos clientes no aprendizado federado veicular se comparado à abordagem de treinamento tradicional. Essa economia representa também um melhor uso da banda passante, pois os modelos enviados ao servidor não são descartados. Entretanto, outro aspecto a ser analisado é o impacto do envio dos parâmetros de forma antecipada na acurácia do modelo global.

4.2.2. Desempenho do Modelo

Este experimento considera a evolução temporal da acurácia para as diferentes abordagens avaliadas, considerando diferentes intervalos máximos para o envio do modelo. O objetivo é verificar qual é o impacto do aumento da eficiência na acurácia do modelo global. A Figura 4 mostra os resultados desse experimento. O atraso máximo utilizado é variado utilizando os valores de [5,10,20,50] segundos, para um cenário de 50 clientes onde são selecionados [10, 30] clientes. Para o cenário com um intervalo máximo de resposta mais restrito, menor do que 50 segundos, o CAIROS apresenta vantagem em

desempenho em relação à abordagem FedAvg. Para o cenário com o atraso máximo tolerado de 10 segundos, o CAIROS apresenta uma acurácia de 25% maior do que o FedAvg. Além disso, é interessante notar que a abordagem de estimar o atraso a cada época, o CAIROS-PE, gera um resultado melhor do que a abordagem a cada lote, o CAIROS-PB, na maioria dos casos. A explicação está relacionada à quantidade de treinamento que cada cliente realiza em cada abordagem. Como o CAIROS-PB possui uma granularidade maior, o treinamento pode ser interrompido ao longo da época local. Quando o estimador identifica a possibilidade de falha, o envio é antecipado. Entretanto, se o modelo foi treinado com poucos dados, o modelo global reduz seu desempenho. Assim, verifica-se que há um compromisso entre a quantidade de treinamento antes do envio e o desempenho do modelo global. Porém, mesmo com o compromisso identificado, o modelo global aprende.

Outra forma de visualizar o resultado anterior é analisando a evolução, a cada rodada, da acurácia, dado um atraso máximo. Assim, a Figura 5 exibe o resultado para os intervalos máximos de 10 e 50 segundos, respectivamente, selecionando 10 clientes entre os 50 disponíveis. Esses dois cenários foram selecionados para estudar os casos nos quais o CAIROS e o FedAvg têm os melhores desempenhos.

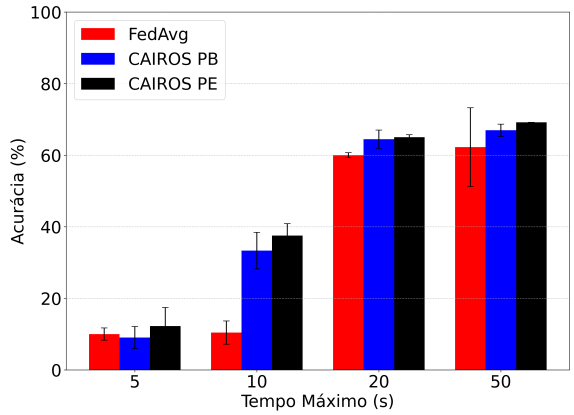
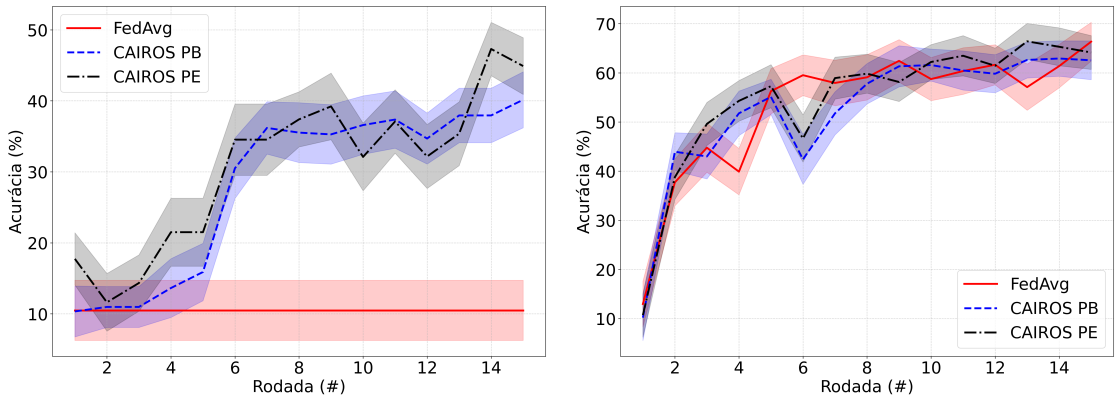


Figura 4. Acurácia de teste para diferentes abordagens com o conjunto de dados CIFAR-10, variando o intervalo máximo para o envio do modelo. Para atrasos inferiores a 50 segundos o CAIROS aumenta o desempenho do modelo em até 25%.

Para um intervalo máximo de envio de modelo igual a 10 segundos, o FedAvg é incapaz de aprender e o modelo estagna em 15% de acurácia. Isso ocorre pelo número de atualizações recebidas pelo servidor. Como a maioria das atualizações são descartadas, o treinamento não evolui. Por outro lado, a acurácia do modelo ao utilizar as duas estratégias com o CAIROS evolui ao longo das rodadas, como exibido na Figura 5(a). Estatisticamente, as estratégias PB e PE apresentam a mesma acurácia devido à superposição do intervalo de confiança. Entretanto, a acurácia média da estratégia CAIROS-PE é um pouco superior, indicando um compromisso entre eficiência e acurácia. No segundo caso, com uma tolerância maior, de 50 segundos, o FedAvg produz um modelo com um nível de acurácia igual ao CAIROS. O comportamento é esperado, pois a partir da observação do resultado da eficiência para $\Delta T_{timeout} = 50s$, as abordagens possuem a mesma quantidade de clientes.



(a) Acurácia do modelo com intervalo máximo de envio do modelo igual a 10 segundos.

(b) Acurácia do modelo com intervalo máximo de envio do modelo igual a 50 segundos.

Figura 5. Comparação da acurácia do modelo global ao treinar com as três abordagens avaliadas e variar o tempo máximo de resposta.

5. Conclusões e Trabalhos Futuros

Este trabalho apresentou o CAIROS, uma estratégia para aumentar a eficiência do aprendizado federado veicular síncrono, considerando variações no canal de comunicação e de carga computacional dos clientes. O CAIROS insere um estimador de atraso no cliente. Assim, a partir da previsão, o cliente pode decidir enviar o modelo antes de concluir todas as épocas locais a fim de evitar o descarte do modelo pelo servidor ao descumprir o tempo máximo de espera. Os resultados mostram que o CAIROS é capaz de aprender em cenários cujo atraso máximo tolerado é restrito, menor do que 20 segundos por rodada, aumentando em até 25% a acurácia do modelo. Ao mesmo tempo, o CAIROS mantém o mesmo nível de acurácia que o FedAvg quando o tempo máximo é menos restrito. Além disso, a proposta atinge alta eficiência, evitando o descarte de modelos, cerca de 35% mais eficiente do que o FedAvg no cenário com um atraso máximo de 20 segundos. Como trabalhos futuros, pretende-se investigar cenários mais desafiadores, com clientes que desconectam por longos períodos, com variabilidade computacional e investigar o impacto computacional no cliente pela adoção de diferentes modelos estimadores. Além disso, outro ponto a ser aprofundado é o compromisso entre o quanto treinar antes de antecipar o envio e o desempenho do modelo.

Referências

Beutel, D. J. et al. (2020). Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390*.

Chatzoulis, D. et al. (2023). 5G V2X Performance Comparison for Different Channel Coding Schemes and Propagation Models. *Sensors*, 23(5):2436.

Clancy, J. et al. (2024). Wireless Access for V2X Communications: Research, Challenges and Opportunities. *Communications Surveys & Tutorials*.

De Souza, L. A. C. et al. (2024). AutoMHS-GPT: Automated Model and Hyperparameter Selection with Generative Pre-Trained Model. Em *CloudNet*, páginas 1–8. IEEE.

de Souza, L. A. C. et al. (2025). TOFL: Time Optimized Federated Learning. Em *SBSeg*. SBC.

Gong, J., Liu, W., Pei, M., Wu, C. e Guo, L. (2022). ResNet10: A Lightweight Residual Network for Remote Sensing Image Classification. Em *ICMTMA*, páginas 975–978.

Horváth, S. et al. (2021). FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. Em *Advances in Neural Information Processing Systems*, volume 34, páginas 12876–12889.

Kim, G. et al. (2025). FedAWT: Adaptive Federated Learning via Dynamic Epoch Adjustment for Heterogeneous Clients in Ad-Hoc Networks. *Internet of Things*, 34:101771.

Krizhevsky, A. et al. (2009). Learning Multiple Layers of Features from Tiny Images.

Liu, J. et al. (2021). Adaptive Asynchronous Federated Learning in Resource-Constrained Edge Computing. *Transactions on Mobile Computing*, 22(2):674–690.

Mahmoud, S. et al. (2025). Speed Up Federated Learning in Heterogeneous Environments: A Dynamic Tiering Approach. *Internet of Things Journal*, 12(5).

McMahan, B. et al. (2017). Communication-efficient Learning of Deep Networks from Decentralized Data. *Artificial Intelligence and Statistics*, páginas 1273–1282.

Nguyen, J. et al. (2022). Federated Learning with Buffered Asynchronous Aggregation. Em *International Conference on Artificial Intelligence and Statistics*, páginas 3581–3607. PMLR.

Ono, S. e Nakao, A. (2025). Adaptive Timing Control of Parameter Aggregation in Vehicular Federated Learning. Em *ICCE*, páginas 1–6. IEEE. ISSN: 2158-4001.

Park, J. et al. (2022). AMBLE: Adjusting Mini-Batch and Local Epoch for Federated Learning with Heterogeneous Devices. *JPDC*, 170:13–23.

Patanè, R. et al. (2024). Can Vehicular Cloud Replace Edge Computing? Em *WCNC*. IEEE.

Thomaz, G. A. et al. (2025). AGATA – Arquitetura para Gerenciamento Automático de Tarefas de Aprendizado Federado. Em *SBRC*, páginas 121–128. SBC.

Wang, Z. et al. (2022). Asynchronous Federated Learning over Wireless Communication Networks. *Transactions on Wireless Communications*, 21(9):6961–6978.

Wu, W. et al. (2020). SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning with Low Overhead. *Transactions on Computers*, 70(5):655–668.

Xu, C., Qu, Y., Xiang, Y. e Gao, L. (2023). Asynchronous Federated Learning on Heterogeneous Devices: A Survey. *Computer Science Review*, 50:100595.

Xu, Z. et al. (2021). Helios: Heterogeneity-aware Federated Learning with Dynamically Balanced Collaboration. Em *DAC*, páginas 997–1002. IEEE.

Yang, E. et al. (2018). An Adaptive Batch-Orchestration Algorithm for the Heterogeneous GPU Cluster Environment in Distributed Deep Learning System. Em *International Conference on Big Data and Smart Computing (BigComp)*, páginas 725–728. IEEE.

Zhu, Q. et al. (2021). 3GPP TR 38.901 Channel Model. Em *The Wiley 5G Ref: The Essential 5G Reference Online*, páginas 1–35. Wiley Press Hoboken, NJ, USA.