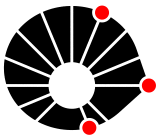

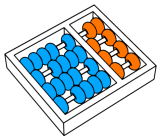

Relatório de Comunicação Eficiente para Aprendizado Federado

G. O. Campos

Technical Report - IC-21-02 - Relatório Técnico
December - 2021 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Comunicação Eficiente para Aprendizado Federado

Gabriel Oliveira Campos*

9 de março de 2022

1 Introdução

Três grandes desenvolvimentos estão atualmente transformando as maneiras como os dados são criados e processados: Em primeiro lugar, com o advento da Internet das Coisas (IoT), o número de dispositivos inteligentes no mundo cresceu rapidamente nos últimos anos. Muitos desses dispositivos são equipados com vários sensores e hardware cada vez mais potente que lhes permite coletar e processar dados em escalas sem precedentes[1].

Em um desenvolvimento simultâneo, o aprendizado profundo revolucionou as maneiras como as informações podem ser extraídas de recursos de dados com sucessos inovadores em áreas como visão computacional, processamento de linguagem natural ou reconhecimento de voz, entre muitos outros.

O aprendizado profundo tem uma boa escalabilidade com quantidades crescentes de dados e seus sucessos surpreendentes nos últimos tempos podem ser pelo menos parcialmente atribuídos à disponibilidade de conjuntos de dados muito grandes para treinamento. Portanto, existe um grande potencial no aproveitamento dos ricos dados fornecidos pelos dispositivos IoT para o treinamento e melhoria dos modelos de aprendizado profundo.

Ao mesmo tempo, a privacidade de dados se tornou uma preocupação crescente para muitos usuários. Vários casos de vazamento de dados e uso indevido nos últimos tempos demonstraram que o processamento centralizado de dados apresenta alto risco para a privacidade dos usuários finais. Como os dispositivos IoT geralmente coletam dados em ambientes privados, muitas vezes mesmo sem o conhecimento explícito dos usuários, essas preocupações são particularmente fortes. Portanto, geralmente não é uma opção compartilhar esses dados com uma entidade centralizada que poderia conduzir o treinamento de um modelo de aprendizado profundo. Em outras situações, o processamento local dos dados pode ser desejável por outros motivos, como maior autonomia do agente local[1, 2].

Sendo assim, o aprendizado federado resolve esse problema, pois ele permite várias partes para treinar em conjunto um modelo de aprendizado profundo em seus dados, sem que nenhum dos participantes tenha que revelar seus dados para um servidor centralizado.

1.1 Aplicações

Existem vários usos para técnicas de aprendizagem federada. Os seguintes aplicativos se adequam melhor a esta ideia de projeto de uso da arquitetura sendo:

- **Recomendação de serviço:** colete históricos de busca e localização, sem violar privacidade do usuário, para treinar um modelo de recomendação.

*Inst. de Computação, UNICAMP, 13083-852 Campinas, SP. g265146@dac.unicamp.br.

- ***Smart Health*:** coletar dados dos pacientes para construir um modelo de recomendação de tratamento ou determinação automática de biomarcador.
- **Detecção de anomalias:** como as informações podem ser coletadas de várias fontes, sua veracidade deve ser verificada. Com isso dito, aprendizado federado pode identificar e prever anomalias e dados perigosos coletados entre os nós

2 Aprendizado Federado

Proposta pelo Google em 2016, o aprendizado federado (FL) é uma técnica que explora a aprendizagem descentralizada e cooperativa entre os nós participantes de uma rede. Inicialmente, FL foi usado em aplicativos relacionados a smartphones, por exemplo, previsão de texto, mas como seu poder de processamento mostrou para ser eficiente, diversas áreas de pesquisa passaram a utilizá-lo. Esse algoritmo melhorou, por exemplo, pesquisas médicas com reconhecimento de imagem. O FL tem como objetivo maximizar a capacidade de computação de ponta (nós não centrais), fazendo com que o processo de aprendizagem possa transferir sem compartilhar informações privadas, e explorar a amostragem aleatória, uma vez que cada nó participante produz e calcula seus dados localmente. Portanto, esse comportamento também promove privacidade de dados não apenas pela posse de dados, mas também pelas restrições de acesso às informações, um aspecto fundamental ao considerar aplicá-lo na Internet das Coisas (IoT). No aprendizado federado, apenas as informações selecionadas, após o processo de treinamento, são compartilhadas com uma unidade central, reduzindo, ainda mais, os custos de comunicação.

As estruturas de aprendizagem federada podem ser classificadas em dois grupos gerais: vertical e horizontais, que estão relacionados à extração de características de dados. Normalmente, o treinamento do aprendizado federado, apresentado na figura 1, consiste em três etapas: iniciação, treinamento e agregação global. Durante o primeiro estágio, uma unidade central (um servidor ou a nuvem) determina as configurações iniciais do treinamento, seleciona os nós participantes e verifica os dados necessários para o processo. Depois de configurações iniciais são definidas, cada unidade de rede recebe um modelo inicial e, em seguida, começa treinamento local com seus dados. Por fim, cada participante da rede compartilha com a unidade central seus parâmetros locais obtidos durante o processo de treinamento, que agregam todos eles, com um algoritmo de agregação específico, para construir um novo modelo. Depois disso, o modelo recém-gerado é mais uma vez redistribuído para os nós para outra execução seguindo a quantidade de rodadas definidas no modelo.

2.1 Problemática

Um grande problema no aprendizado federado é a enorme sobrecarga de comunicação que surge do envio de atualizações do modelo. Ao seguir ingenuamente o protocolo descrito anteriormente, cada cliente participante deve comunicar uma atualização completa do modelo durante cada iteração de treinamento. Cada uma dessas atualizações tem o mesmo tamanho do modelo treinado, que pode estar na faixa de gigabytes para arquiteturas modernas com milhões de parâmetros. Ao longo de várias centenas de milhares de iterações de treinamento em conjuntos de big data, a comunicação total para cada cliente pode facilmente crescer para mais de um petabyte. Consequentemente, se a largura de banda de comunicação for limitada ou a comunicação for cara (ingênua), o aprendizado federado pode se tornar improdutivo ou até mesmo completamente inviável.

Outros problemas apresentados nos artigos são:

- **Redes não confiáveis:** na prática, muitos fatores podem afetar a comunicação e, à medida que o tráfego aumenta, a probabilidade de interrupção da comunicação aumenta. Além disso,

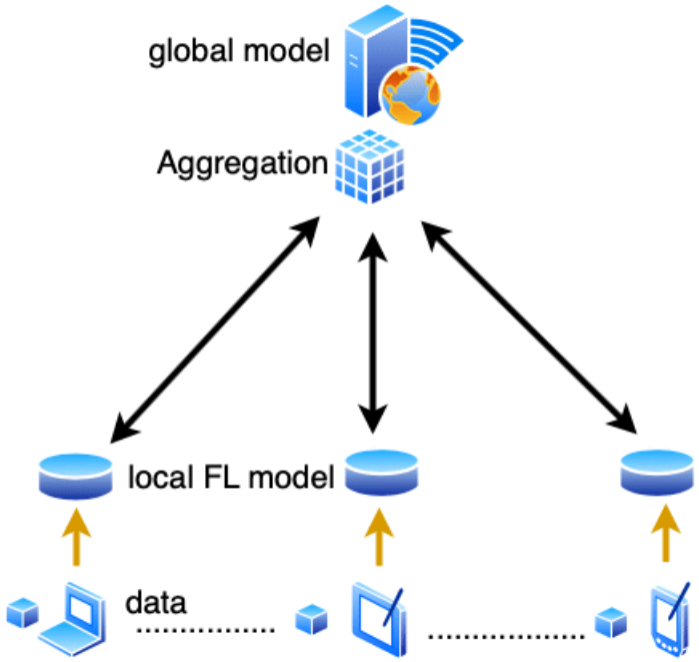

Figura 1: FedAvg

os clientes participantes (incluindo smartphones, laptops, carros autônomos, etc.) podem interromper a transmissão por movimento, esgotamento da bateria ou outros motivos. Infelizmente, a maior parte do trabalho de aprendizagem federado existentes não faz pesquisas aprofundadas sobre esse desafio.

- **Taxa de transferência de rede:** Em algoritmos de aprendizagem federados tradicionais, todos os clientes selecionados enviam seus modelos locais para o mesmo servidor. Esses clientes, geralmente distribuídos em grande escala, causarão pressão de pico na rede. Portanto, requisitos estritos são impostos à taxa de transferência da rede. Além disso, a capacidade de suporte da rede também restringe a escala dos clientes, o que pode fazer com que os novos dados gerados deixem de funcionar como deveriam.

- **Largura de banda dinâmica:** Na largura de banda dinâmica, o aprendizado federado deve estender seu tempo de compartilhamento global para esperar por aqueles clientes com largura de banda baixa, o que diminui a velocidade de treinamento e convergência. Além disso, a propriedade assimétrica das velocidades de conexão à Internet (por exemplo, a velocidade de download no celular nos EUA é de 33,88 Mbps, enquanto a velocidade de upload é de 9,75 Mbps em 2019) também desperdiça uma série de recursos de largura de banda.

- **Convergência teórica:** A distribuição de dados da aprendizagem federada é desequilibrada e não-IID (não independente e idêntica), enquanto a maioria dos métodos de aprendizagem distribuída assume a distribuição IID como pressuposto. Além disso, a maioria dos algoritmos de aprendizagem federados são validados apenas por experimentos e raramente analisam teoricamente os efeitos da rede não confiável e da compressão de comunicação.

3 Trabalhos Realizados

Um desafio a ser considerado no aprendizado federado é a sobrecarga de comunicação que surge devido ao envio periódico de atualizações do modelo. Assim, a seguir são apresentados soluções para compressão de modelos para o aprendizado federado, com objetivo de melhorar a eficiência de comunicação.

3.0.1 Conceitos Utilizados

Os conceitos utilizados nos trabalhos analisados sobre comunicação eficiente em aprendizado distribuído, são:

- **Quantização de gradientes**: os gradientes podem ser representados por precisões de bits menores (i.e. de *float32* para *int8*). Consequentemente, diminuindo em quantidade de informações, em troca de precisão. Com isso tornando o armazenamento e a transmissão ao dos modelos mais simples.
- **Redução de gradientes**: é uma técnica utilizada para deixar a representação dos modelos mais esparsa (i.e., remover valores da matriz de pesos que representa o modelo). Alguns dos métodos utilizados são: $rank_k$ e top_k, onde a $rank_k$ seleciona aleatoriamente k elementos da matriz que representa o modelo, enquanto o método top_k seleciona os k valores de maior magnitude. Dessa forma, os valores selecionados são mantidos, enquanto os outros são excluídos.
- **Compressão de Huffman**: a codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. Uma árvore binária completa, chamada de árvore de Huffman é construída recursivamente a partir da junção dos dois símbolos de menor probabilidade, que são então somados em símbolos auxiliares e estes símbolos auxiliares recolocados no conjunto de símbolos. O processo termina quando todos os símbolos forem unidos em símbolos auxiliares, formando uma árvore binária. A árvore é então percorrida, atribuindo-se valores binários de 1 ou 0 para cada aresta, e os códigos são gerados a partir desse percurso.
- **Codificação Golomb**: é um conjunto de códigos livres de prefixo que podem ser utilizados na compressão de dados em substituição ao código de Huffman, apresentando resultados ótimos para determinadas distribuições de probabilidade dos símbolos codificados Os códigos de Golomb se aplicam a todo número inteiro e não negativo, e dependem de um parâmetro b que deve ser previamente computado para que o código seja adequado aos dados.

3.1 Sign Sthocastic Gradient Descent (Sign SGD)

O treinamento de redes neurais requerem a distribuição do aprendizado entre vários clientes, onde o custo de comunicação de gradientes pode ser um gargalo significativo. O Sign SGD [3] alivia esse problema transmitindo apenas o sinal de cada *minibatch* do gradiente estocástico. Segundos os autores, o algoritmo é tão simples quanto jogar fora o expoente e a mantissa de um número de ponto flutuante de 32 bits.

O sinal do gradiente estocástico é uma maneira tendenciosa de aproximação do gradiente verdadeiro, tornando-o mais difícil de analisar em comparação com o SGD padrão. quando os gradientes são densos ou mais denso que estocasticidade e curvatura, então Sign SGD pode convergir com

uma taxa que tem dependência de dimensão semelhante ou até melhor do que SGD. O único conceito utilizado nesse método é aplicação desse *Sign* e a aplicação de um método de quantização de gradientes.

3.2 Deep Gradient Descent (DGC)

Deep Gradient Compression (DGC) [4] resolve o problema de largura de banda de comunicação comprimindo os gradientes, conforme mostrado na Figura 2. Para garantir que não haja perda de precisão, o DGC emprega correção de *momentum* e recorte de gradiente local no topo da esparsificação para manter o desempenho do modelo. O DGC também usa mascaramento de fator de impulso e treinamento para superar o problema de obsolescência causada pela comunicação reduzida.

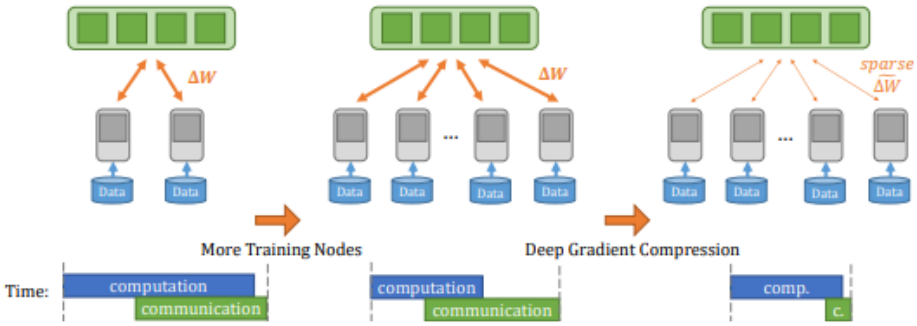

Figura 2: Funcionamento do DGC

3.3 Sparse Ternary Compression (STC)

Muitos trabalhos existentes sobre aprendizado federado adotam modelos com pesos de precisão total, além disso, esses modelos contêm um grande número de parâmetros redundantes, os quais não precisam ser transmitidos ao servidor, consumindo uma quantidade excessiva de custos de comunicação. Para resolver esses problemas, o STC propõe um mecanismo de quantização nos clientes por meio de um fator de quantização de auto-aprendizagem. Entretanto, essa quantização pode reduzir a eficiência dos modelos [1]. Assim, para reduzir esse problema um método ternário é proposto, onde os pesos dos modelos são quantizados para uma representação $\{-1, 0, 1\}$.

A Figura 3 apresenta o funcionamento do mecanismo de quantização ternária. Inicialmente, os pesos do modelo com precisão total normalizados são quantizados para $\{-1, 0, +1\}$. Em seguida, fatores de quantização positivos e negativos são usados para dimensionar os pesos ternários. Por fim, os gradientes calculados são propagados de volta para cada camada.

É importante destacar que ao aplicar a quantização ternária, apenas os pesos com valores diferentes são enviados para o servidor, consequentemente reduzindo ainda mais a comunicação entre dispositivos finais e servidor. Além disso, ao reduzir o custo da comunicação, mais clientes podem ser selecionados para participar da federação, assim melhorando a generalização e eficiência do modelo.

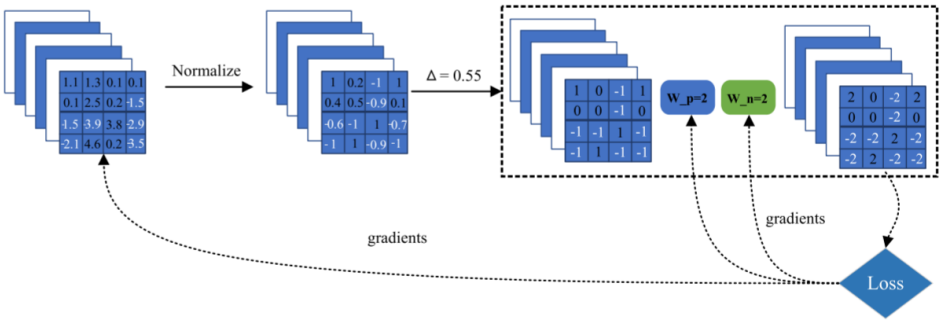

Figura 3: Funcionamento do STC, retirado de [1].

3.4 FedZip

O FedZip [5] é uma solução para compressão de modelos em aprendizado federado composto por três módulos principais, sendo eles: *(i)* redução de gradientes (i.e., poda de modelo); *(ii)* quantização dos pesos; e *(iii)* codificação. Para a redução dos gradientes, o FedZip utiliza o método de top_k, onde um subconjunto dos k maiores valores é selecionado w_k, em seguida, um filtragem dos pesos w_i de cada camada do modelo é feita considerando a seguinte condição:

$$top_k(w) = \{ w_i \, se \; w_i > w_k se \; w_i < w_k$$

Por outro lado, para quantizar o modelo, o FedZip utiliza uma abordagem baseada em agrupamentos com o algoritmo de *k-means*. Segundo os autores, a quantização baseada em agrupamento fornece uma representação mais fiel da distribuição dos pesos para cada camada do modelo, quando comparado com outros métodos de quantização que utilizam um conjunto fixo. A Figura 4 apresenta uma comparação entre a compressão de modelos fornecida pelo FedZip em relação ao FedAvg (solução tradicional sem compressão de modelos). Cada gráfico representa a distribuição dos pesos de cada camada do modelo, assim, como pode ser visto o FedZip apresenta uma representação mais esparsa devido ao agrupamento realizado, assim, consequentemente reduzindo a representação do modelo.

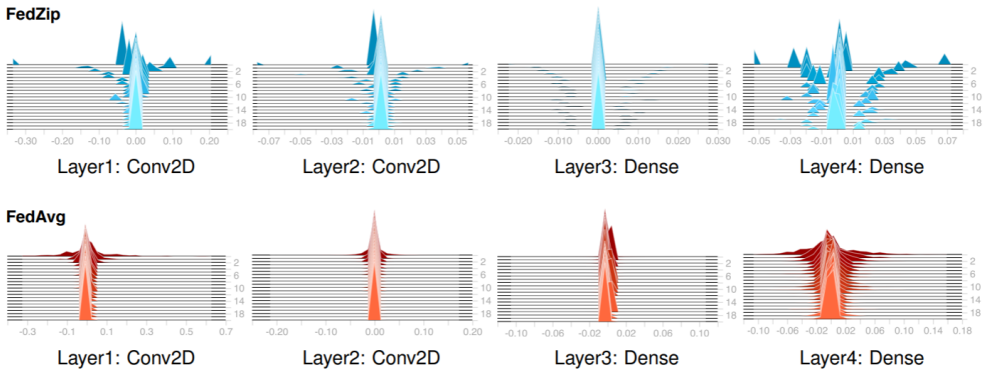

Figura 4: Comparação da redução dos gradientes e quantização aplicados pelo FedZip em relação a solução FedAvg, retirado de [5].

Por fim, o FedZip aplica uma compressão de Huffman para criar uma tabela de endereços. A

compressão de Huffman assimila uma tabela de bits menores para os valores mais frequentes e os bits maiores para os valores menos frequentes. Como foi analisado o k-means, os centroides menos frequentes recebem os maiores bits e os centroides mais frequentes recebem os menores bits. Assim reduzindo ainda mais a representação do modelo para ser enviado para o servidor.

3.5 Modelos Implementados

Após algumas iterações sobre os modelos, foi identificado um repositório publico no GitHub dos modelos de STC, DGC e SGD, sendo todos esses em um mesmo repositório. Nesse repositório, pode ser encontrado diversos tipos de maneiras de instanciar os testes desses modelos de comunicação eficiente, além desses testes padrões, já foi implementado outros modelos, que já serão inclusos nos itens abaixo.

- Base de dados utilizadas
    1. MNIST [6]
    2. FashionMNIST [7]
    3. Cifar-10 (IID e Non-IID)
- Balanceamento dos dados
- Modelo de aprendizado utilizado
    1. Regressão Logistica [8]
    2. *Long Short Term Memory* (LSTM) [9]
    3. CNN
    4. VGG-11
- Quantidade de clientes
- Quantidade de épocas
- Quantidade de iterações

Porém, toda essa execução de métodos e criação de clientes e servidores é feita de maneira iterativa através de codificação com PyTorch. Então ainda deve ser tratado e agregado a transformação dessas maneiras de compressão no ambiente de aprendizado federado a ser utilizado (inicialmente em Flower).

4 Avaliação dos modelos já realizadas

Nesta seção será apresentados as avaliações já realizadas com os métodos até o mês de março.

4.1 Acoplamento dos Modelos de Comunicação Eficiente no Flower

Assim como mencionado previamente, todo o ambiente de processamento foi construído através do PyTorch no repositório disponibilizado pelos autores. Porém os experimentos iniciais realizados pela equipe do Eldorado estão sendo inicialmente aplicados no Flower. Portanto, será feito reuniões com o Alexandre (representante do Eldorado que está desenvolvendo o código), para juntar o modelo criado por eles com os métodos de comunicação eficiente.

Essa parte é essencial para utilizar os métodos analisados, e também serve para ver como esses modelos reagiriam em um ambiente como o Flower, onde a criação do Cliente é feita através de diversas execuções diferentes, diferentemente da maneira iterativa de criação dos clientes realizados através do PyTorch.

Essa parte já foi realizada e está atualmente disponível o código do Cliente atualizado com os modelos de compressão, a única parte não realizada foi a implementação do método de poda, o qual é feito pelo servidor. Porém a codificação da poda está disponível no ambiente do PyTorch.

4.2 Análise do FedZip

Após avaliações sobre o FedZip, foi identificado que esse método, devido sua complexidade, realizava um treinamento muito pesado e demorado sobre as mesmas bases apresentadas anteriormente. Também, diferentemente dos métodos já avaliados anteriormente, ele utiliza de modelos pré-treinados muito pesados e, mesmo alterando esse modelo para um mais leve, ainda consumiu toda a memória RAM disponibilizada pelo Google Colab, ambiente que estava sendo executado todos os modelos anteriores. Portanto, ainda pode ser feito a execução desse método em um ambiente físico (obrigatoriamente Linux), para verificar a eficiência desse modelo.

4.3 Testes utilizando as bases FashionMNIST, MNIST e Cifar-10

As bases de dados do FashionMNIST e MNIST já estavam implementadas no github inicial dos testes realizados pelos autores do artigo sobre o STC. Portanto, elas já estavam de fácil acesso, o arquivo de formato .json controla qual a base de dados a ser utilizada.

Já o Cifar-10, foi implementado manualmente uma versão comum e também uma versão com dados Non-IID. Para utilizar esse algoritmo é necessário somente trocar no mesmo arquivo .json para "Cifar", e para utilizar os dados Non-IID, é necessário abrir o "*Data_utils.py*"e procurar a função chamada "*get_data_loaders*"e colocar o *real_wd* como True.

4.4 Testes utilizando os modelos de Regressão Logistica, LSTM, CNN e VGG-11

Já os modelos implementados como testes iniciais foram os modelos de regressão logística, LSTM, CNN e VGG-11. Todos eles utilizando a maneira de implementação do PyTorch, uma vez que é necessário a utilização dos modelos de PyTorch para aplicar os métodos de compressão.

Todos esses modelos já estavam previamente implementados pelo github. Porém foi feito algumas modificações por questões de versionamento. O algoritmo de LSTM funciona somente para as bases de dados do MNIST e FashionMNIST.

5 Resultados Obtidos

Nessa seção serão apresentados alguns resultados iniciais e exploratórios dos algoritmos de comunicação eficiente mencionados na seção anterior. Todos os resultados apresentado podem ser encontrados no drive: `https://drive.google.com/drive/folders/1rc28gJPNWCH2HJZ5Y92LrClIjLF_lT-a?usp=sharing`

5.1 Especificação dos métodos utilizados

Para os testes exploratórios do repositório dos métodos utilizados, foram feitas análises rodando localmente na minha máquina, utilizando um balanceamento de 0.9 na base de dados. Foi utilizado

a base do MNIST para fazer essas primeiras análises e também foi utilizado a Regressão logística para fazer a parte de aprendizado de máquina. A quantidade de clientes foi variada entre 10, 100, 1000, 2000, 3000, 4000 e 5000 clientes e a quantidade de iterações que são feitas entre o servidor e os clientes foi de 500.

```
"main" :
[
{
    "dataset" : ["mnist"],
    "net" : ["logisticregression"],

    "iterations" : [500],

    "n_clients" : [10,100,1000,2000,3000,4000,5000],
    "participation_rate" : [1],
    "classes_per_client" : [10],
    "batch_size" : [1],
    "balancedness" : [0.9],

    "momentum" : [0.9],


    "compression" : ["none","stc","dgc","signsgd"],

    "log_frequency" : [1],
    "log_path" : ["results/"]
}
]
}
```

Figura 5: Exemplo da configuração

5.2 Resultados da Quantidade de Bits enviados

A Figura 6 representa os resultados iniciais demonstrando a quantidade de bits que são enviados com 1000 clientes em 500 iterações de código.

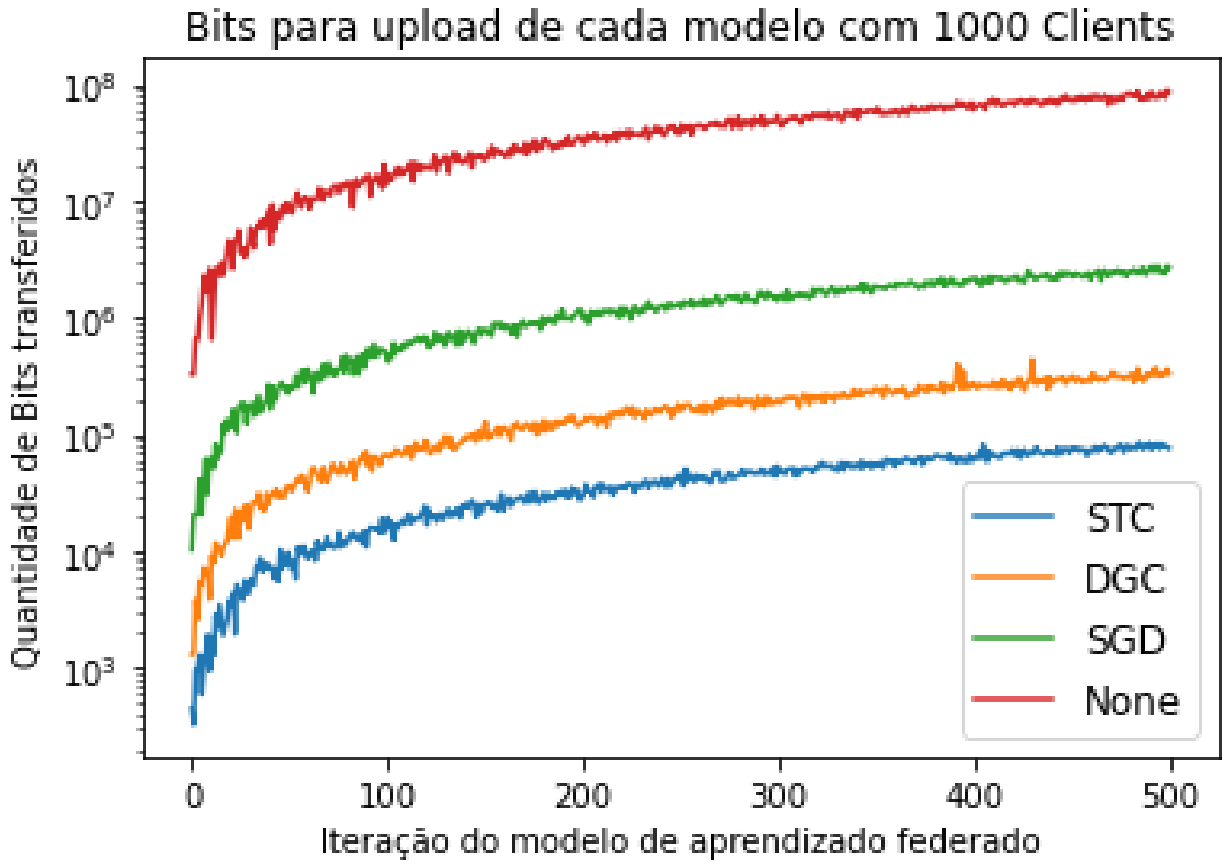

Figura 6: Quantidade de bits enviados com 1000 Clientes

5.3 Resultados do Tempo de Processamento Total do Código

A Figura 7 representa o tempo total de processamento de todo o código (não só da parte de compressão de modelos), variando pela quantidade de clientes.

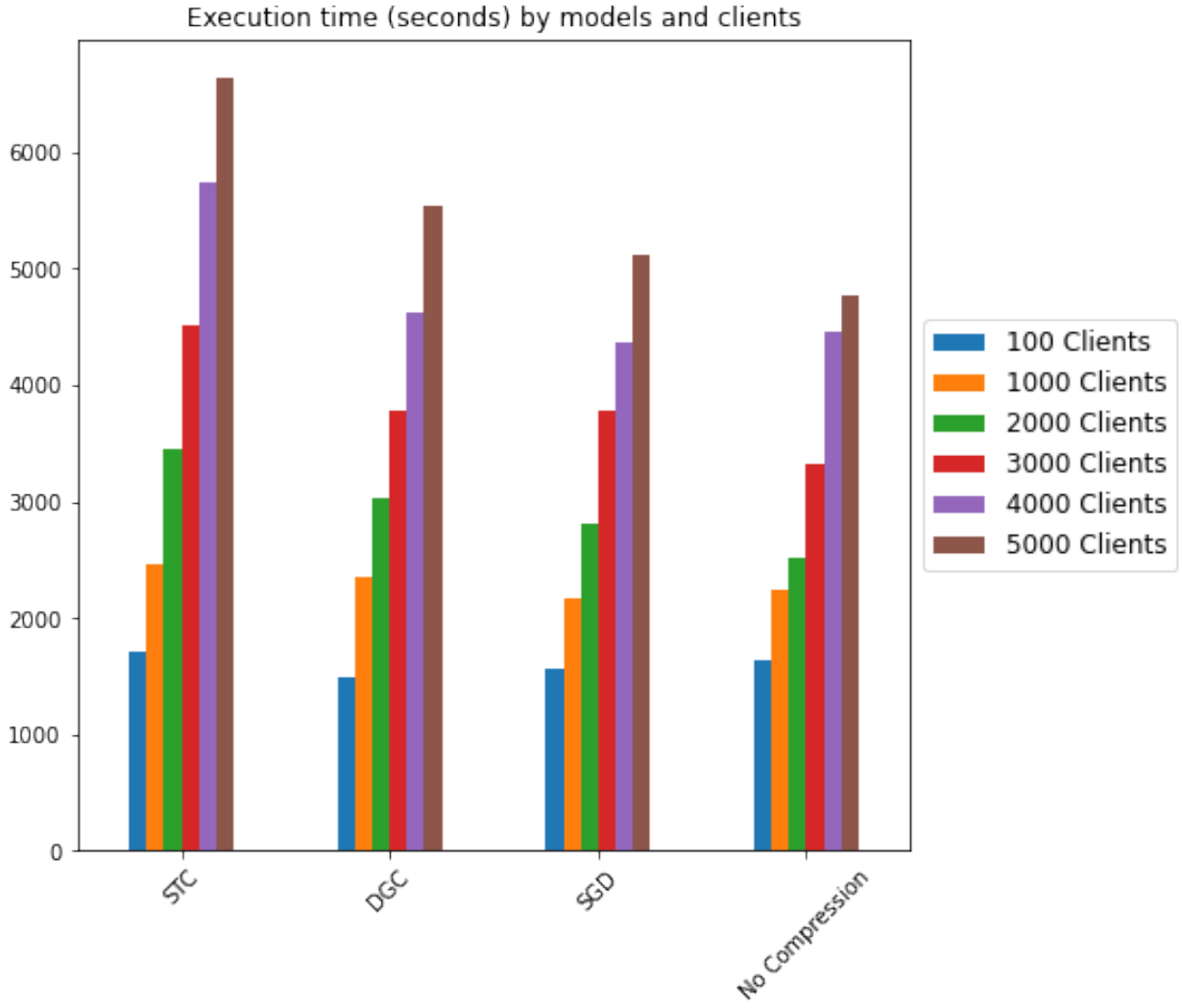

Figura 7: Tempo de execução de todos os métodos

5.4 Resultados da Utilização de Energia do Código

Utilizando uma biblioteca chamada PyJoules, foi feita a avaliação do custo de energia do código inteiro (não só da parte de compressão de modelos), utilizando uma máquina física para verificar o custo energético em watts por hora. A Figura 8 representa esse custo energético.

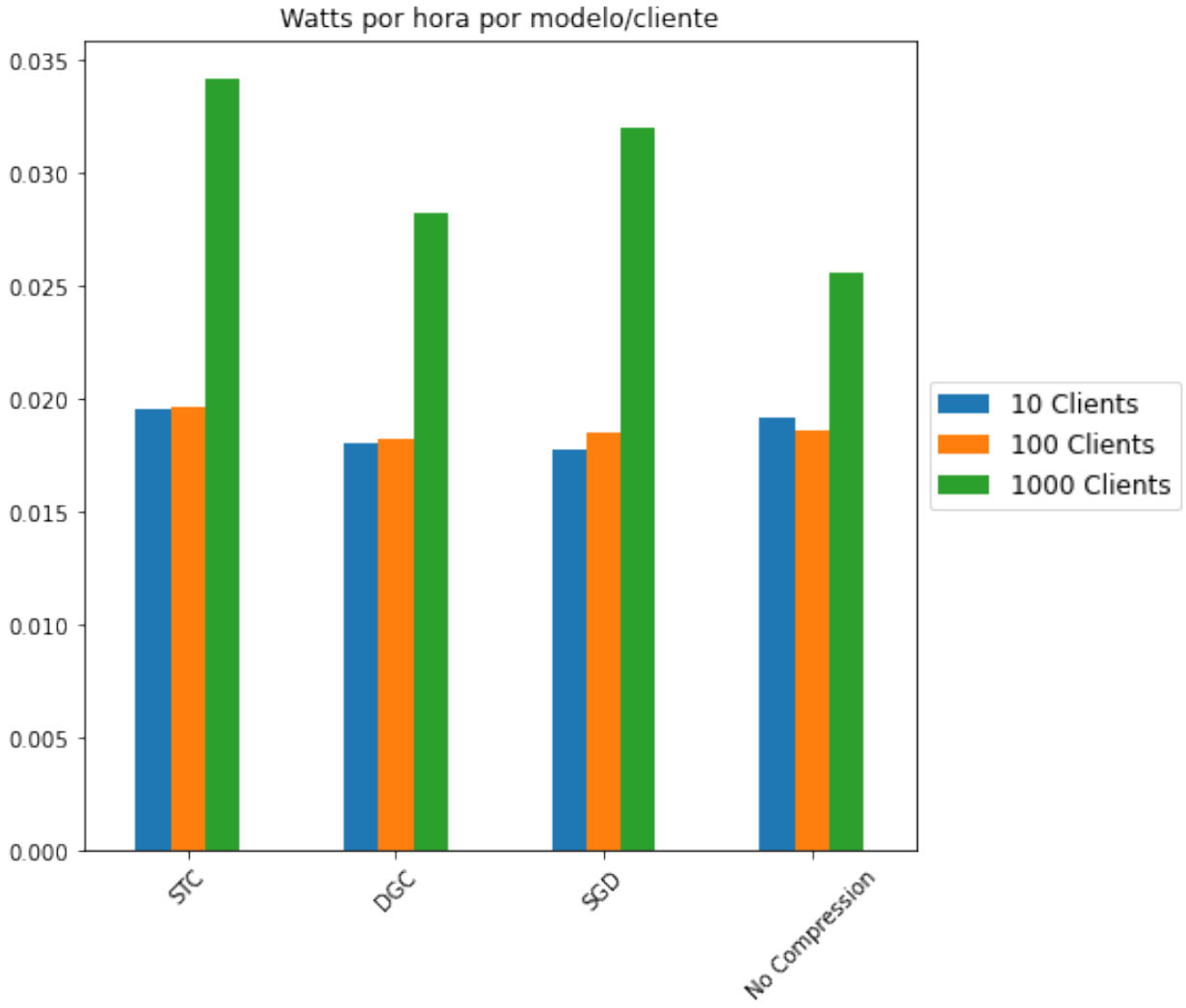

Figura 8: Gasto energético de todos os métodos

5.5 Resultados da Acurácia e Precisão dos Modelos

Um fator a ser avaliado é se os métodos de compressão afetam a acurácia e precisão dos modelos, então através da Figura 9 podemos ver que para esses resultados iniciais podemos concluir que os resultados não são muito afetados pela compressão.

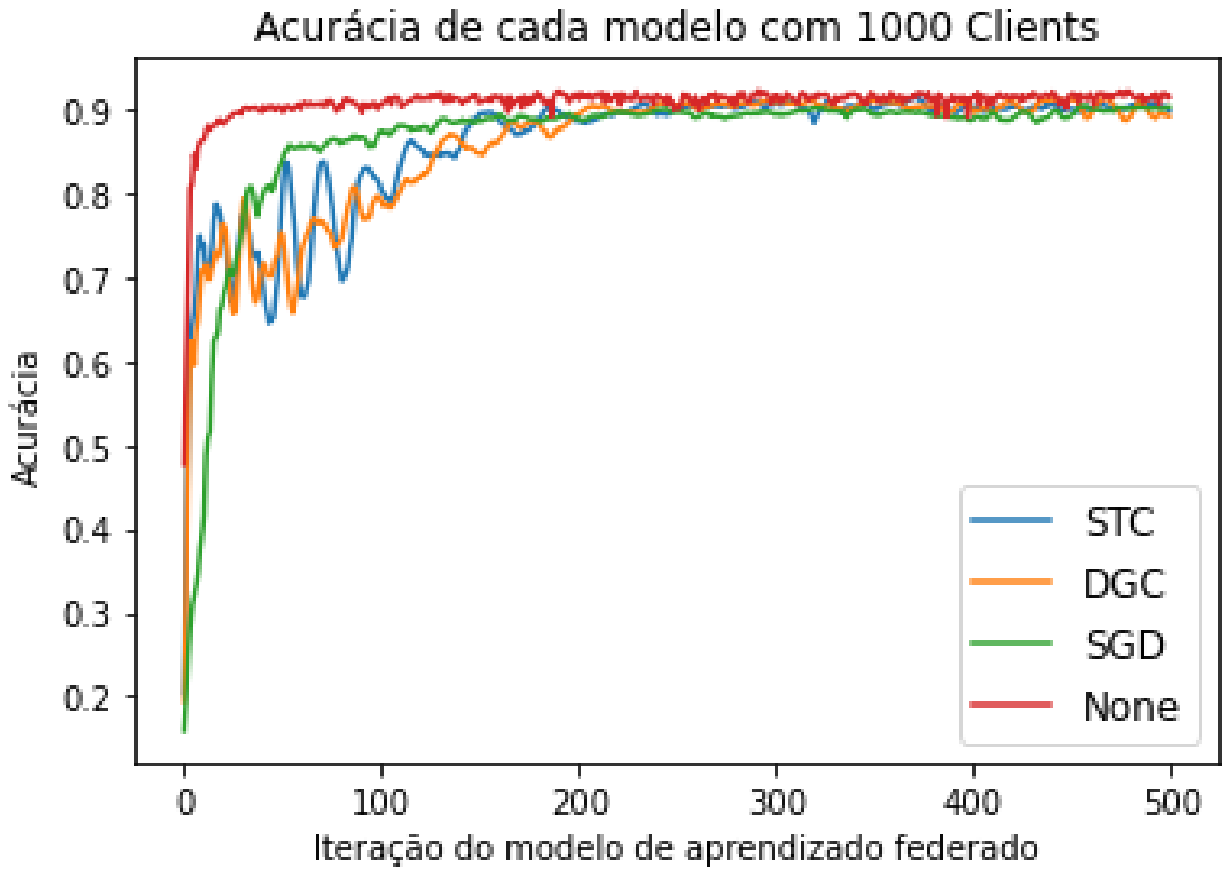

Figura 9: Acurácia dos modelos com 1000 Clientes

5.6 Tabela Comparativa

A Tabela 10 mostrando brevemente uma comparação entre os resultados apresentados pelos modelos, demonstrando algumas das métricas que devem ser analisadas. Toda a tabela foi aplicada em 1000 Clientes e em 500 iterações do MNIST.

Métodos	Base de dados	Taxa de compressão	Rota de compressão	Acurácia do modelo	Tempo de processamento	Quantidade de watts por hora com 1000 Clientes
Sem compressão	MNist	1x	---	0.9225	37:20 minutos	0.0256 W/h
STC	MNist	1021x	Bidirecional	0.9145	40:50 minutos	0.0341 W/h
SignSGD	MNist	32x	Upload	0.9045	36:15 minutos	0.0319 W/h
DGC	MNist	259x	Bidirecional	0.914	39:15 minutos	0.0281 W/h
FedZip	---	---	---	---	---	---

Figura 10: Tabela Comparativa

5.7 Resultados da Acurácia e Precisão dos Modelos Utilizando Todos os Modelos no Cifar-10 (Non-IID)

A última avaliação foi a de avaliar a base de dados do Cifar-10, pois a mesma possuí uma maior replicabilidade em modelos de aprendizado federado e é um contexto diferente em qualidade imagens, variando em matrizes de imagens coloridas, diferentemente do MNIST, que são em escalas de cinza.

Essa avaliação foi realizada utilizando não só a CNN e Regressão Logística, mas também com o modelo de VGG-11 implementado manualmente, uma vez que ele tem mais parâmetros que os modelos utilizados previamente. Portanto, será verificado o comportamento do sistema, quando utilizado um modelo com uma maior quantidade de parâmetros que os modelos mais básicos de CNN e Regressão Logística.

Nas figuras 11, 12 e 13 podemos ver a comparação entre cada um desses métodos e quais foram suas acurácias e gastos computacionais.

5.8 Resultados da Acurácia e Precisão dos Modelos Utilizando Métodos de Pruning na CNN

Por último, foi avaliado um método de Pruning para tentar diminuir a carga de processamento dos clientes do ambiente de aprendizado federado, pois, uma vez que existem menos parâmetros nos neurônios, o custo computacional é menor.

Portanto, foi utilizado de um próprio método de Pruning do PyTorch chamado L1Unstructured, para fazer essa poda no modelo de CNN, utilizando como parâmetros 20% e 40% de poda.

Podemos ver que a acurácia dos modelos varia bastante utilizando esse método, fazendo com que tenha que ser avaliado uma análise mais profunda da melhor porcentagem de poda que menos impacta na acurácia do modelo.

Outro fator que vemos é que o tempo de processamento aumenta, uma vez que essa poda é feita no servidor, gastando um bom tempo para replicar o modelo e realizar a poda, porém é esperado que os clientes tenham uma maior velocidade com menos parâmetros.

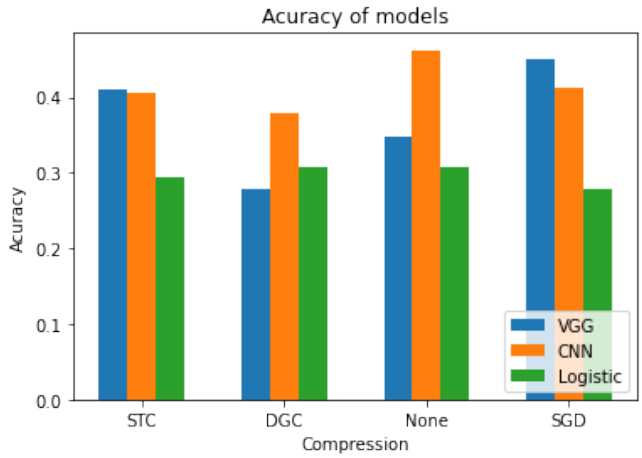

Figura 11: Acurácia dos modelos utilizando Cifar-10 Non-IID

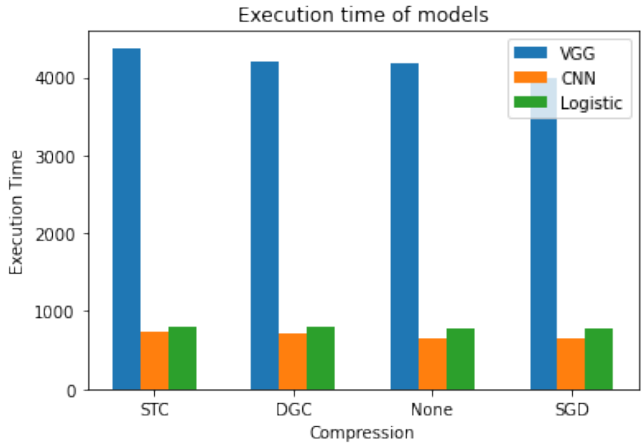

Figura 12: Tempo de processamento dos modelos utilizando Cifar-10 Non-IID

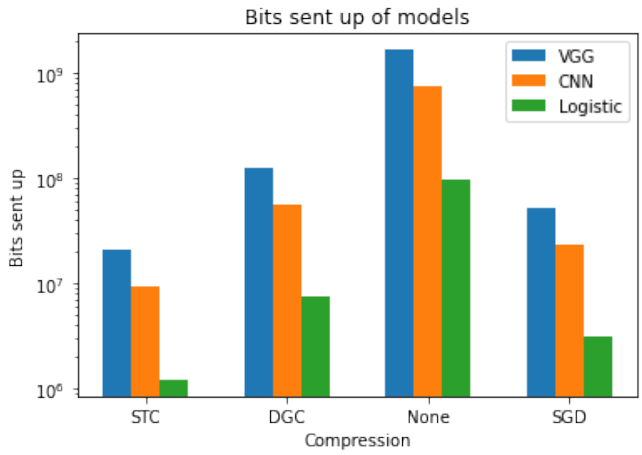

Figura 13: Quantidade de bits enviados dos modelos utilizando Cifar-10 Non-IID

Por último, podemos verificar que a quantidade de bits enviados pelos clientes acaba sendo a mesma, mesmo com diferentes variações das bases de dados em cada execução isso não deveria acontecer. Portanto, deve ser analisado o código que calcula essa quantidade de bits, pois pode estar defasada, ou pode já ser feito essa avaliação pelo ambientes de processamento já produzidos no projeto.

Nas figuras 14, 15 e 16 podemos ver essa avaliação mais claramente.

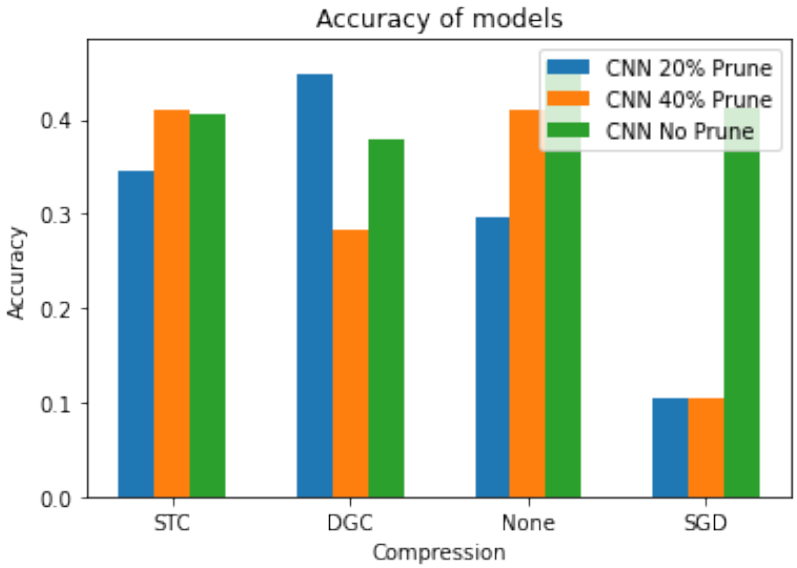

Figura 14: Acurácia dos modelos utilizando Poda

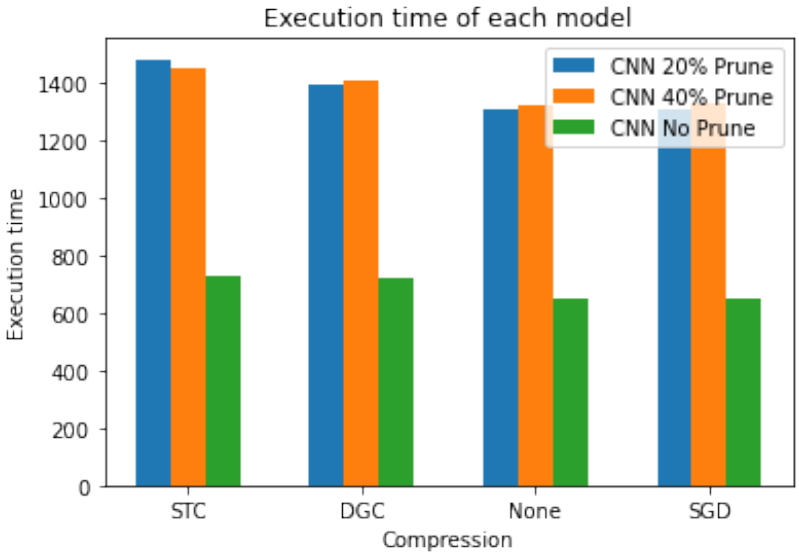

Figura 15: Tempo de processamento dos modelos utilizando Poda

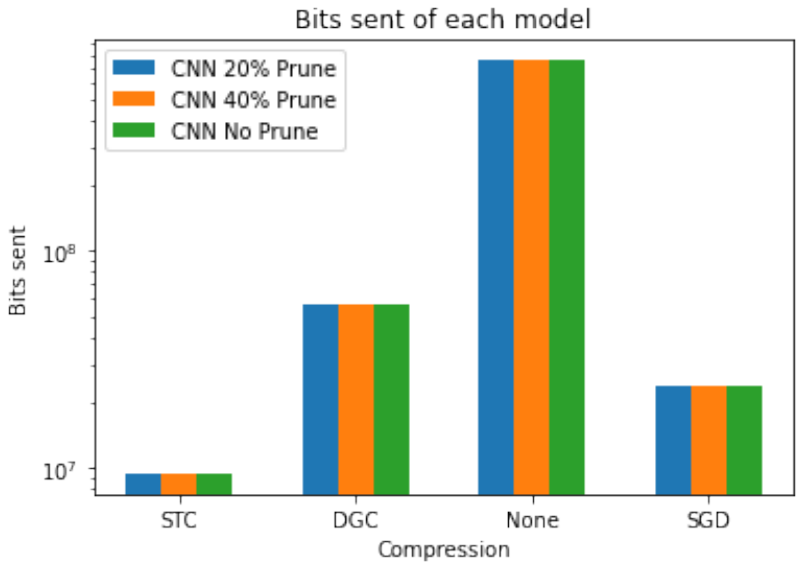

Figura 16: Quantidade de bits enviados dos modelos utilizando Poda

6 Lições Aprendidas

Apesar de existirem diversas métricas de redes para se analisar se o método de compressão é bom ou não, os mais analisados pelos artigos e os principais focos, são a rota de compressão e se utiliza ou não de algum tipo de técnica auxiliar além dos próprios algoritmos de compressão. Isso, porque, uma vez que smartphones e dispositivos de baixo custo não possuem um poder de processamento tão bom, é necessário que essas técnicas auxiliares podem executar normalmente nesses tipos de dispositivo. Já para a rota de compressão é muito necessário saber onde está o gargalo, se está presente na comunicação cliente-servidor ou servidor-cliente.

Outro fator em comum nos diversos artigos, é que a maioria deles analisam já esses algoritmos pensando em dados provenientes de smartphones, isso devido que o algoritmo de aprendizado federado é principalmente utilizado nesse contexto de smartphones e IoT (sensores de baixo custo).

Os principais algoritmos estudados nessa fase inicial foram os do STC, SGD, DGC e FedZip. Devido à grande complexidade e custo computacional o FedZip foi descartado, porém se for possível fazer uma redução no custo dele, seria de grande interesse, uma vez que nos artigos avaliados ele possuí o melhor desempenho em aprendizado federado.

Dentre os métodos de compressão implementados, o que possuí uma melhor acurácia geral foi o modelo do STC, porém ele tem uma complexidade computacional maior que os outros modelos, fazendo ele ter um custo mais elevado para o cliente. Já o método com menor custo foi identificado como o SGD.

Referências

[1] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3400–3413, 2020.

[2] Amiehossein Maleki, Mohammad Fadaeieslam, Hanieh Malekijou, Morteza Homayounfar, Farshid Alizadeh, and Reza Rawassizadeh. Fedzip: A compression framework for communication-efficient federated learning, 02 2021.

[3] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: compressed optimisation for non-convex problems. 02 2018.

[4] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. 2018.

[5] Amirhossein Malekijoo, Mohammad Javad Fadaeieslam, Hanieh Malekijou, Morteza Homayounfar, Farshid Alizadeh-Shabdiz, and Reza Rawassizadeh. Fedzip: A compression framework for communication-efficient federated learning, 2021.

[6] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[7] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[8] Raymond E Wright. Logistic regression. 1995.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.