ARTHUR FERNANDES SCANONI
ROBERTO DE MEDEIROS FARIAS FILHO
(Autores. Disciplina IN1098-2025.2, Prof. Geber Ramalho)

Uso de sistemas éticos-inteligentes: *Differential Privacy* e *Federated Learning*

RECIFE
2025

1. **Uso de sistemas éticos-inteligentes: Differential Privacy e Federated Learning**

(Título com tema focado no curso)

2. **Resumo:**

Escolhido: "Segurança: Escolher problema, dados e ferramentas, desenhar experimentos, escrever guia (vídeo, página web, documento) com passo a passo."

3. **Problema de Pesquisa:**

Qual seria o meu problema de Pesquisa?

(Fazer em forma de pergunta, contextualização do tema, depois fazer a pergunta, define-se quantitativa ou qualitativa)

4. **Hipóteses:**

- **H0:** Não seriam os modelos escolhidos aplicáveis (traçar gráfico): Hipótese negada, como visto nas análises do documento Júpiter (IPYNB).
- **H1.1:** Seria o Differential Privacy aplicável? Sim. Como comprovado no IPYNB do projeto específico ao DP.
- **H1.2:** Seria o Federated Learning aplicável? Sim. Como comprovado no IPYNB do projeto de FL.

5. **Objetivos:**

Estudar ética no contexto do tema “Segurança”. Abordagens do Differencial Privacy (DP) e Federated Learning (FL). Gerar: vídeo, página web (GitHub), documento com passo a passo (apêndice deste).

- Estudar o Differential Privacy;
- Aplicar o DP em Python;

- Estudar o Federated Learning;
- Aplicar o FL em Python;
- Deixar documentos com professor da disciplina e, dicionalmente, online (GitHub), com intuito de alunos poderem estudar e acessar maneira de aplicar os estudos;
- Elaborar exercícios didáticos de forma a testar se aluno absorveu o conhecimento, nos critérios:
    i. Questão para medir a capacidade de uso da ferramenta;
    ii. Questão para desafiar e testar capacidade de evoluir a ferramenta;
- Gerar vídeo, página web (GitHub), documento com passo a passo (Apêndices A e B deste documento de projeto).

6. Justificativa:

A aprendizagem das técnicas e seu uso abre leque para maior segurança cibernética (*Cybersecurity*) por dar exemplo de projetos que podem ser evoluídos e usados pelos alunos em seu trabalho; atém mesmo o fato de conhecer e ter visto codificação com capacidade. O Federated Learning permite identificar usuários mal-intencionados em uma rede distribuída através do Flower. O Differential Privacy dá oportunidade de capturar dados dos usuários, escondendo (com ruídos) suas particularidades (privacidades) através do Opacus.

7. Metologia:

EstudoForam seguidos alguns passos:

Estudo dos websites e especificações das ferramentas: i) Flower; ii) Opacus.

Aplicação da ferramenta estudada em alguma linguagem e acessível aos alunos: i) Python.

8. Referencias:

Durante os documentos Jupyter, foram apresentados conteúdos (de cunbho de revisão bibliográfica) sobre os temas e suas devidas referências, que são:

DWORK, C.; MCSHERRY, F.; NISSIM, K.; SMITH, A. **Calibrating Noise to Sensitivity in Private Data Analysis**. In: Proceedings of the 3rd Theory of Cryptography Conference (TCC 2006), Lecture Notes in Computer Science, vol. 3876, pp. 265–284. Springer. 2006.

DWORK, C. & ROTH, A. **The Algorithmic Foundations of Differential Privacy**. Foundations and Trends in Theoretical Computer Science, 9(3–4), 211–407. 2014.

ALZOUBI, Y. I. & MISHRA, A. **Differential privacy and artificial intelligence: potentials, challenges, and future avenues**. EURASIP Journal on Information Security, 2025(18). 2025.

ABUAH, Chike. **Automatic Proofs of Differential Privacy**. Cybersecurity Insights a NIST blog. Publicado em 22 de julho de 2021. Disponível em https://www.nist.gov/blogs/cybersecurity-insights/automatic-proofs-differential-privacy. Acesso em 01 de dez de 2025.

ABOWD, J. M. **The U.S. Census Bureau Adopts Differential Privacy**. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2867–2867. 2018.

APPLE. Differential Privacy Overview. Apple Machine Learning Journal, Vol. 1. 2017.

ERLINGSSON, Ú.; PIHUR, V.; KOROLOVA, A. **RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response**. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1054–1067. 2014.

DANKAR, F. K.; EL EMAM, K. **The application of differential privacy to health data**. Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society (WPES ’12), pp. 123–132. 2012.

YOUSEFPOUR, A; SHILOV, I; SABLAYROLLES, A; TESTUGGINE, D; PRASAD, K; MALEK, M; NGUYEN, J; GHOSH, S; BHARADWAJ, A; ZHAO, J; CORMODE, G; MIRONOV, I. **Opacus: User-Friendly Differential Privacy Library in PyTorch**. arXiv preprint arXiv:2109.12298. 2021.

SWAPNA. **Convolutional Neural Network | Deep Learning**. Developers Breach. Publicado em 21 de agosto de 2020. Disponível em https://developersbreach.com/convolution-neural-network-deep-learning/. Acesso em 08 de dez de 2025.

CHRISLB. **File:MultiLayerNeuralNetwork english.png**. CC BY-SA 3.0 <http://creativecommons.org/licenses/by-sa/3.0/>, via Wikimedia Commons. Disponível em https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetwork_english.png. Acesso em 09 de dez de 2025.

MCMAHAN, B; MOORE, E; RAMAGE, D; HAMPSON, S Y; ARCAS, B. A. **Communication-Efficient Learning of Deep Networks from Decentralized Data**. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), PMLR 54, pp. 1273-1282. DOI/URL: http://proceedings.mlr.press/v54/mcmahan17a.html . 2017.

KAIROUZ, P, et al. **Advances and Open Problems in Federated Learning**. Foundations and Trends® in Machine Learning, 14(1–2), 1–210. DOI/URL: https://federated.withgoogle.com/ . 2021.

BEUTEL, D J; TOPAL, T; MATHUR, A; QIU, X; PARCOLLET, T; LANE, N D. **Flower: A Friendly Federated Learning Framework**. The Flower Authors. Disponível em: https://flower.ai . Acesso em 16 de dez de 2025.

ANEXO A: Passo-a-passo do Federated Learning

Vide vídeo sobre FL no Readme.md do GitHub https://github.com/rdmff/IN1098RobertoArthur/ ou em https://drive.google.com/file/d/1eWy_pzRmufcCAdESJsww6VgZV9pBFvjU/ .

Atividades:

Tal qual no documento

```
# Imports necessários
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
import torchvision
import torchvision.transforms as transforms

import numpy as np
from collections import Counter, defaultdict

# Seed para reprodutibilidade
SEED = 42
torch.manual_seed(SEED)
np.random.seed(SEED)

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"🔧 Usando device: {device}")

# Configurações
NUM_CLIENTS = 3
NUM_CLASSES = 10  # CIFAR-10 tem 10 classes
UNDERSAMPLE_COUNT = 50  # Número de amostras para classes com undersample
NORMAL_SAMPLES_PER_CLASS = 800  # Número de amostras para classes normais

# Classes do CIFAR-10
CIFAR10_CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']
```

```
print(f"✅ Configurações definidas:")
print(f"   - Número de clientes: {NUM_CLIENTS}")
print(f"   - Número de classes: {NUM_CLASSES}")
print(f"   - Undersample count: {UNDERSAMPLE_COUNT}")
print(f"   - Normal samples per class: {NORMAL_SAMPLES_PER_CLASS}")
```

```
🩹 Usando device: cuda
✅ Configurações definidas:
  - Número de clientes: 3
  - Número de classes: 10
  - Undersample count: 50
  - Normal samples per class: 800
```

1. Carregamento de Dados

Carregamos o dataset CIFAR-10 com as transformações necessárias.

```
# Transformações para os dados
transform = transforms.Compose([
  transforms.ToTensor(),
  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

# Carregar datasets
print("📥 Carregando CIFAR-10...")
trainset = torchvision.datasets.CIFAR10(
  root='./data',
  train=True,
  download=True,
  transform=transform
)

testset = torchvision.datasets.CIFAR10(
  root='./data',
  train=False,
  download=True,
  transform=transform
)

print(f"✅ Dataset de treino carregado: {len(trainset)} amostras")
print(f"✅ Dataset de teste carregado: {len(testset)} amostras")
print(f"✅ Classes: {CIFAR10_CLASSES}")
```

```
📥 Carregando CIFAR-10...
Files already downloaded and verified
Files already downloaded and verified
✅ Dataset de treino carregado: 50000 amostras
✅ Dataset de teste carregado: 10000 amostras
✅ Classes: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
```

2. Separação de Clientes

Criamos 3 clientes, onde cada cliente possui dados com undersampling para 2 labels específicas.

```
def create_non_iid_clients(dataset, num_clients, num_classes,
                undersample_count=50, normal_samples=800):
  """
     Cria divisão não-IID dos dados onde cada cliente tem 2 classes com undersample.

  Args:
    dataset: Dataset PyTorch
    num_clients: Número de clientes (3)
    num_classes: Número de classes (10 para CIFAR-10)
    undersample_count: Número de amostras para classes com undersample
    normal_samples: Número de amostras para classes normais

  Returns:
    Lista de datasets (um por cliente) e informações sobre distribuição
  """
  # Organizar índices por classe
  class_indices = defaultdict(list)
  for idx in range(len(dataset)):
    _, label = dataset[idx]
    class_indices[label].append(idx)

  # Embaralhar índices de cada classe
  for class_id in class_indices:
    np.random.shuffle(class_indices[class_id])

  client_datasets = []
  client_info = []
  used_indices = set()

  # Definir quais 2 classes terão undersample para cada cliente
  # Cliente 0: classes 0 e 1 com undersample
  # Cliente 1: classes 2 e 3 com undersample
  # Cliente 2: classes 4 e 5 com undersample
  undersample_classes_per_client = [
    [0, 1],  # Cliente 0
    [2, 3],  # Cliente 1
    [4, 5]   # Cliente 2
  ]

  # Para cada cliente
  for client_id in range(num_clients):
    client_indices = []
    client_dist = {}

    # Classes com undersample para este cliente
```

```
    undersample_classes = undersample_classes_per_client[client_id]

    # Para cada classe
    for class_id in range(num_classes):
      # Determinar quantas amostras pegar
      if class_id in undersample_classes:
        samples_to_take = undersample_count
      else:
        samples_to_take = normal_samples

      # Pegar amostras não usadas
          available = [idx for idx in class_indices[class_id] if idx not in
used_indices]
      selected = available[:samples_to_take]

      client_indices.extend(selected)
      used_indices.update(selected)
      client_dist[class_id] = len(selected)

    # Criar subset para o cliente
    client_dataset = Subset(dataset, client_indices)
    client_datasets.append(client_dataset)
    client_info.append({
      'distribution': client_dist,
      'undersample_classes': undersample_classes,
      'total_samples': len(client_indices)
    })

  return client_datasets, client_info

# Criar divisão dos clientes
print("🔄 Criando divisão de clientes...\n")
client_trainsets, client_info = create_non_iid_clients(
  trainset,
  NUM_CLIENTS,
  NUM_CLASSES,
  undersample_count=UNDERSAMPLE_COUNT,
  normal_samples=NORMAL_SAMPLES_PER_CLASS
)

# Imprimir informações de cada cliente
print("="*80)
print("📋 DISTRIBUIÇÃO DOS DADOS POR CLIENTE")
print("="*80)

for client_id, info in enumerate(client_info):
  print(f"\n👤 Cliente {client_id + 1}:")
  print(f"   Total de amostras: {info['total_samples']}")
          undersample_class_names    =    [CIFAR10_CLASSES[c]    for    c    in
info['undersample_classes']]
  print(f"   Classes com UNDERSAMPLE: {undersample_class_names}")
  print(f"   Distribuição por classe:")
```

```
  for class_id in range(NUM_CLASSES):
    count = info['distribution'][class_id]
     marker = "⚠ UNDERSAMPLE" if class_id in info['undersample_classes'] else
""
        print(f"         {CIFAR10_CLASSES[class_id]:12s}:  {count:4d}  amostras
{marker}")

print("\n" + "="*80)
🔄 Criando divisão de clientes...

================================================================================
=
📊 DISTRIBUIÇÃO DOS DADOS POR CLIENTE
================================================================================
=

📱 Cliente 1:
  Total de amostras: 6500
  Classes com UNDERSAMPLE: ['airplane', 'automobile']
  Distribuição por classe:
    airplane    :   50 amostras ⚠ UNDERSAMPLE
    automobile  :   50 amostras ⚠ UNDERSAMPLE
    bird        :  800 amostras
    cat         :  800 amostras
    deer        :  800 amostras
    dog         :  800 amostras
    frog        :  800 amostras
    horse       :  800 amostras
    ship        :  800 amostras
    truck       :  800 amostras

📱 Cliente 2:
  Total de amostras: 6500
  Classes com UNDERSAMPLE: ['bird', 'cat']
  Distribuição por classe:
    airplane    :  800 amostras
    automobile  :  800 amostras
    bird        :   50 amostras ⚠ UNDERSAMPLE
    cat         :   50 amostras ⚠ UNDERSAMPLE
    deer        :  800 amostras
    dog         :  800 amostras
    frog        :  800 amostras
    horse       :  800 amostras
    ship        :  800 amostras
    truck       :  800 amostras

📱 Cliente 3:
  Total de amostras: 6500
  Classes com UNDERSAMPLE: ['deer', 'dog']
  Distribuição por classe:
    airplane    :  800 amostras
    automobile  :  800 amostras
    bird        :  800 amostras
```

```
   cat          :  800 amostras
   deer         :   50 amostras ⚠ UNDERSAMPLE
   dog          :   50 amostras ⚠ UNDERSAMPLE
   frog         :  800 amostras
   horse        :  800 amostras
   ship         :  800 amostras
   truck        :  800 amostras

================================================================================
=
```

3. Federated Learning

Implemente aqui o código de Federated Learning usando o framework Flower.

Os dados já estão preparados:

- `client_trainsets`: Lista com 3 datasets, um para cada cliente
- `testset`: Dataset de teste para avaliação
- `client_info`: Informações sobre a distribuição de dados de cada cliente

Dicas:

- Use `flwr` (Flower) para implementar o federated learning
- Cada cliente deve treinar localmente com seu dataset (`client_trainsets[client_id]`)
- O servidor deve agregar os modelos dos clientes
- Use `testset` para avaliar o modelo global

```
# ================================================================================
# IMPLEMENTE AQUI O CÓDIGO DE FEDERATED LEARNING
# ================================================================================

print("📝 Implemente o código de Federated Learning acima!")
```

```
📝 Implemente o código de Federated Learning acima!
```

ANEXO B: Passo-a-passo do Differential Privacy

O DP de exemplo se encontra no endereço a seguir:

https://github.com/rdmff/IN1098RobertoArthur/tree/main/DifferentialPrivacy

Após acessar o endereço, siga as etapas:

1. Busque um arquivo, com maior versão, nomeado:

DifferentialPrivacy_vX.Y.beta.ipynb

2. Abra no Jupyter Notebook do Anaconda ou Plataforma que carregue o formato que desejar.

3. Observe o bloco de imports e instale as dependências faltantes, inclusive Opacus:

pip install opacus==1.3.0

4. Execute os blocos de código (apenas os que têm código Python, ignorando os de texto/Markdown). Execute-os de cima para baixo, um após o outro.

5. O documento separa os testes em duas partes:

- **A) Baselines:** são as redes neurais, comuns, usadas – neste caso Convolutional Network. Observar gráficos e resultados.

- **B) DP:** parte onde o sistema irá adicionar ruído branco nos dados, escondendo informações pessoais dos dados originais. Vai gerar alguns gráficos, onde a precisão (*acuracy*) perdida é pouca, devido ao Opacus e ocultamento das informações do cliente. Observar gráficos e resultados.

Exercícios:

1) altere alguns parâmetros da seção de código “Ajustes de parâmetros” da parte B e execute novamente todos os passos, gerando novos gráficos e resultados. Altere: batch_size_dp para *32*, max_grad_norm para *1.5* e a privacidade para “*média*”.

2) Desafio. Dada a rede neural residual, abaixo, com padrões similares aos das anteriores, adicione esta e plote gráfico e resultados com as demais e o DP.

Arquivo, também, no repositório do GitHub com nome “**resnet.py**”:

“

```
# ResNet Block
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
         self.conv1 = nn.Conv2d(in_channels, out_channels, 3, stride, padding=1,
bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
            self.conv2 = nn.Conv2d(out_channels, out_channels, 3, 1, padding=1,
bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, 1, stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        return F.relu(out)

class SimpleResNet(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(3, 64, 3, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(64, 2, stride=1)
        self.layer2 = self._make_layer(128, 2, stride=2)
        self.layer3 = self._make_layer(256, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(256, num_classes)
```

```
    def _make_layer(self, out_channels, blocks, stride):
        strides = [stride] + [1]*(blocks-1)
        layers = []
        for stride in strides:
            layers.append(ResidualBlock(self.in_channels, out_channels, stride))
            self.in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        return self.fc(x)
```

“