import torch  
import torch.nn as nn  
import copy  
from torch.distributions import Laplace  
  
  
def add\_laplace\_noise(tensor, epsilon, sensitivity=0.5, is\_training=True):  
if not is\_training:  
 # No noise added during testing  
 return tensor  
  
 scale = sensitivity / epsilon  
 noise = Laplace(loc=0.0, scale=scale).sample(tensor.shape)  
 device = tensor.device # Get the device of the tensor  
  
 # Ensure noise is on the same device as tensor  
 noise = noise.to(device)  
 return tensor + noise  
  
  
def FedAvgDP(w, epsilon=1e-8, sensitivity=0.5, is\_training=True):  
 *"""  
 Federated Averaging with Differential Privacy (DP) during training.  
  
 Parameters:  
 w: List of model weights from clients.  
 epsilon: The privacy budget for differential privacy.  
 sensitivity: The sensitivity of the model parameters.  
 is\_training: Flag to indicate whether it's training or testing.  
  
 Returns:  
 w\_avg: The aggregated model weights with noise during training.  
 """* w\_avg = copy.deepcopy(w[0])  
  
 # Aggregate model updates  
 for k in w\_avg.keys():  
 for i in range(1, len(w)):  
 w\_avg[k] += w[i][k]  
 w\_avg[k] = torch.div(w\_avg[k], len(w))  
  
 # Add Laplace noise only during training  
 if is\_training:  
 w\_avg[k] = add\_laplace\_noise(w\_avg[k], epsilon, sensitivity, is\_training)  
  
 return w\_avg