Experiment: PneumoniaMNIST dataset 50 runs (code only)

```
import random
import numpy as np
import math
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from torch.utils.data import DataLoader, Subset, TensorDataset
import torch
import torch.nn as nn
import torch.nn.functional as F
from tqdm import tqdm
from torchvision.transforms import ToTensor
from medmnist import PneumoniaMNIST
import medmnist

# —— Config ————————————————————————————————
SEEDS       = list(range(1, 51))   # 50 independent runs
ROUNDS      = 100
CLIENTS     = 5
LOCAL_EPOCHS = 1
BASE_LR     = 1e-3
LAM         = 1e-2
BATCH_SIZE   = 64
WARMUP_FRAC  = 0.05

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
medmnist.INFO['pneumoniamnist']['task'] = 'binary-class'

# —— Load PneumoniaMNIST ————————————————————————
train_data = PneumoniaMNIST(split="train", download=True, transform=ToTensor())
test_data  = PneumoniaMNIST(split="test",  download=True, transform=ToTensor())

X_test = torch.stack([x[0] for x in test_data]).to(device)
y_test = torch.tensor([x[1] for x in test_data], device=device)

train_labels = np.array([x[1] for x in train_data])

# —— CNN Model ————————————————————————————
class CNN(nn.Module):
   def __init__(self):
      super().__init__()
      self.conv = nn.Sequential(
         nn.Conv2d(1, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
         nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
      )
      self.fc = nn.Sequential(
         nn.Flatten(),
         nn.Linear(64*7*7, 128), nn.ReLU(),
         nn.Linear(128, 2),
      )
   def forward(self, x):
      return self.fc(self.conv(x))
```

```
# —— Entropy Gradient ————————————————————————————
def entropy_grad(model, w0):
    with torch.no_grad():
        deltas = torch.cat([(p.data - w0i).abs().flatten()
                            for p, w0i in zip(model.parameters(), w0)])
        Z = deltas.sum().clamp_min(1e-12)
        p_j = deltas / Z
        hbar = (p_j * p_j.log()).sum()
        grads, idx = [], 0
        for p, w0i in zip(model.parameters(), w0):
            n  = p.numel()
            pj = p_j[idx:idx+n].view_as(p)
            gH = ((hbar - pj.log())/Z) * (p.data - w0i).sign()
            grads.append(gH.clone())
            idx += n
    return grads

# —— Local Training ————————————————————————————
def local_train(w0, indices, lam, lr, local_epochs, scheduler=None):
    m = CNN().to(device)
    for p, w0i in zip(m.parameters(), w0):
        p.data.copy_(w0i)
    opt = torch.optim.Adam(m.parameters(), lr=lr)
    sched = scheduler(opt) if scheduler else None
    loader = DataLoader(Subset(train_data, indices),
```

```
                batch_size=BATCH_SIZE, shuffle=True)
    for _ in range(local_epochs):
        gH = entropy_grad(m, w0)
        for xb, yb in loader:
            xb, yb = xb.to(device), yb.squeeze().long().to(device)
            opt.zero_grad()
            out = m(xb)
            loss = F.cross_entropy(out, yb)
            for p, gh in zip(m.parameters(), gH):
                if p.grad is not None:
                    p.grad.data += lam * gh.to(device)
            loss.backward()
            torch.nn.utils.clip_grad_norm_(m.parameters(), 1.0)
            opt.step()
            if sched:
                sched.step()
    return [p.data - w0i for p, w0i in zip(m.parameters(), w0)]


# ─── Client split by class ──────────────────────────────────────────
def get_client_indices(labels):
    idxs = {i: [] for i in range(CLIENTS)}
    for cls in [0, 1]:
        cls_idx = np.where(labels == cls)[0]
        np.random.shuffle(cls_idx)
        splits = np.array_split(cls_idx, CLIENTS)
        for i in range(CLIENTS):
```

```
        idxs[i].extend(splits[i].tolist())
    return idxs

# —— One ERFO run, return metric arrays ——————————————————————
def run_erfo(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)

    # warm-up IID Adam on a small fraction
    warm_size = int(WARMUP_FRAC * len(train_data))
    warm_idx  = np.random.choice(len(train_data), warm_size, replace=False)
    warm_model = CNN().to(device)
    opt0 = torch.optim.Adam(warm_model.parameters(), lr=BASE_LR)
    for xb, yb in DataLoader(Subset(train_data, warm_idx), batch_size=128, shuffle=True):
        xb, yb = xb.to(device), yb.squeeze().long().to(device)
        opt0.zero_grad()
        F.cross_entropy(warm_model(xb), yb).backward()
        opt0.step()
    global_w = [p.data.clone() for p in warm_model.parameters()]

    # build non-IID splits
    client_idxs = get_client_indices(train_labels)

    accs = np.zeros(ROUNDS)
    f1s  = np.zeros(ROUNDS)
```

```
aucs = np.zeros(ROUNDS)

# federated rounds
def lr_sched(opt):
   return torch.optim.lr_scheduler.LambdaLR(
      opt, lambda step: (step+1)/10 if step < 10 else max(0.1, 1 - (step-10)/(ROUNDS-10))
   )

for r in range(ROUNDS):
   lam_t = LAM * (1 - r/ROUNDS)
   deltas, sizes = [], []
   for cid in range(CLIENTS):
      d = local_train(global_w,
                  client_idxs[cid],
                  lam=lam_t,
                  lr=BASE_LR,
                  local_epochs=LOCAL_EPOCHS,
                  scheduler=lr_sched)
      deltas.append(d)
      sizes.append(len(client_idxs[cid]))

   total = sum(sizes)
   with torch.no_grad():
      for i, p in enumerate(global_w):
         update = sum(sizes[c]*deltas[c][i] for c in range(CLIENTS)) / total
         p.data += update
```

```
        # evaluate
        eval_m = CNN().to(device).eval()
        for p, w in zip(eval_m.parameters(), global_w):
            p.data.copy_(w)
        with torch.no_grad():
            logits = eval_m(X_test)
            probs  = F.softmax(logits, dim=1)[:,1].cpu().numpy()
            preds  = (probs >= 0.5).astype(int)

        accs[r] = accuracy_score(y_test.cpu().numpy(), preds) * 100
        f1s[r]  = f1_score(y_test.cpu().numpy(), preds, average="macro")
        aucs[r] = roc_auc_score(y_test.cpu().numpy(), probs)

    return accs, f1s, aucs

# ── Run all seeds & collect ──────────────────────────────────────
n_runs = len(SEEDS)
all_acc = np.zeros((n_runs, ROUNDS))
all_f1  = np.zeros((n_runs, ROUNDS))
all_auc = np.zeros((n_runs, ROUNDS))

for i, sd in enumerate(SEEDS):
    a, f, u = run_erfo(sd)
    all_acc[i] = a
    all_f1[i]  = f
```

```
    all_auc[i] = u

# —— Print mean ± std (95% CI) per round ————————————————————————
for r in range(ROUNDS):
    mu_a, sd_a = all_acc[:,r].mean(), all_acc[:,r].std(ddof=1)
    ci_a       = 1.96 * sd_a / math.sqrt(n_runs)
    mu_f, sd_f = all_f1[:,r].mean(),  all_f1[:,r].std(ddof=1)
    ci_f       = 1.96 * sd_f / math.sqrt(n_runs)
    mu_u, sd_u = all_auc[:,r].mean(), all_auc[:,r].std(ddof=1)
    ci_u       = 1.96 * sd_u / math.sqrt(n_runs)

    print(
        f"Round {r+1:3d} → "
        f"Acc={mu_a:6.2f}% ±{sd_a:5.2f}% (95% CI ±{ci_a:4.2f})   "
        f"Macro-F1={mu_f:5.3f} ±{sd_f:5.3f} (95% CI ±{ci_f:5.3f})   "
        f"ROC-AUC={mu_u:5.3f} ±{sd_u:5.3f} (95% CI ±{ci_u:5.3f})"
    )
```