Library Imports

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
from scipy.io import arff
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from scipy.stats import entropy, wasserstein_distance
from xgboost import XGBClassifier
import os
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
```

Class and Function Definitions

```
class FusionNetwork(nn.Module):
    def __init__(self, num_generators, feature_dim):
        super(FusionNetwork, self).__init__()
        self.weights = nn.Parameter(torch.ones(num_generators, feature_dim) / num_generators)

    def forward(self, outputs):
        stacked = torch.stack(outputs, dim=0)
        gamma = torch.softmax(self.weights, dim=0)
        fused = torch.einsum('gd,gbd->bd', gamma, stacked)
        return fused, gamma

# Masked Generator
class MaskedGenerator(nn.Module):
    def __init__(self, input_dim):
        super(MaskedGenerator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, input_dim)
        )

    def forward(self, x, mask):
        return self.net(x * mask)

# Discriminator
class Discriminator(nn.Module):
    def __init__(self, feature_dim):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(feature_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.net(x)

# MEG Model
class MEG(nn.Module):
    def __init__(self, input_dim, num_generators=None, alpha=0.1, beta=1.0):
        super(MEG, self).__init__()
        self.input_dim = input_dim
        self.num_generators = input_dim if num_generators is None else num_generators
        self.alpha = alpha
        self.beta = beta
        self.generators = nn.ModuleList([MaskedGenerator(input_dim) for _ in range(self.num_generators)])
        self.fusion = FusionNetwork(self.num_generators, input_dim)
        self.discriminator = Discriminator(input_dim)
        self.opt_gen = optim.Adam(list(self.generators.parameters()) + list(self.fusion.parameters()), lr=0.001)
        self.opt_disc = optim.Adam(self.discriminator.parameters(), lr=0.001)
```

```
        self.opt_disc = optim.Adam(self.discriminator.parameters(), lr=0.001)
        self.bce = nn.BCELoss()
        self.mse = nn.MSELoss()

    def forward(self, x):
        masks = [(torch.rand_like(x) < 0.8).float() for _ in range(self.num_generators)]
        outputs = [g(x, m) for g, m in zip(self.generators, masks)]
        fused, gamma = self.fusion(outputs)
        return outputs, fused, gamma

    def train_meg(self, data, epochs=50, batch_size=64):
        for epoch in range(epochs):
            perm = torch.randperm(data.size(0))
            for i in range(0, data.size(0), batch_size):
                idx = perm[i:i + batch_size]
                real_batch = data[idx]
                bs = real_batch.size(0)
                real_labels = torch.ones(bs, 1).to(real_batch.device)
                fake_labels = torch.zeros(bs, 1).to(real_batch.device)

                self.opt_disc.zero_grad()
                loss_real = self.bce(self.discriminator(real_batch), real_labels)
                _, fused, _ = self.forward(real_batch)
                loss_fake = self.bce(self.discriminator(fused.detach()), fake_labels)
                loss_D = loss_real + loss_fake
                loss_D.backward()
                self.opt_disc.step()

                self.opt_gen.zero_grad()
                outputs, fused, gamma = self.forward(real_batch)
                loss_proxy = self.mse(fused, real_batch)
                loss_group = sum(torch.mean(w * (out - real_batch).pow(2)) for out, w in zip(outputs, gamma))
                loss_adv = self.bce(self.discriminator(fused), real_labels)
                loss_G = loss_proxy + self.alpha * loss_group + self.beta * loss_adv
                loss_G.backward()
                self.opt_gen.step()

            if epoch % 10 == 0:
                print(f"Epoch {epoch}: Loss_D={loss_D.item():.4f}, Loss_proxy={loss_proxy.item():.4f}, "
                      f"Loss_group={loss_group.item():.4f}, Loss_adv={loss_adv.item():.4f}")

    def generate(self, x):
        _, fused, _ = self.forward(x)
        return fused

# Evaluation Functions
def jensen_shannon_divergence(real, synthetic):
    real_hist, _ = np.histogram(real, bins=50, density=True)
    syn_hist, _ = np.histogram(synthetic, bins=50, density=True)
    real_hist = real_hist + 1e-10
    syn_hist = syn_hist + 1e-10
    m = 0.5 * (real_hist + syn_hist)
    return 0.5 * (entropy(real_hist, m) + entropy(syn_hist, m))

def evaluate_dataset(dataset_name, X, y, le, results_file='results.csv'):
    dataset_size = len(X)
    epochs = 50 if dataset_size >= 30000 else 100
    print(f"\nEvaluating {dataset_name} ({dataset_size} samples, {epochs} epochs):")

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_tensor = torch.tensor(X_scaled, dtype=torch.float32)

    # Use StratifiedKFold to ensure class distribution is preserved
    skf = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)
    all_classes = np.unique(y)
    num_classes = len(all_classes)

    print(f"Number of classes: {num_classes}, Labels: {all_classes}")

    classifiers = {
        "Logistic Regression": LogisticRegression(max_iter=500),
        "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
        "MLP": MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, random_state=42),
    }

    # Add XGBoost wrapped in OneVsRestClassifier for more stable multiclass handling
    if num_classes > 2:
        classifiers["XGBoost"] = OneVsRestClassifier(XGBClassifier(eval_metric='logloss'))
    else:
        classifiers["XGBoost"] = XGBClassifier(objective='binary:logistic', eval_metric='logloss')

    tstr_scores = {name: [] for name in classifiers.keys()}
```

```
    jsd_scores = []
    wd_scores = []

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

    for fold, (train_idx, test_idx) in enumerate(skf.split(X_scaled, y)):
        X_train, X_test = X_tensor[train_idx].to(device), X_tensor[test_idx].to(device)
        y_train, y_test = y[train_idx], y[test_idx]

        # Ensure all classes are present in training data
        train_classes = np.unique(y_train)
        print(f"Fold {fold} - Classes in training data: {train_classes}")

        # Check for missing classes in training
        missing_classes = set(all_classes) - set(train_classes)
        if missing_classes:
            print(f"Warning: Missing classes in training fold {fold}: {missing_classes}")
            # Add at least one sample from each missing class
            for cls in missing_classes:
                cls_indices = np.where(y == cls)[0]
                if len(cls_indices) > 0:
                    # Add sample to training
                    idx = cls_indices[0]
                    train_idx = np.append(train_idx, idx)
                    # Remove from test if present
                    if idx in test_idx:
                        test_idx = test_idx[test_idx != idx]

            # Update splits
            X_train = X_tensor[train_idx].to(device)
            X_test = X_tensor[test_idx].to(device)
            y_train = y[train_idx]
            y_test = y[test_idx]

            print(f"After adjustment - Classes in training: {np.unique(y_train)}")

        meg = MEG(input_dim=X_train.shape[1], num_generators=X.shape[1] - 1, alpha=0.1, beta=1.0)
        meg.to(device)
        meg.train_meg(X_train, epochs=epochs, batch_size=64)

        # Generate synthetic data with stratification
        synthetic_data = []
        synthetic_labels = []

        # Process each class to ensure representation
        for cls in all_classes:
            cls_indices = np.where(y_train == cls)[0]
            if len(cls_indices) > 0:
                # Select up to half the samples for this class, minimum 1
                cls_sample_size = max(1, int(len(cls_indices) * 0.5))
                cls_sample_indices = cls_indices[:cls_sample_size]

                # Generate data for this class
                cls_synthetic = meg.generate(X_train[cls_sample_indices]).detach().cpu().numpy()
                cls_labels = np.full(cls_sample_size, cls)

                synthetic_data.append(cls_synthetic)
                synthetic_labels.append(cls_labels)

        # Combine all synthetic data
        X_syn = np.vstack(synthetic_data)
        y_syn = np.concatenate(synthetic_labels)

        print(f"Fold {fold} - Classes in synthetic data: {np.unique(y_syn)}")

        for clf_name, clf in classifiers.items():
            try:
                clf.fit(X_syn, y_syn)
                y_pred = clf.predict(X_test.cpu().numpy())
                score = accuracy_score(y_test, y_pred)
                tstr_scores[clf_name].append(score)
                print(f"Fold {fold} - {clf_name} accuracy: {score:.4f}")
            except Exception as e:
                print(f"Error with {clf_name} on fold {fold}: {e}")
                # Try to recover with OneVsRestClassifier if needed
                if clf_name == "XGBoost" and "Invalid classes" in str(e):
                    try:
                        print("Trying with OneVsRestClassifier...")
                        recovery_clf = OneVsRestClassifier(XGBClassifier(eval_metric='logloss'))
                        recovery_clf.fit(X_syn, y_syn)
                        y_pred = recovery_clf.predict(X_test.cpu().numpy())
                        score = accuracy_score(y_test, y_pred)
                        tstr_scores[clf_name].append(score)
```

```
                    print(f"Recovery successful! Accuracy: {score:.4f}")
                except Exception as e2:
                    print(f"Recovery failed: {e2}")
                    tstr_scores[clf_name].append(0)
            else:
                tstr_scores[clf_name].append(0)

        real_flat = X_test.cpu().numpy().flatten()
        syn_flat = X_syn.flatten()
        jsd_scores.append(jensen_shannon_divergence(real_flat, syn_flat))
        wd_scores.append(wasserstein_distance(real_flat, syn_flat))

    result = {'Dataset': dataset_name}
    for clf_name in classifiers.keys():
        result[f'TSTR ({clf_name}) Mean'] = np.mean(tstr_scores[clf_name])
        result[f'TSTR ({clf_name}) Std'] = np.std(tstr_scores[clf_name])
    result['JSD Mean'] = np.mean(jsd_scores)
    result['JSD Std'] = np.std(jsd_scores)
    result['WD Mean'] = np.mean(wd_scores)
    result['WD Std'] = np.std(wd_scores)

    if os.path.exists(results_file):
        df = pd.read_csv(results_file)
        df = pd.concat([df, pd.DataFrame([result])], ignore_index=True)
    else:
        df = pd.DataFrame([result])

    df.to_csv(results_file, index=False)
    print(f"\nResults for {dataset_name}:")
    print(df)

    return result
```

Data Loading Function

```
def load_arff(file_path):
    data, meta = arff.loadarff(file_path)
    df = pd.DataFrame(data)
    for col in df.columns:
        if df[col].dtype == object:
            if isinstance(df[col].iloc[0], bytes):
                df[col] = df[col].str.decode('utf-8')
            else:
                df[col] = df[col]
    class_col = meta.names()[-1]
    df_features = pd.get_dummies(df.drop(columns=[class_col]), drop_first=True)
    le = LabelEncoder()
    df[class_col] = le.fit_transform(df[class_col])
    df = pd.concat([df_features, df[class_col]], axis=1)
    X = df.drop(columns=[class_col]).values
    y = df[class_col].values
    print(f"Dataset: {file_path}, Unique labels: {np.unique(y)}")
    return X, y, le
```

Evaluate `car`

```
X, y, le = load_arff("car 1.arff")
evaluate_dataset("car", X, y, le, results_file='results.csv')
```

```
Dataset: car 1.arff, Unique labels: [0 1 2 3]

Evaluating car (1728 samples, 100 epochs):
Number of classes: 4, Labels: [0 1 2 3]
Fold 0 - Classes in training data: [0 1 2 3]
Epoch 0: Loss_D=1.2544, Loss_proxy=0.7852, Loss_group=0.8310, Loss_adv=0.5729
Epoch 10: Loss_D=1.4314, Loss_proxy=0.1066, Loss_group=0.4012, Loss_adv=0.6586
Epoch 20: Loss_D=1.3972, Loss_proxy=0.0291, Loss_group=0.3339, Loss_adv=0.6924
Epoch 30: Loss_D=1.3933, Loss_proxy=0.0233, Loss_group=0.3053, Loss_adv=0.7002
Epoch 40: Loss_D=1.3901, Loss_proxy=0.0215, Loss_group=0.2824, Loss_adv=0.6960
Epoch 50: Loss_D=1.3883, Loss_proxy=0.0196, Loss_group=0.2659, Loss_adv=0.6987
Epoch 60: Loss_D=1.3871, Loss_proxy=0.0211, Loss_group=0.2679, Loss_adv=0.6946
Epoch 70: Loss_D=1.3859, Loss_proxy=0.0189, Loss_group=0.2581, Loss_adv=0.6942
Epoch 80: Loss_D=1.3857, Loss_proxy=0.0179, Loss_group=0.2601, Loss_adv=0.6960
Epoch 90: Loss_D=1.3848, Loss_proxy=0.0201, Loss_group=0.2559, Loss_adv=0.6942
Fold 0 - Classes in synthetic data: [0 1 2 3]
Fold 0 - Logistic Regression accuracy: 0.7465
Fold 0 - Random Forest accuracy: 0.7153
```

```
Fold 0 - MLP accuracy: 0.7778
Fold 0 - XGBoost accuracy: 0.8148
Fold 1 - Classes in training data: [0 1 2 3]
Epoch 0: Loss_D=1.3067, Loss_proxy=0.8416, Loss_group=0.8922, Loss_adv=0.6056
Epoch 10: Loss_D=1.4293, Loss_proxy=0.0896, Loss_group=0.3914, Loss_adv=0.6990
Epoch 20: Loss_D=1.4195, Loss_proxy=0.0477, Loss_group=0.3475, Loss_adv=0.6985
Epoch 30: Loss_D=1.3931, Loss_proxy=0.0239, Loss_group=0.3167, Loss_adv=0.7057
Epoch 40: Loss_D=1.3870, Loss_proxy=0.0177, Loss_group=0.2695, Loss_adv=0.6966
Epoch 50: Loss_D=1.3876, Loss_proxy=0.0167, Loss_group=0.2792, Loss_adv=0.6988
Epoch 60: Loss_D=1.3865, Loss_proxy=0.0184, Loss_group=0.2558, Loss_adv=0.6988
Epoch 70: Loss_D=1.3857, Loss_proxy=0.0179, Loss_group=0.2613, Loss_adv=0.6907
Epoch 80: Loss_D=1.3844, Loss_proxy=0.0193, Loss_group=0.2617, Loss_adv=0.6994
Epoch 90: Loss_D=1.3838, Loss_proxy=0.0195, Loss_group=0.2647, Loss_adv=0.6964
Fold 1 - Classes in synthetic data: [0 1 2 3]
Fold 1 - Logistic Regression accuracy: 0.6852
Fold 1 - Random Forest accuracy: 0.7257
Fold 1 - MLP accuracy: 0.7373
Fold 1 - XGBoost accuracy: 0.7593

Results for car:
    Dataset  TSTR (Logistic Regression) Mean  TSTR (Logistic Regression) Std  \
0  credit-a                         0.821256                        0.016368
1       car                         0.715278                        0.016204
2       car                         0.715856                        0.030671

   TSTR (Random Forest) Mean  TSTR (Random Forest) Std  TSTR (MLP) Mean  \
0                   0.847343                  0.017445         0.809662
1                   0.713542                  0.000579         0.758102
2                   0.720486                  0.005208         0.757523

   TSTR (MLP) Std  TSTR (XGBoost) Mean  TSTR (XGBoost) Std  JSD Mean  \
0        0.017920             0.857971            0.014874  0.106695
1        0.031250             0.793981            0.000000  0.583143
2        0.020255             0.787037            0.027778  0.588338

    JSD Std   WD Mean    WD Std
0  0.070606  0.052980  0.006941
1  0.018523  0.095162  0.000923
2  0.002535  0.095527  0.001674
{'Dataset': 'car'
```

Evaluate `credit-a`

```
X, y, le = load_arff("credit-a.arff")
evaluate_dataset("credit-a", X, y, le, results_file='results.csv')
```

```
Fold 0 - Logistic Regression accuracy: 0.8609
Fold 0 - Random Forest accuracy: 0.8667
Fold 0 - MLP accuracy: 0.8232
Fold 0 - XGBoost accuracy: 0.8551
Fold 1 - Classes in training data: [0 1]
Epoch 0: Loss_D=1.3034, Loss_proxy=0.8207, Loss_group=0.8604, Loss_adv=0.6284
Epoch 10: Loss_D=1.4726, Loss_proxy=0.7132, Loss_group=0.8127, Loss_adv=0.4396
Epoch 20: Loss_D=1.5271, Loss_proxy=0.5098, Loss_group=0.6769, Loss_adv=0.6407
Epoch 30: Loss_D=1.4552, Loss_proxy=0.5613, Loss_group=1.0616, Loss_adv=0.6387
Epoch 40: Loss_D=1.4077, Loss_proxy=0.0907, Loss_group=0.3121, Loss_adv=0.6726
Epoch 50: Loss_D=1.3994, Loss_proxy=0.0648, Loss_group=0.4946, Loss_adv=0.6765
Epoch 60: Loss_D=1.3948, Loss_proxy=0.0564, Loss_group=0.3445, Loss_adv=0.7042
Epoch 70: Loss_D=1.3950, Loss_proxy=0.0390, Loss_group=0.4805, Loss_adv=0.6979
Epoch 80: Loss_D=1.3921, Loss_proxy=0.0176, Loss_group=0.3126, Loss_adv=0.7012
Epoch 90: Loss_D=1.3914, Loss_proxy=0.0133, Loss_group=0.2877, Loss_adv=0.7070
Fold 1 - Classes in synthetic data: [0 1]
Fold 1 - Logistic Regression accuracy: 0.8319
Fold 1 - Random Forest accuracy: 0.8522
Fold 1 - MLP accuracy: 0.8435
Fold 1 - XGBoost accuracy: 0.8609

Results for credit-a:
    Dataset  TSTR (Logistic Regression) Mean  TSTR (Logistic Regression) Std  \
0  credit-a                         0.821256                        0.016368
1       car                         0.715278                        0.016204
2       car                         0.715856                        0.030671
```

```
1  0.018523  0.095162  0.000923
2  0.002535  0.095527  0.001674
3  0.084962  0.041119  0.003898
{'Dataset': 'credit-a',
 'TSTR (Logistic Regression) Mean': np.float64(0.846376811594203),
 'TSTR (Logistic Regression) Std': np.float64(0.014492753623188415),
 'TSTR (Random Forest) Mean': np.float64(0.8594202898550725),
 'TSTR (Random Forest) Std': np.float64(0.007246376811594235),
 'TSTR (MLP) Mean': np.float64(0.8333333333333333),
 'TSTR (MLP) Std': np.float64(0.010144927536231918),
 'TSTR (XGBoost) Mean': np.float64(0.8579710144927537),
 'TSTR (XGBoost) Std': np.float64(0.002898550724637683),
 'JSD Mean': np.float64(0.15947554664262437),
 'JSD Std': np.float64(0.08496183163354602),
 'WD Mean': np.float64(0.04111857703108171),
 'WD Std': np.float64(0.0038980460302406802)}
```

Evaluate `nursery`

```
X, y, le = load_arff("nursery 1.arff")
evaluate_dataset("nursery", X, y, le, results_file='results.csv')
```

```
Dataset: nursery 1.arff, Unique labels: [0 1 2 3 4]

Evaluating nursery (12960 samples, 100 epochs):
Number of classes: 5, Labels: [0 1 2 3 4]
Fold 0 - Classes in training data: [0 1 2 3 4]
Epoch 0: Loss_D=1.4300, Loss_proxy=0.1868, Loss_group=0.4628, Loss_adv=0.5548
Epoch 10: Loss_D=1.3860, Loss_proxy=0.0133, Loss_group=0.2578, Loss_adv=0.6894
Epoch 20: Loss_D=1.3856, Loss_proxy=0.0125, Loss_group=0.2440, Loss_adv=0.6927
Epoch 30: Loss_D=1.3876, Loss_proxy=0.0194, Loss_group=0.2613, Loss_adv=0.6842
Epoch 40: Loss_D=1.3850, Loss_proxy=0.0158, Loss_group=0.2593, Loss_adv=0.6947
Epoch 50: Loss_D=1.3900, Loss_proxy=0.0178, Loss_group=0.2505, Loss_adv=0.6805
Epoch 60: Loss_D=1.3875, Loss_proxy=0.0155, Loss_group=0.2331, Loss_adv=0.7029
Epoch 70: Loss_D=1.3805, Loss_proxy=0.0226, Loss_group=0.2443, Loss_adv=0.7072
Epoch 80: Loss_D=1.3923, Loss_proxy=0.0283, Loss_group=0.2278, Loss_adv=0.7034
Epoch 90: Loss_D=1.3723, Loss_proxy=0.0436, Loss_group=0.2507, Loss_adv=0.7390
Fold 0 - Classes in synthetic data: [0 1 2 3 4]
Fold 0 - Logistic Regression accuracy: 0.8136
Fold 0 - Random Forest accuracy: 0.8020
Fold 0 - MLP accuracy: 0.8309
Fold 0 - XGBoost accuracy: 0.8341
Fold 1 - Classes in training data: [0 1 2 3 4]
Epoch 0: Loss_D=1.4385, Loss_proxy=0.1828, Loss_group=0.4257, Loss_adv=0.5199
Epoch 10: Loss_D=1.3874, Loss_proxy=0.0154, Loss_group=0.2657, Loss_adv=0.6959
Epoch 20: Loss_D=1.3862, Loss_proxy=0.0145, Loss_group=0.2556, Loss_adv=0.6994
Epoch 30: Loss_D=1.3668, Loss_proxy=0.0324, Loss_group=0.2724, Loss_adv=0.6987
Epoch 40: Loss_D=1.3295, Loss_proxy=0.0345, Loss_group=0.2605, Loss_adv=0.7175
Epoch 50: Loss_D=1.3228, Loss_proxy=0.0339, Loss_group=0.2644, Loss_adv=0.6943
Epoch 60: Loss_D=1.2893, Loss_proxy=0.0496, Loss_group=0.2659, Loss_adv=0.7823
Epoch 70: Loss_D=1.4029, Loss_proxy=0.0604, Loss_group=0.2613, Loss_adv=0.8120
Epoch 80: Loss_D=1.3693, Loss_proxy=0.0388, Loss_group=0.2485, Loss_adv=0.7371
Epoch 90: Loss_D=1.2318, Loss_proxy=0.0259, Loss_group=0.2667, Loss_adv=0.9511
Fold 1 - Classes in synthetic data: [0 1 2 3 4]
Fold 1 - Logistic Regression accuracy: 0.8591
Fold 1 - Random Forest accuracy: 0.7923
Fold 1 - MLP accuracy: 0.8586
Fold 1 - XGBoost accuracy: 0.8431

Results for nursery:
    Dataset  TSTR (Logistic Regression) Mean  TSTR (Logistic Regression) Std  \
0  credit-a                         0.821256                        0.016368   
1       car                         0.715278                        0.016204   
2       car                         0.715856                        0.030671   
3  credit-a                         0.846377                        0.014493   
4   nursery                         0.836343                        0.022762   

   TSTR (Random Forest) Mean  TSTR (Random Forest) Std  TSTR (MLP) Mean  \
0                   0.847343                  0.017445         0.809662   
1                   0.713542                  0.000579         0.758102   
2                   0.720486                  0.005208         0.757523   
3                   0.859420                  0.007246         0.833333   
4                   0.797145                  0.004861         0.844753   

   TSTR (MLP) Std  TSTR (XGBoost) Mean  TSTR (XGBoost) Std  JSD Mean  \
0        0.017920             0.857971            0.014874  0.106695   
1        0.031250             0.793981            0.000000  0.583143   
2        0.020255             0.787037            0.027778  0.588338   
3        0.010145             0.857971            0.002899  0.159476   
4        0.013889             0.838580            0.004475  0.445901   
```

Evaluate `dermatology`

```
X, y, le = load_arff("dermatology.arff")
evaluate_dataset("dermatology", X, y, le, results_file='results.csv')
```

```
Fold 1 - Logistic Regression accuracy: 0.8852
Fold 1 - Random Forest accuracy: 0.7322
Fold 1 - MLP accuracy: 0.8525
Fold 1 - XGBoost accuracy: 0.3497

Results for dermatology:
       Dataset  TSTR (Logistic Regression) Mean  \
0     credit-a                         0.821256
1          car                         0.715278
2          car                         0.715856
3     credit-a                         0.846377
4      nursery                         0.836343
5  dermatology                         0.882514

   TSTR (Logistic Regression) Std  TSTR (Random Forest) Mean  \
0                        0.016368                   0.847343
1                        0.016204                   0.713542
2                        0.030671                   0.720486
3                        0.014493                   0.859420
4                        0.022762                   0.797145
5                        0.002732                   0.737705

   TSTR (Random Forest) Std  TSTR (MLP) Mean  TSTR (MLP) Std  \
0                  0.017445         0.809662        0.017920
1                  0.000579         0.758102        0.031250
2                  0.005208         0.757523        0.020255
3                  0.007246         0.833333        0.010145
4                  0.004861         0.844753        0.013889
5                  0.005464         0.830601        0.021858

   TSTR (XGBoost) Mean  TSTR (XGBoost) Std  JSD Mean   JSD Std   WD Mean  \
0             0.857971            0.014874  0.106695  0.070606  0.052980
1             0.793981            0.000000  0.583143  0.018523  0.095162
2             0.787037            0.027778  0.588338  0.002535  0.095527
3             0.857971            0.002899  0.159476  0.084962  0.041119
4             0.838580            0.004475  0.445901  0.049923  0.090403
5             0.371585            0.021858  0.453897  0.225141  0.341872

     WD Std
0  0.006941
1  0.000923
2  0.001674
3  0.003898
4  0.004790
5  0.001922
{'Dataset': 'dermatology',
 'TSTR (Logistic Regression) Mean': np.float64(0.8825136612021858),
 'TSTR (Logistic Regression) Std': np.float64(0.002732240437158473),
 'TSTR (Random Forest) Mean': np.float64(0.7377049180327869),
 'TSTR (Random Forest) Std': np.float64(0.005464480874316946),
 'TSTR (MLP) Mean': np.float64(0.8306010928961749),
 'TSTR (MLP) Std': np.float64(0.02185792349726773),
 'TSTR (XGBoost) Mean': np.float64(0.3715846994535519),
 'TSTR (XGBoost) Std': np.float64(0.021857923497267756),
 'JSD Mean': np.float64(0.45389741250670385),
 'JSD Std': np.float64(0.22514073877973023),
 'WD Mean': np.float64(0.34187178680631675),
 'WD Std': np.float64(0.0019223334785075097)}
```

```
X, y, le = load_arff("connect-4.arff")
evaluate_dataset("connec-4", X, y, le, results_file='results.csv')
```

```
Dataset: connect-4.arff, Unique labels: [0 1 2]

Evaluating connec-4 (67557 samples, 50 epochs):
Number of classes: 3, Labels: [0 1 2]
Fold 0 - Classes in training data: [0 1 2]
Epoch 0: Loss_D=0.9930, Loss_proxy=0.7022, Loss_group=0.8269, Loss_adv=1.1782
Epoch 10: Loss_D=1.3894, Loss_proxy=0.0125, Loss_group=0.3688, Loss_adv=0.6972
Epoch 20: Loss_D=1.3928, Loss_proxy=0.0277, Loss_group=0.2750, Loss_adv=0.7037
Epoch 30: Loss_D=1.3739, Loss_proxy=0.0244, Loss_group=0.2480, Loss_adv=0.7071
Epoch 40: Loss_D=1.3388, Loss_proxy=0.0232, Loss_group=0.2904, Loss_adv=0.7098
Fold 0 - Classes in synthetic data: [0 1 2]
Fold 0 - Logistic Regression accuracy: 0.7528
Fold 0 - Random Forest accuracy: 0.7204
Fold 0 - MLP accuracy: 0.7920
Fold 0 - XGBoost accuracy: 0.7612
Fold 1 - Classes in training data: [0 1 2]
Epoch 0: Loss_D=1.0439, Loss_proxy=0.9815, Loss_group=1.1542, Loss_adv=1.7139
Epoch 10: Loss_D=1.3911, Loss_proxy=0.0129, Loss_group=0.4073, Loss_adv=0.6825
Epoch 20: Loss_D=1.3785, Loss_proxy=0.0172, Loss_group=0.2511, Loss_adv=0.7134
Epoch 30: Loss_D=1.3635, Loss_proxy=0.0226, Loss_group=0.2698, Loss_adv=0.7045
Epoch 40: Loss_D=1.3605, Loss_proxy=0.0244, Loss_group=0.3368, Loss_adv=0.6982
Fold 1 - Classes in synthetic data: [0 1 2]
Fold 1 - Logistic Regression accuracy: 0.7567
Fold 1 - Random Forest accuracy: 0.7137
```

```
Fold 1 - MLP accuracy: 0.7940
Fold 1 - XGBoost accuracy: 0.7751

Results for connec-4:
    Dataset  TSTR (Logistic Regression) Mean  TSTR (Logistic Regression) Std  \
0  connec-4                          0.75477                        0.001935

   TSTR (Random Forest) Mean  TSTR (Random Forest) Std  TSTR (MLP) Mean  \
0                  0.717024                  0.003364          0.79299

   TSTR (MLP) Std  TSTR (XGBoost) Mean  TSTR (XGBoost) Std  JSD Mean  \
0        0.001018             0.768137            0.006924  0.648031

    JSD Std   WD Mean    WD Std
0  0.008814  0.039352  0.004224
{'Dataset': 'connec-4',
 'TSTR (Logistic Regression) Mean': np.float64(0.7547700746846856),
 'TSTR (Logistic Regression) Std': np.float64(0.0019354733051302198),
 'TSTR (Random Forest) Mean': np.float64(0.7170240927762597),
 'TSTR (Random Forest) Std': np.float64(0.0033643142221712607),
 'TSTR (MLP) Mean': np.float64(0.7929896386502584),
 'TSTR (MLP) Std': np.float64(0.0010182955080694778),
 'TSTR (XGBoost) Mean': np.float64(0.7681366390463067),
 'TSTR (XGBoost) Std': np.float64(0.006924051343888071),
 'JSD Mean': np.float64(0.6480312959552712),
 'JSD Std': np.float64(0.008813960876347116),
 'WD Mean': np.float64(0.03935235932818211),
 'WD Std': np.float64(0.004223978276841997)}
```

```
result_df = pd.read_csv('results.csv')
result_df
```

	Dataset	TSTR (Logistic Regression) Mean	TSTR (Logistic Regression) Std	TSTR (Random Forest) Mean	TSTR (Random Forest) Std	TSTR (MLP) Mean	TSTR (MLP) Std	TSTR (XGBoost) Mean	TSTR (XGBoost) Std	JSD Mean	JSD Std	WD Mean
0	car	0.715856	0.030671	0.720486	0.005208	0.757523	0.020255	0.787037	0.027778	0.588338	0.002535	0.095527
1	credit-a	0.846377	0.014493	0.859420	0.007246	0.833333	0.010145	0.857971	0.002899	0.159476	0.084962	0.041119
2	nursery	0.836343	0.022762	0.797145	0.004861	0.844753	0.013889	0.838580	0.004475	0.445901	0.049923	0.090403

```
X, y, le = load_arff("letter-recog.arff")
evaluate_dataset("letter-recog", X, y, le, results_file='results.csv')
```

```
Dataset: letter-recog.arff, Unique labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]

Evaluating letter-recog (20000 samples, 100 epochs):
Number of classes: 26, Labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]
Fold 0 - Classes in training data: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]
Epoch 0: Loss_D=1.2469, Loss_proxy=0.9986, Loss_group=1.0308, Loss_adv=1.6929
Epoch 10: Loss_D=0.5682, Loss_proxy=1.1818, Loss_group=1.5140, Loss_adv=1.6047
Epoch 20: Loss_D=0.7818, Loss_proxy=1.0434, Loss_group=1.7180, Loss_adv=2.0063
Epoch 30: Loss_D=1.7524, Loss_proxy=0.7084, Loss_group=1.7159, Loss_adv=0.9100
Epoch 40: Loss_D=1.5354, Loss_proxy=0.3552, Loss_group=1.5524, Loss_adv=0.7565
Epoch 50: Loss_D=1.4587, Loss_proxy=0.1354, Loss_group=1.3607, Loss_adv=0.7962
Epoch 60: Loss_D=1.3966, Loss_proxy=0.0648, Loss_group=0.9453, Loss_adv=0.6279
Epoch 70: Loss_D=1.4234, Loss_proxy=0.0347, Loss_group=0.7350, Loss_adv=0.6869
Epoch 80: Loss_D=1.3930, Loss_proxy=0.0207, Loss_group=0.5889, Loss_adv=0.6915
Epoch 90: Loss_D=1.3853, Loss_proxy=0.0118, Loss_group=0.3461, Loss_adv=0.6948
Fold 0 - Classes in synthetic data: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]
Fold 0 - Logistic Regression accuracy: 0.7736
Fold 0 - Random Forest accuracy: 0.2408
Fold 0 - MLP accuracy: 0.8201
Fold 0 - XGBoost accuracy: 0.4843
Fold 1 - Classes in training data: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]
Epoch 0: Loss_D=1.1012, Loss_proxy=0.9299, Loss_group=0.9550, Loss_adv=1.5794
Epoch 10: Loss_D=0.8355, Loss_proxy=1.0611, Loss_group=1.4760, Loss_adv=1.3588
Epoch 20: Loss_D=1.1899, Loss_proxy=1.0232, Loss_group=1.6483, Loss_adv=1.2147
Epoch 30: Loss_D=1.3857, Loss_proxy=0.6468, Loss_group=1.3214, Loss_adv=0.8996
Epoch 40: Loss_D=1.5608, Loss_proxy=0.3884, Loss_group=1.4302, Loss_adv=0.8578
Epoch 50: Loss_D=1.4006, Loss_proxy=0.2139, Loss_group=1.4587, Loss_adv=0.7951
Epoch 60: Loss_D=1.3799, Loss_proxy=0.0871, Loss_group=1.1454, Loss_adv=0.7384
Epoch 70: Loss_D=1.3971, Loss_proxy=0.0331, Loss_group=0.7888, Loss_adv=0.6909
Epoch 80: Loss_D=1.3863, Loss_proxy=0.0170, Loss_group=0.5049, Loss_adv=0.6934
Epoch 90: Loss_D=1.3893, Loss_proxy=0.0238, Loss_group=0.4859, Loss_adv=0.7045
Fold 1 - Classes in synthetic data: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]
Fold 1 - Logistic Regression accuracy: 0.7955
Fold 1 - Random Forest accuracy: 0.5636
Fold 1 - MLP accuracy: 0.8256
Fold 1 - XGBoost accuracy: 0.5448
```

```
Results for letter-recog:
        Dataset  TSTR (Logistic Regression) Mean  \
0      connec-4                          0.75477
1  letter-recog                          0.78455

   TSTR (Logistic Regression) Std  TSTR (Random Forest) Mean  \
0                        0.001935                   0.717024
1                        0.010950                   0.402200

   TSTR (Random Forest) Std  TSTR (MLP) Mean  TSTR (MLP) Std  \
0                  0.003364          0.79299        0.001018
1                  0.161400          0.82285        0.002750

   TSTR (XGBoost) Mean  TSTR (XGBoost) Std  JSD Mean   JSD Std   WD Mean  \
0             0.768137            0.006924  0.648031  0.008814  0.039352
```

```
X, y, le = load_arff("adult 1.arff")
evaluate_dataset("adult 1", X, y, le, results_file='results.csv')
```

```
Dataset: adult 1.arff, Unique labels: [0 1]

Evaluating adult 1 (48842 samples, 50 epochs):
Number of classes: 2, Labels: [0 1]
Fold 0 - Classes in training data: [0 1]
Epoch 0: Loss_D=1.2419, Loss_proxy=0.9318, Loss_group=1.0381, Loss_adv=0.7569
Epoch 10: Loss_D=1.2680, Loss_proxy=0.7673, Loss_group=1.7374, Loss_adv=0.8964
Epoch 20: Loss_D=1.4151, Loss_proxy=0.0500, Loss_group=0.7102, Loss_adv=0.7106
```

```
X, y, le = load_arff("chess.arff")
evaluate_dataset("chess", X, y, le, results_file='results.csv')
```