CODE

```python
import os
import torch
import torch.nn.functional as F
from torch_geometric.data import Dataset, Data
from torch_geometric.loader import DataLoader
from torch_geometric.nn import GCNConv, global_mean_pool
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,
roc_auc_score, f1_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Paths to the graph data directories
human_audio_path = '/kaggle/input/english-gnn/graph_data/human_audio_graph'
amazon_audio_path = '/kaggle/input/english-gnn/graph_data/amazon_audio_graph'
azure_audio_path = '/kaggle/input/english-gnn/graph_data/azure_audio_graph'
gTTS_audio_path = '/kaggle/input/english-gnn/graph_data/gTTS_audio_graph'

class AudioGraphDataset(Dataset):
  def __init__(self, root, label, transform=None, pre_transform=None):
     super(AudioGraphDataset, self).__init__(root, transform, pre_transform)
     self.graph_files = [os.path.join(root, f) for f in os.listdir(root) if f.endswith('.pt')]
     self.label = label

  def len(self):
     return len(self.graph_files)

  def get(self, idx):
     graph = torch.load(self.graph_files[idx])
     graph.y = torch.tensor([self.label], dtype=torch.long)
     return graph

# Load datasets with labels
human_dataset = AudioGraphDataset(human_audio_path, label=0)
amazon_dataset = AudioGraphDataset(amazon_audio_path, label=1)
azure_dataset = AudioGraphDataset(azure_audio_path, label=2)
gTTS_dataset = AudioGraphDataset(gTTS_audio_path, label=3)

# Combine datasets
dataset = [human_dataset[i] for i in range(len(human_dataset))] + \
      [amazon_dataset[i] for i in range(len(amazon_dataset))] + \
      [azure_dataset[i] for i in range(len(azure_dataset))] + \
```

```
    [gTTS_dataset[i] for i in range(len(gTTS_dataset))]

# Split the dataset into train, validation, and test sets (70/15/15 split)
train_dataset, temp_dataset = train_test_split(dataset, test_size=0.3, stratify=[graph.y.item() for
graph in dataset])
val_dataset, test_dataset = train_test_split(temp_dataset, test_size=0.5, stratify=[graph.y.item() for
graph in temp_dataset])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Define the GNN model
class GNN(torch.nn.Module):
  def __init__(self, num_node_features, num_classes):
    super(GNN, self).__init__()
    self.conv1 = GCNConv(num_node_features, 64)
    self.conv2 = GCNConv(64, 64)
    self.conv3 = GCNConv(64, 64)
    self.fc1 = torch.nn.Linear(64, 32)
    self.fc2 = torch.nn.Linear(32, num_classes)

  def forward(self, x, edge_index, batch):
    x = F.relu(self.conv1(x, edge_index))
    x = F.relu(self.conv2(x, edge_index))
    x = F.relu(self.conv3(x, edge_index))
    x = global_mean_pool(x, batch)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return F.log_softmax(x, dim=1)

# Initialize the model, optimizer, and loss function
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GNN(num_node_features=dataset[0].num_node_features, num_classes=4).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()

# Training function
def train():
  model.train()
  for data in train_loader:
    data = data.to(device)
    optimizer.zero_grad()
    out = model(data.x, data.edge_index, data.batch)
    loss = criterion(out, data.y)
    loss.backward()
```

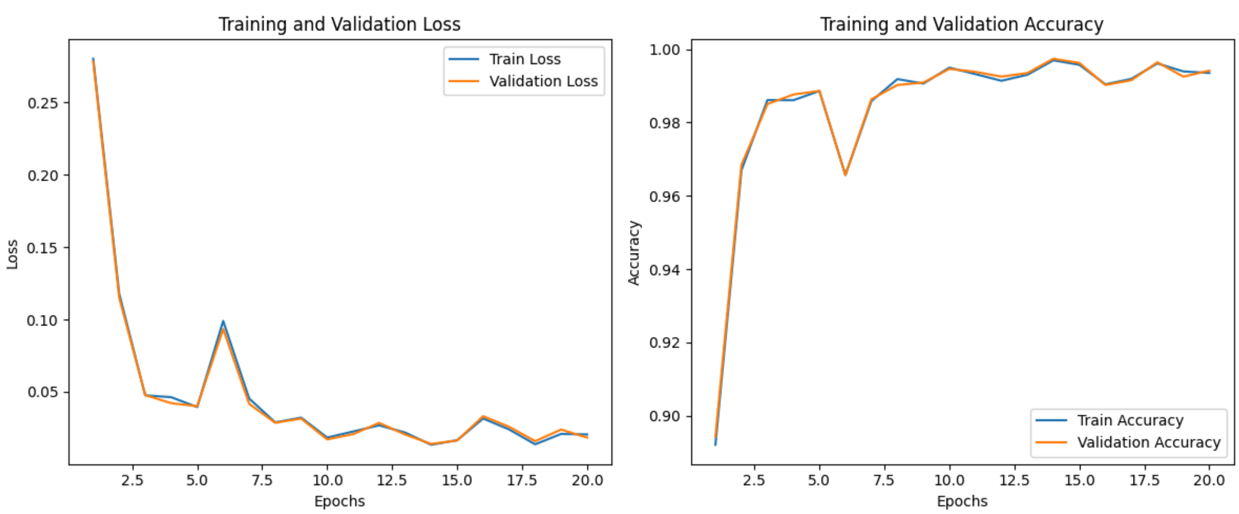
```
        optimizer.step()

# Validation function
def validate(loader):
    model.eval()
    correct = 0
    loss = 0
    with torch.no_grad():
        for data in loader:
            data = data.to(device)
            out = model(data.x, data.edge_index, data.batch)
            loss += criterion(out, data.y).item()
            pred = out.argmax(dim=1)
            correct += pred.eq(data.y).sum().item()
    return correct / len(loader.dataset), loss / len(loader)

# Test function
def test(loader):
    model.eval()
    all_preds = []
    all_labels = []
    all_probs = []
    with torch.no_grad():
        for data in loader:
            data = data.to(device)
            out = model(data.x, data.edge_index, data.batch)
            pred = out.argmax(dim=1)
            prob = out.softmax(dim=1)
            all_preds.extend(pred.cpu().numpy())
            all_labels.extend(data.y.cpu().numpy())
            all_probs.extend(prob.cpu().numpy())
    return all_labels, all_preds, all_probs

# Training loop with metric tracking
num_epochs = 20
best_val_acc = 0
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(1, num_epochs + 1):
    train()
    train_acc, train_loss = validate(train_loader)
    val_acc, val_loss = validate(val_loader)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
```

```
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)

    print(f'Epoch {epoch:03d}, Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}, Val Loss:
{val_loss:.4f}, Val Acc: {val_acc:.4f}')

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(model.state_dict(), '/kaggle/working/best_model.pth')

# Plotting the metrics
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.tight_layout()
plt.savefig('/kaggle/working/training_validation_metrics.png')
plt.show()

# Load the best model and test
model.load_state_dict(torch.load('/kaggle/working/best_model.pth'))
test_labels, test_preds, test_probs = test(test_loader)

# Evaluation
test_accuracy = accuracy_score(test_labels, test_preds)
classification_rep = classification_report(test_labels, test_preds, target_names=['human', 'amazon',
'azure', 'gTTS'], output_dict=True)
conf_matrix = confusion_matrix(test_labels, test_preds)

# Additional metrics
test_labels_bin = label_binarize(test_labels, classes=[0, 1, 2, 3])
roc_auc = roc_auc_score(test_labels_bin, test_probs, average='macro', multi_class='ovo')
```

```
f1 = f1_score(test_labels, test_preds, average='macro')
recall = recall_score(test_labels, test_preds, average='macro')

print(f"Test Accuracy: {test_accuracy:.4f}")
print("Classification Report:")
print(pd.DataFrame(classification_rep).transpose())
print("Confusion Matrix:")
print(conf_matrix)
print(f"ROC-AUC Score: {roc_auc:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Recall Score: {recall:.4f}")

# Save results
results = {
  'Test Accuracy': [test_accuracy],
  'ROC-AUC Score': [roc_auc],
  'F1 Score': [f1],
  'Recall Score': [recall]
}

results_df = pd.DataFrame(results)
results_df.to_csv('/kaggle/working/test_results.csv', index=False)

# Save the classification report
classification_report_df = pd.DataFrame(classification_rep).transpose()
classification_report_df.to_csv('/kaggle/working/classification_report.csv', index=True)

# Save the confusion matrix
conf_matrix_df = pd.DataFrame(conf_matrix, index=['human', 'amazon', 'azure', 'gTTS'],
columns=['human', 'amazon', 'azure', 'gTTS'])
conf_matrix_df.to_csv('/kaggle/working/confusion_matrix.csv', index=True)

# Save the test predictions and probabilities
test_results_df = pd.DataFrame({
  'True Labels': test_labels,
  'Predicted Labels': test_preds,
  'Probabilities': [list(prob) for prob in test_probs]
})
test_results_df.to_csv('/kaggle/working/test_predictions.csv', index=False)

# Plot the heat map for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="YlGnBu", xticklabels=['human', 'amazon',
'azure', 'gTTS'], yticklabels=['human', 'amazon', 'azure', 'gTTS'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
```

```
plt.title('Confusion Matrix Heatmap')
plt.savefig('/kaggle/working/confusion_matrix_heatmap.png')
plt.show()
```

RESULT
Epoch 001, Train Loss: 0.2802, Train Acc: 0.8921, Val Loss: 0.2783, Val Acc: 0.8943
Epoch 002, Train Loss: 0.1178, Train Acc: 0.9670, Val Loss: 0.1151, Val Acc: 0.9685
Epoch 003, Train Loss: 0.0476, Train Acc: 0.9861, Val Loss: 0.0477, Val Acc: 0.9850
Epoch 004, Train Loss: 0.0463, Train Acc: 0.9861, Val Loss: 0.0422, Val Acc: 0.9876
Epoch 005, Train Loss: 0.0395, Train Acc: 0.9886, Val Loss: 0.0400, Val Acc: 0.9886
Epoch 006, Train Loss: 0.0990, Train Acc: 0.9657, Val Loss: 0.0934, Val Acc: 0.9657
Epoch 007, Train Loss: 0.0453, Train Acc: 0.9859, Val Loss: 0.0416, Val Acc: 0.9863
Epoch 008, Train Loss: 0.0289, Train Acc: 0.9918, Val Loss: 0.0286, Val Acc: 0.9902
Epoch 009, Train Loss: 0.0321, Train Acc: 0.9906, Val Loss: 0.0315, Val Acc: 0.9909
Epoch 010, Train Loss: 0.0184, Train Acc: 0.9950, Val Loss: 0.0171, Val Acc: 0.9946
Epoch 011, Train Loss: 0.0226, Train Acc: 0.9932, Val Loss: 0.0208, Val Acc: 0.9938
Epoch 012, Train Loss: 0.0268, Train Acc: 0.9914, Val Loss: 0.0285, Val Acc: 0.9925
Epoch 013, Train Loss: 0.0218, Train Acc: 0.9930, Val Loss: 0.0205, Val Acc: 0.9935
Epoch 014, Train Loss: 0.0134, Train Acc: 0.9970, Val Loss: 0.0140, Val Acc: 0.9974
Epoch 015, Train Loss: 0.0165, Train Acc: 0.9957, Val Loss: 0.0164, Val Acc: 0.9963
Epoch 016, Train Loss: 0.0316, Train Acc: 0.9904, Val Loss: 0.0332, Val Acc: 0.9902
Epoch 017, Train Loss: 0.0240, Train Acc: 0.9919, Val Loss: 0.0258, Val Acc: 0.9915
Epoch 018, Train Loss: 0.0137, Train Acc: 0.9961, Val Loss: 0.0159, Val Acc: 0.9964
Epoch 019, Train Loss: 0.0209, Train Acc: 0.9939, Val Loss: 0.0239, Val Acc: 0.9925
Epoch 020, Train Loss: 0.0206, Train Acc: 0.9935, Val Loss: 0.0184, Val Acc: 0.9941



Test Accuracy: 0.9958
Classification Report:

	precision	recall	f1-score	support
human	0.996730	0.986408	0.991542	1545.000000
amazon	0.995516	1.000000	0.997753	1554.000000
azure	0.998040	1.000000	0.999019	1528.000000
gTTS	0.992810	0.996719	0.994761	1524.000000
accuracy	0.995773	0.995773	0.995773	0.995773
macro avg	0.995774	0.995782	0.995769	6151.000000
weighted avg	0.995778	0.995773	0.995766	6151.000000

Confusion Matrix:

```
[[1524    7    3   11]
 [   0 1554    0    0]
 [   0    0 1528    0]
 [   5    0    0 1519]]
```

ROC-AUC Score: 0.9999

F1 Score: 0.9958

Recall Score: 0.9958