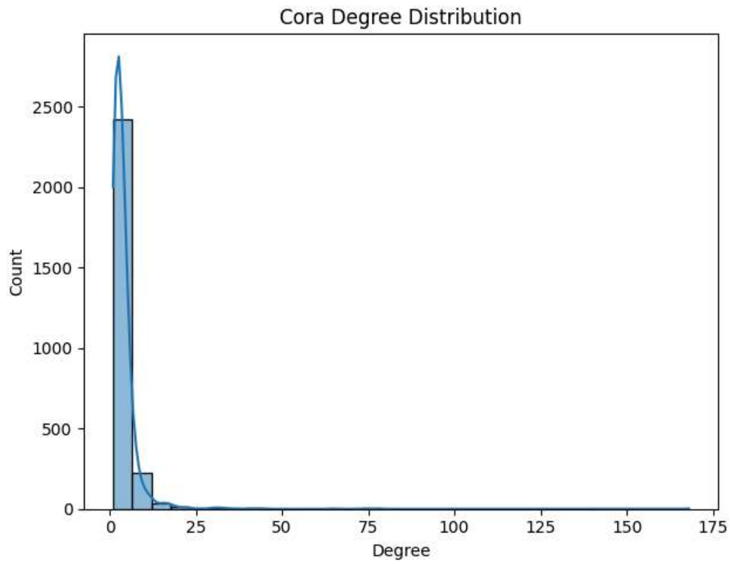
```
!pip install torch_geometric
```

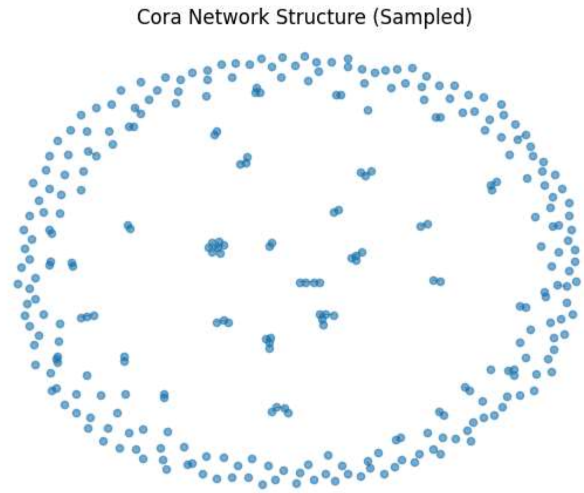
```
Collecting torch_geometric
  Downloading torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.1/63.1 kB 4.0 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (3.11.15)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (2025.3.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (2.0.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (3.2.3)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (4.67.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (2.6.1
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (6.4.3)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (1.19.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch_geometric) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->torch_geometric) (3.4
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->torch_geometric) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->torch_geometric) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->torch_geometric) (2025.1.31
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 17.2 MB/s eta 0:00:00
Installing collected packages: torch_geometric
Successfully installed torch_geometric-2.6.1
```

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.metrics import classification_report, confusion_matrix

# Deep learning libraries
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.datasets import Planetoid, FacebookPagePage
from torch_geometric.utils import to_networkx
from torch_geometric.nn import GCNConv, GATConv, HeteroConv, SAGEConv
from torch_geometric.data import HeteroData
from torch_geometric.transforms import NormalizeFeatures

# Set random seed for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Check for GPU availability
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Load datasets
def load_datasets():
    # Cora dataset (homogeneous)
    cora = Planetoid(root='data/Cora', name='Cora')

    # Facebook dataset
    facebook = FacebookPagePage(root='data/Facebook')

    # Twitter dataset is not available in PyTorch Geometric, so we'll create synthetic data
    twitter = None  # We'll handle this in the HetGNN section

    return cora, facebook, twitter

cora, facebook, twitter = load_datasets()

# Dataset analysis function (modified to handle None dataset)
def analyze_dataset(dataset, name):
    if dataset is None:
        print(f"\n{name} Dataset not available, will use synthetic data for HetGNN")
        return None
```

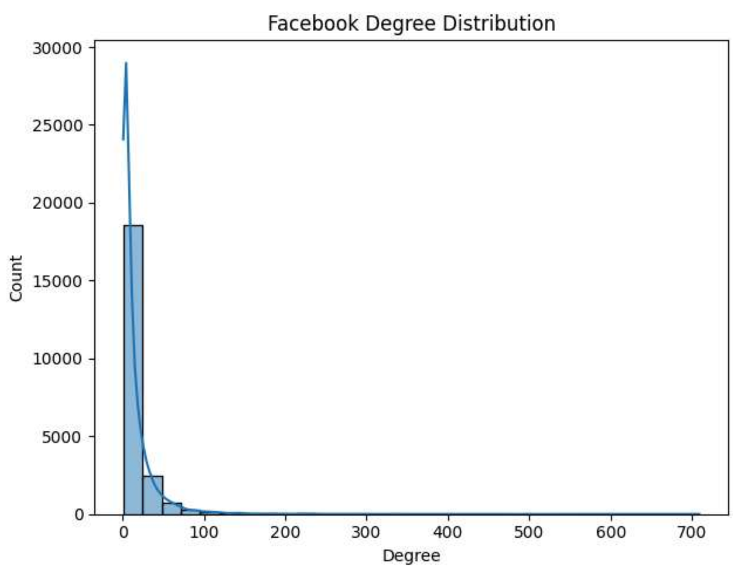
```
    print(f"\n{name} Dataset Info:")
    print("====================")
    print(f"Number of nodes: {dataset[0].num_nodes}")
    print(f"Number of edges: {dataset[0].num_edges}")
    print(f"Number of node features: {dataset[0].num_node_features}")
    print(f"Number of classes: {dataset[0].y.unique().shape[0]}")
    if hasattr(dataset[0], 'edge_type'):
        print(f"Heterogeneous graph with {dataset[0].edge_type.unique().shape[0]} edge types")

    # Plot degree distribution
    g = to_networkx(dataset[0], to_undirected=True)
    degrees = [d for n, d in g.degree()]

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    sns.histplot(degrees, bins=30, kde=True)
    plt.title(f'{name} Degree Distribution')
    plt.xlabel('Degree')
    plt.ylabel('Count')

    # Plot graph (sampled for large networks)
    plt.subplot(1, 2, 2)
    if len(g) > 1000:
        sampled_nodes = np.random.choice(list(g.nodes()), 300, replace=False)
        sub_g = g.subgraph(sampled_nodes)
    else:
        sub_g = g

    pos = nx.spring_layout(sub_g, seed=42)
    nx.draw(sub_g, pos, node_size=20, alpha=0.6, width=0.5)
    plt.title(f'{name} Network Structure (Sampled)')
    plt.tight_layout()
    plt.show()

    return g

# Analyze each dataset
print("Loading and analyzing datasets...")
cora_g = analyze_dataset(cora, "Cora")
facebook_g = analyze_dataset(facebook, "Facebook")
analyze_dataset(twitter, "Twitter")
```

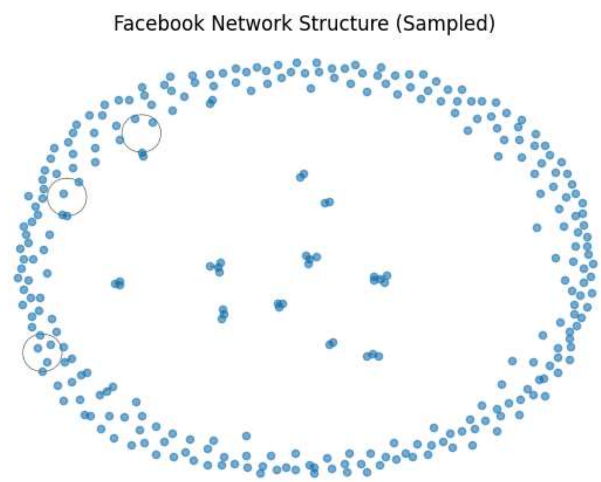
```
Using device: cpu
Loading and analyzing datasets...

Cora Dataset Info:
====================
Number of nodes: 2708
Number of edges: 10556
Number of node features: 1433
Number of classes: 7
```






```
Facebook Dataset Info:
====================
Number of nodes: 22470
Number of edges: 342004
Number of node features: 128
Number of classes: 4
```





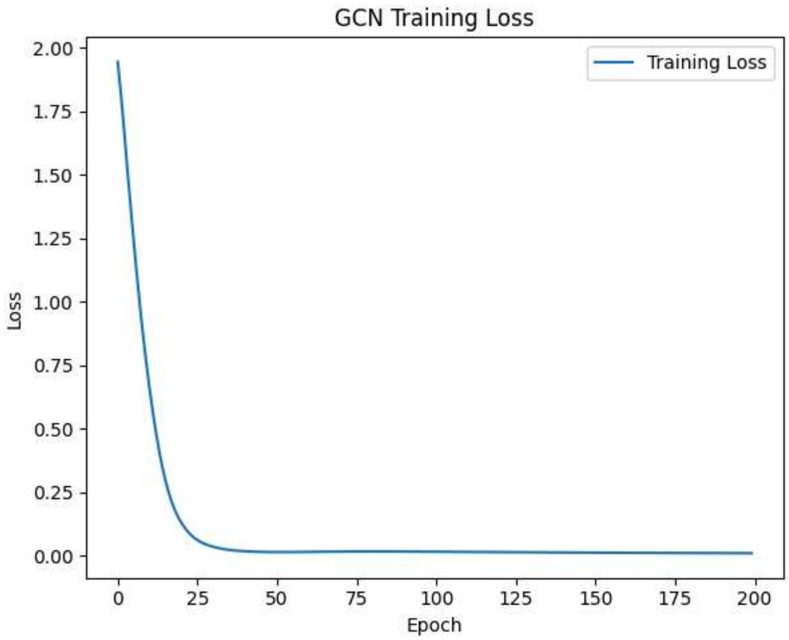

```
Twitter Dataset not available, will use synthetic data for HetGNN
```

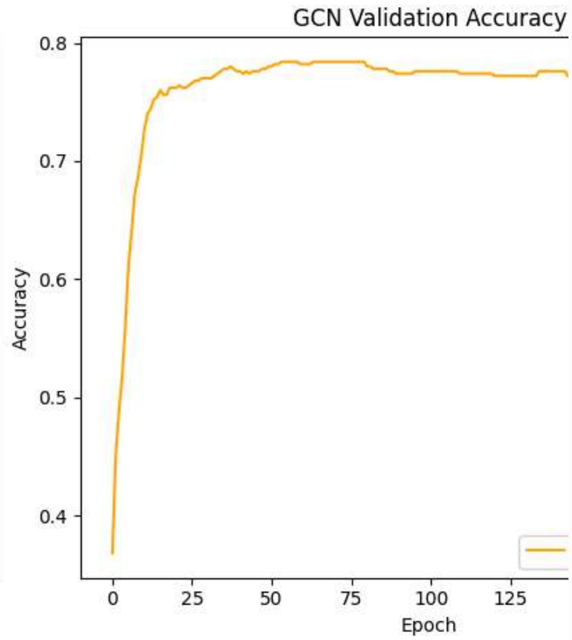
```
class GCN(torch.nn.Module):
    def __init__(self, num_features, hidden_channels, num_classes):
        super().__init__()
        self.conv1 = GCNConv(num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, num_classes)

    def forward(self, x, edge_index):
```

```
    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

def train_gcn(model, data, epochs=200):
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
    criterion = nn.NLLLoss()

    train_losses = []
    val_accuracies = []

    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(data.x, data.edge_index)
        loss = criterion(out[data.train_mask], data.y[data.train_mask])
        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())
        val_acc = test_gcn(model, data, data.val_mask)
        val_accuracies.append(val_acc)

        if epoch % 50 == 0:
            print(f'Epoch: {epoch:03d}, Loss: {loss.item():.4f}, Val Acc: {val_acc:.4f}')

    return train_losses, val_accuracies

def test_gcn(model, data, mask):
    model.eval()
    with torch.no_grad():
        out = model(data.x, data.edge_index)
        pred = out.argmax(dim=1)
        correct = pred[mask] == data.y[mask]
        acc = int(correct.sum()) / int(mask.sum())
    return acc

# Prepare Cora dataset for GCN
data = cora[0].to(device)
data.x = data.x.to(device)
data.edge_index = data.edge_index.to(device)
data.y = data.y.to(device)

# Initialize and train GCN
gcn = GCN(num_features=data.num_features,
          hidden_channels=16,
          num_classes=data.y.unique().shape[0]).to(device)

print("\nTraining GCN on Cora dataset...")
train_losses, val_accuracies = train_gcn(gcn, data)

# Plot training results
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
plt.title('GCN Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(val_accuracies, label='Validation Accuracy', color='orange')
plt.title('GCN Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()

# Test GCN
test_acc = test_gcn(gcn, data, data.test_mask)
print(f'\nGCN Test Accuracy: {test_acc:.4f}')

# Visualize embeddings
def visualize_embeddings(model, data):
```

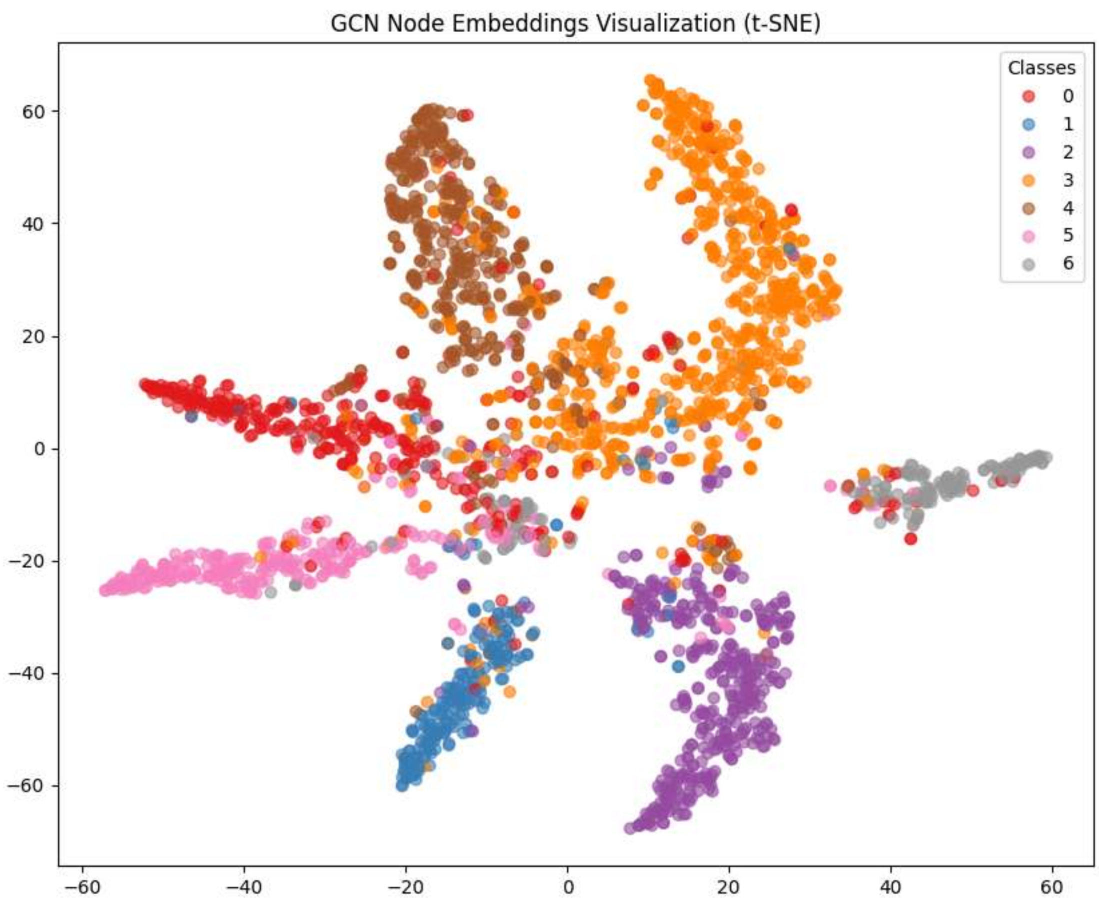
```
    model.eval()
    with torch.no_grad():
        out = model(data.x, data.edge_index)
        h = out.cpu().numpy()

    # Reduce dimensions with t-SNE
    tsne = TSNE(n_components=2, random_state=42)
    h_2d = tsne.fit_transform(h)

    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(h_2d[:, 0], h_2d[:, 1], c=data.y.cpu(), cmap='Set1', alpha=0.6)
    plt.legend(*scatter.legend_elements(), title="Classes")
    plt.title('GCN Node Embeddings Visualization (t-SNE)')
    plt.show()

visualize_embeddings(gcn, data)
```

```
Training GCN on Cora dataset...
Epoch: 000, Loss: 1.9457, Val Acc: 0.3680
Epoch: 050, Loss: 0.0152, Val Acc: 0.7800
Epoch: 100, Loss: 0.0164, Val Acc: 0.7760
Epoch: 150, Loss: 0.0126, Val Acc: 0.7720
```





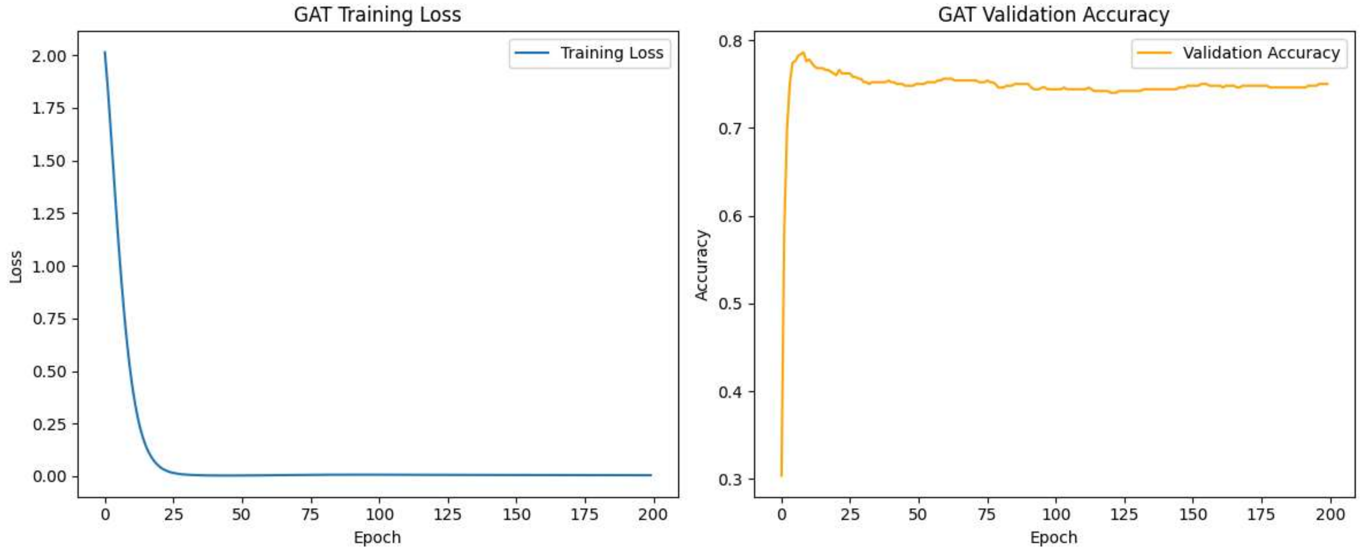

```
GCN Test Accuracy: 0.8040
```



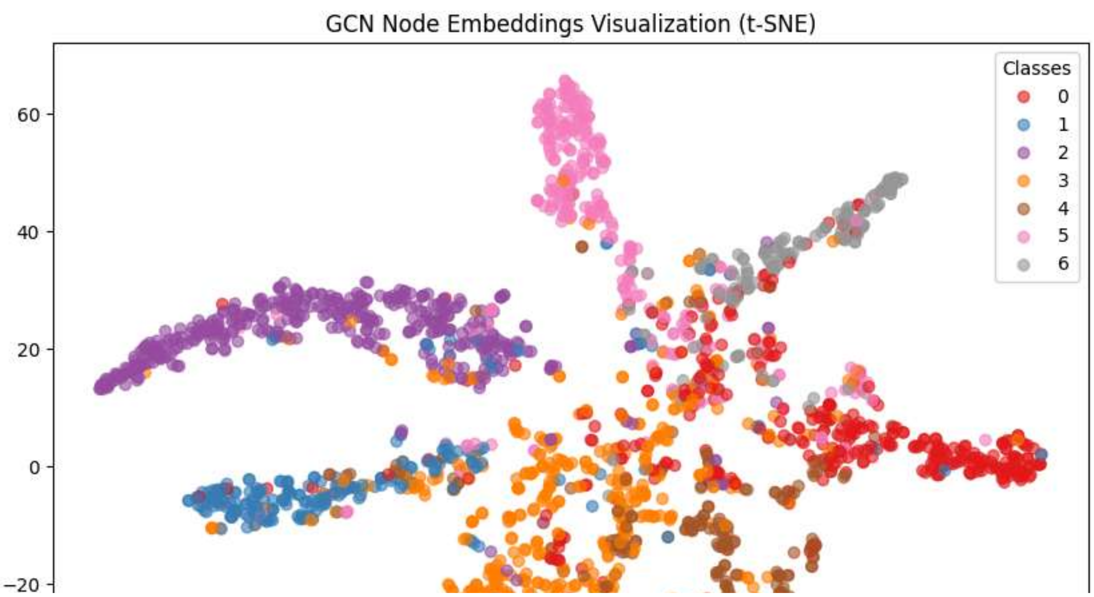

```
class GAT(torch.nn.Module):
    def __init__(self, num_features, hidden_channels, num_classes, heads=8):
        super().__init__()
        self.conv1 = GATConv(num_features, hidden_channels, heads=heads, dropout=0.6)
        self.conv2 = GATConv(hidden_channels * heads, num_classes, heads=1, concat=False, dropout=0.6)
```

```
    def forward(self, x, edge_index):
        x = F.dropout(x, p=0.6, training=self.training)
        x = self.conv1(x, edge_index)
        x = F.elu(x)
        x = F.dropout(x, p=0.6, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

def train_gat(model, data, epochs=200):
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.005, weight_decay=5e-4)
    criterion = nn.NLLLoss()

    train_losses = []
    val_accuracies = []

    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(data.x, data.edge_index)
        loss = criterion(out[data.train_mask], data.y[data.train_mask])
        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())
        val_acc = test_gat(model, data, data.val_mask)
        val_accuracies.append(val_acc)

        if epoch % 50 == 0:
            print(f'Epoch: {epoch:03d}, Loss: {loss.item():.4f}, Val Acc: {val_acc:.4f}')

    return train_losses, val_accuracies

def test_gat(model, data, mask):
    model.eval()
    with torch.no_grad():
        out = model(data.x, data.edge_index)
        pred = out.argmax(dim=1)
        correct = pred[mask] == data.y[mask]
        acc = int(correct.sum()) / int(mask.sum())
    return acc

# Initialize and train GAT
gat = GAT(num_features=data.num_features,
          hidden_channels=8,
          num_classes=data.y.unique().shape[0],
          heads=8).to(device)

print("\nTraining GAT on Cora dataset...")
train_losses_gat, val_accuracies_gat = train_gat(gat, data)

# Plot training results
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses_gat, label='Training Loss')
plt.title('GAT Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(val_accuracies_gat, label='Validation Accuracy', color='orange')
plt.title('GAT Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()

# Test GAT
test_acc_gat = test_gat(gat, data, data.test_mask)
print(f'\nGAT Test Accuracy: {test_acc_gat:.4f}')

# Visualize GAT embeddings
visualize_embeddings(gat, data)
```

```
Training GAT on Cora dataset...
Epoch: 000, Loss: 2.0130, Val Acc: 0.3040
Epoch: 050, Loss: 0.0037, Val Acc: 0.7500
Epoch: 100, Loss: 0.0074, Val Acc: 0.7440
Epoch: 150, Loss: 0.0058, Val Acc: 0.7480
```




```
GAT Test Accuracy: 0.7780
```



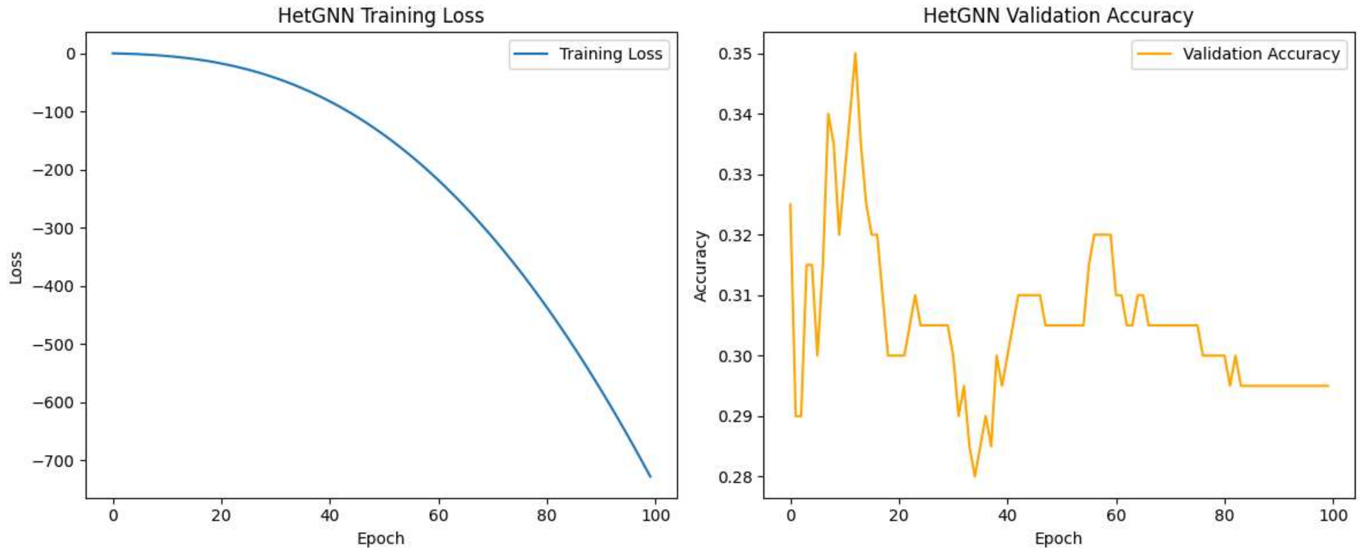

```
def create_hetero_data():
    data = HeteroData()

    # User nodes
    num_users = 1000
    data['user'].x = torch.randn(num_users, 64)  # User features
    data['user'].y = torch.randint(0, 3, (num_users,))  # User classes

    # Post nodes
    num_posts = 5000
    data['post'].x = torch.randn(num_posts, 128)  # Post features

    # Create relationships
    # User-post (written) relationships
    written_edges = torch.randint(0, num_users, (2, 3000))
    data['user', 'writes', 'post'].edge_index = written_edges

    # User-user (follows) relationships
    follow_edges = torch.randint(0, num_users, (2, 5000))
    data['user', 'follows', 'user'].edge_index = follow_edges

    # Post-post (reply) relationships
    reply_edges = torch.randint(0, num_posts, (2, 2000))
```

```
    data['post', 'reply_to', 'post'].edge_index = reply_edges

    # Add some masks
    data['user'].train_mask = torch.zeros(num_users, dtype=torch.bool)
    data['user'].val_mask = torch.zeros(num_users, dtype=torch.bool)
    data['user'].test_mask = torch.zeros(num_users, dtype=torch.bool)

    # Select random samples for masks
    idx = torch.randperm(num_users)
    data['user'].train_mask[idx[:600]] = True
    data['user'].val_mask[idx[600:800]] = True
    data['user'].test_mask[idx[800:]] = True

    return data

hetero_data = create_hetero_data().to(device)

class HetGNN(torch.nn.Module):
    def __init__(self, hidden_channels, num_classes, metadata):
        super().__init__()
        self.convs = torch.nn.ModuleDict()

        # Get input dimensions for each node type
        in_channels_user = hetero_data['user'].x.size(1)
        in_channels_post = hetero_data['post'].x.size(1)

        # Create separate GNN layers for each edge type
        for edge_type in metadata[1]:
            if edge_type == ('user', 'follows', 'user'):
                self.convs['user_follows_user'] = GATConv(in_channels_user, hidden_channels)
            elif edge_type == ('user', 'writes', 'post'):
                self.convs['user_writes_post'] = SAGEConv((in_channels_user, in_channels_post), hidden_channels)
            elif edge_type == ('post', 'reply_to', 'post'):
                self.convs['post_reply_to_post'] = GCNConv(in_channels_post, hidden_channels)

        self.lin = torch.nn.Linear(hidden_channels, num_classes)

    def forward(self, x_dict, edge_index_dict):
        # Apply separate convolutions for each edge type
        out_dict = {}

        # Process user-follows-user relationships
        if 'user_follows_user' in self.convs:
            edge_type = ('user', 'follows', 'user')
            out = self.convs['user_follows_user'](x_dict['user'], edge_index_dict[edge_type])
            if 'user' in out_dict:
                out_dict['user'] = (out_dict['user'] + out) / 2
            else:
                out_dict['user'] = out

        # Process user-writes-post relationships
        if 'user_writes_post' in self.convs:
            edge_type = ('user', 'writes', 'post')
            out = self.convs['user_writes_post'](
                (x_dict['user'], x_dict['post']),
                edge_index_dict[edge_type]
            )
            if 'post' in out_dict:
                out_dict['post'] = (out_dict['post'] + out) / 2
            else:
                out_dict['post'] = out

        # Process post-reply_to-post relationships
        if 'post_reply_to_post' in self.convs:
            edge_type = ('post', 'reply_to', 'post')
            out = self.convs['post_reply_to_post'](x_dict['post'], edge_index_dict[edge_type])
            if 'post' in out_dict:
                out_dict['post'] = (out_dict['post'] + out) / 2
            else:
                out_dict['post'] = out

        # We're only classifying users in this example
        return self.lin(out_dict['user'])

    def encode(self, x_dict, edge_index_dict):
        # Get node embeddings
        out_dict = {}
```

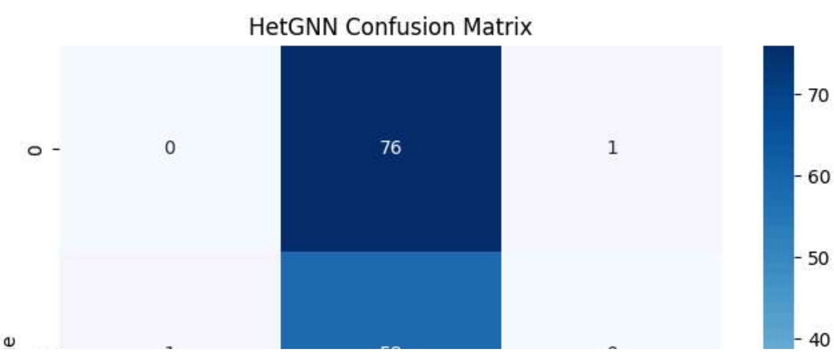
```
        # Process user-follows-user relationships
        if 'user_follows_user' in self.convs:
            edge_type = ('user', 'follows', 'user')
            out = self.convs['user_follows_user'](x_dict['user'], edge_index_dict[edge_type])
            if 'user' in out_dict:
                out_dict['user'] = (out_dict['user'] + out) / 2
            else:
                out_dict['user'] = out

        # Process user-writes-post relationships
        if 'user_writes_post' in self.convs:
            edge_type = ('user', 'writes', 'post')
            out = self.convs['user_writes_post'](
                (x_dict['user'], x_dict['post']),
                edge_index_dict[edge_type]
            )
            if 'post' in out_dict:
                out_dict['post'] = (out_dict['post'] + out) / 2
            else:
                out_dict['post'] = out

        # Process post-reply_to-post relationships
        if 'post_reply_to_post' in self.convs:
            edge_type = ('post', 'reply_to', 'post')
            out = self.convs['post_reply_to_post'](x_dict['post'], edge_index_dict[edge_type])
            if 'post' in out_dict:
                out_dict['post'] = (out_dict['post'] + out) / 2
            else:
                out_dict['post'] = out

        return out_dict

def final_test_hetgnn(model, data):
    """
    Evaluates the HetGNN model and returns test accuracy, confusion matrix, and classification report.
    """
    model.eval()
    with torch.no_grad():
        out = model(data.x_dict, data.edge_index_dict)
        pred = out[data['user'].test_mask].argmax(dim=1)
        y_true = data['user'].y[data['user'].test_mask]

        acc = int(pred.eq(y_true).sum()) / int(data['user'].test_mask.sum())

        cm = confusion_matrix(y_true.cpu(), pred.cpu())
        report = classification_report(y_true.cpu(), pred.cpu())

    return acc, cm, report

def train_hetgnn(model, data, epochs=100):
    """
    Trains the HetGNN model and returns training losses and validation accuracies.
    """
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
    criterion = nn.NLLLoss()

    train_losses = []
    val_accuracies = []

    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(data.x_dict, data.edge_index_dict)

        # Calculate loss only on the labeled user nodes in the training set
        loss = criterion(out[data['user'].train_mask], data['user'].y[data['user'].train_mask])

        loss.backward()
        optimizer.step()

        train_losses.append(loss.item())

        # Calculate validation accuracy
        val_acc, _, _ = final_test_hetgnn(model, data)  # Use the test function with the validation mask
        val_accuracies.append(val_acc)
```

```
        # Print progress every 50 epochs
        if epoch % 50 == 0:
            print(f'Epoch: {epoch:03d}, Loss: {loss.item():.4f}, Val Acc: {val_acc:.4f}')

    return train_losses, val_accuracies
def visualize_hetero_embeddings(model, data):
    """
    Visualizes embeddings for the HetGNN model
    """
    model.eval()
    with torch.no_grad():
        embeddings = model.encode(data.x_dict, data.edge_index_dict)
        user_embeddings = embeddings['user'].cpu().numpy()

    tsne = TSNE(n_components=2, random_state=42)
    h_2d = tsne.fit_transform(user_embeddings)

    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(h_2d[:, 0], h_2d[:, 1], c=data['user'].y.cpu(), cmap='Set1', alpha=0.6)
    plt.legend(*scatter.legend_elements(), title="Classes")
    plt.title('HetGNN Node Embeddings Visualization (t-SNE)')
    plt.show()

# Initialize and train HetGNN
metadata = hetero_data.metadata()
hetgnn = HetGNN(hidden_channels=32,
            num_classes=hetero_data['user'].y.unique().shape[0],
            metadata=metadata).to(device)

print("\nTraining HetGNN on synthetic heterogeneous dataset...")
train_losses_het, val_accuracies_het = train_hetgnn(hetgnn, hetero_data)

# Plot training results
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses_het, label='Training Loss')
plt.title('HetGNN Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(val_accuracies_het, label='Validation Accuracy', color='orange')
plt.title('HetGNN Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()

# Test HetGNN
test_acc_het, cm_het, report_het = final_test_hetgnn(hetgnn, hetero_data)
print(f'\nHetGNN Test Accuracy: {test_acc_het:.4f}')
print("\nClassification Report:")
print(report_het)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_het, annot=True, fmt='d', cmap='Blues')
plt.title('HetGNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Visualize heterogeneous embeddings
visualize_hetero_embeddings(hetgnn, hetero_data)
```

```
Training HetGNN on synthetic heterogeneous dataset...
Epoch: 000, Loss: -0.0571, Val Acc: 0.3250
Epoch: 050, Loss: -140.5761, Val Acc: 0.3050
```




```
HetGNN Test Accuracy: 0.2950

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        77
           1       0.30      0.98      0.46        59
           2       0.50      0.02      0.03        64

    accuracy                           0.29       200
   macro avg       0.27      0.33      0.16       200
weighted avg       0.25      0.29      0.14       200
```




```
# Compare model performances
models = ['GCN', 'GAT', 'HetGNN']
test_accs = [test_acc, test_acc_gat, test_acc_het]

plt.figure(figsize=(8, 6))
plt.bar(models, test_accs, color=['blue', 'orange', 'green'])
plt.title('Model Comparison on Test Accuracy')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
for i, v in enumerate(test_accs):
    plt.text(i, v + 0.02, f"{v:.4f}", ha='center')
plt.show()

# Compare training curves
plt.figure(figsize=(10, 6))
plt.plot(train_losses, label='GCN Training Loss')
plt.plot(train_losses_gat, label='GAT Training Loss')
plt.plot(train_losses_het, label='HetGNN Training Loss')
plt.title('Training Loss Comparison')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
plt.figure(figsize=(10, 6))
plt.plot(val_accuracies, label='GCN Validation Accuracy')
plt.plot(val_accuracies_gat, label='GAT Validation Accuracy')
plt.plot(val_accuracies_het, label='HetGNN Validation Accuracy')
plt.title('Validation Accuracy Comparison')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Community detection comparison (using GCN, GAT, and HetGNN embeddings)
def detect_communities(embeddings, true_labels, title):
    from sklearn.cluster import KMeans

    # Cluster embeddings
    kmeans = KMeans(n_clusters=true_labels.unique().shape[0], random_state=42)
    clusters = kmeans.fit_predict(embeddings)

    # Calculate adjusted rand score
    from sklearn.metrics import adjusted_rand_score
    ari = adjusted_rand_score(true_labels.cpu(), clusters)

    # Plot clusters
    tsne = TSNE(n_components=2, random_state=42)
    emb_2d = tsne.fit_transform(embeddings)

    plt.figure(figsize=(10, 8))
    plt.scatter(emb_2d[:, 0], emb_2d[:, 1], c=clusters, cmap='Set1', alpha=0.6)
    plt.title(f'{title} - Detected Communities (ARI: {ari:.4f})')
    plt.show()

    return ari
```