```
!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 4.8 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (3.11.14)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch_geometric) (2025.3.0)
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_geometr
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (1.
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (25.3.
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (1
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (0.
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->torch_geometric) (1
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch_geometric) (3.0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->torch_geome
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)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->torch_geometric)
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 50.8 MB/s eta 0:00:00
Installing collected packages: torch_geometric
Successfully installed torch_geometric-2.6.1
```

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, GATConv
from torch_geometric.datasets import Planetoid
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
from torch_geometric.data import Data, DataLoader, Dataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
!rm QG_jets.npz
!wget "https://zenodo.org/records/3164691/files/QG_jets.npz?download=1" -O QG_jets_new.npz
```

```
--2025-04-02 05:15:31--  https://zenodo.org/records/3164691/files/QG_jets.npz?download=1
Resolving zenodo.org (zenodo.org)... 188.185.45.92, 188.185.48.194, 188.185.43.25, ...
Connecting to zenodo.org (zenodo.org)|188.185.45.92|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 106689379 (102M) [application/octet-stream]
Saving to: ‘QG_jets_new.npz’

QG_jets_new.npz     100%[===================>] 101.75M  12.1MB/s    in 9.7s

2025-04-02 05:15:42 (10.5 MB/s) - ‘QG_jets_new.npz’ saved [106689379/106689379]
```

```
data_path = '/content/QG_jets_new.npz'
data = np.load(data_path, allow_pickle=True)

X = data['X']
y = data['y']

# Define a function to construct edge indices based on proximity
def construct_edge_index(particle_features, k=3):
    """
    Constructs edge indices for a k-nearest neighbors graph.
    particle_features: np.array of shape (num_particles, num_features)
    k: number of nearest neighbors
    """
    from sklearn.neighbors import NearestNeighbors
    num_particles = particle_features.shape[0]
    if num_particles < k + 1:
        k = num_particles - 1
    nbrs = NearestNeighbors(n_neighbors=k+1).fit(particle_features)
    distances, indices = nbrs.kneighbors(particle_features)
    edge_index = []
    for i in range(num_particles):
```

```
        for j in range(1, k+1):
            edge_index.append((i, indices[i, j]))
    edge_index = np.array(edge_index).T
    return torch.tensor(edge_index, dtype=torch.long)

class ParticleJetDataset(torch.utils.data.Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        x = torch.tensor(self.X[idx], dtype=torch.float)
        edge_index = construct_edge_index(x[:, :3])
        y = torch.tensor(self.y[idx], dtype=torch.long)
        return Data(x=x, edge_index=edge_index, y=y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

train_dataset = ParticleJetDataset(X_train, y_train)
test_dataset = ParticleJetDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

```
from torch_geometric.utils import dense_to_sparse
from torch.utils.data import random_split

class ParticleJetDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = torch.tensor(y, dtype=torch.long)
        self.num_graphs = X.shape[0]

    def __len__(self):
        return self.num_graphs

    def __getitem__(self, idx):
        node_features = torch.tensor(self.X[idx], dtype=torch.float)
        label = self.y[idx]

        edge_index = torch.combinations(torch.arange(node_features.shape[0]), r=2).T
        edge_index = edge_index.to(torch.long)

        return Data(x=node_features, edge_index=edge_index, y=label, num_nodes=node_features.shape[0])

dataset = ParticleJetDataset(X, y)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

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

print(f"Training Samples: {len(train_dataset)}, Testing Samples: {len(test_dataset)}")
```

```
Training Samples: 80000, Testing Samples: 20000
```

```
from torch_geometric.nn import global_mean_pool

# class GCN(torch.nn.Module):
#     def __init__(self, in_channels, hidden_channels, out_channels):
#         super(GCN, self).__init__()
#         self.conv1 = GCNConv(in_channels, hidden_channels)
#         self.conv2 = GCNConv(hidden_channels, out_channels)

#     def forward(self, x, edge_index):
#         x = F.relu(self.conv1(x, edge_index))
#         x = F.dropout(x, p=0.5, training=self.training)
#         x = self.conv2(x, edge_index)
#         return F.log_softmax(x, dim=1)
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
```

```
        self.fc = torch.nn.Linear(hidden_channels, out_channels)

    def forward(self, x, edge_index, batch):
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))

        x = global_mean_pool(x, batch)
        x = self.fc(x)
        return F.log_softmax(x, dim=1)

class GAT(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GAT, self).__init__()
        self.conv1 = GATConv(in_channels, hidden_channels, heads=4, dropout=0.6)
        self.conv2 = GATConv(hidden_channels * 4, hidden_channels, heads=1, dropout=0.6)
        self.fc = torch.nn.Linear(hidden_channels, out_channels)

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



# def train_model(model, train_loader, optimizer):
#     model.train()
#     total_loss = 0
#     for data in train_loader:
#         optimizer.zero_grad()
#         output = model(data.x, data.edge_index)
#         loss = F.nll_loss(output[data.batch], data.y)
#         loss.backward()
#         optimizer.step()
#         total_loss += loss.item()
#     return total_loss / len(train_loader)

# def evaluate_model(model, test_loader):
#     model.eval()
#     correct = 0
#     for data in test_loader:
#         output = model(data.x, data.edge_index)
#         pred = output.argmax(dim=1)
#         correct += pred.eq(data.y).sum().item()
#     return correct / len(test_loader.dataset)
def train_model(model, train_loader, optimizer):
    model.train()
    total_loss = 0
    for data in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        output = model(data.x, data.edge_index, data.batch)
        loss = F.nll_loss(output, data.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(train_loader)

def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    for data in test_loader:
        data = data.to(device)

        output = model(data.x, data.edge_index, data.batch)
        pred = output.argmax(dim=1)
        correct += (pred == data.y).sum().item()
        total += data.y.size(0)
    return correct / total


# # Choose model: 'GCN' or 'GAT'
# model_choice = 'GCN'

# if model_choice == 'GCN':
#     model = GCN(in_channels=4, hidden_channels=32, out_channels=2)
# else:
#     model = GAT(in_channels=4, hidden_channels=32, out_channels=2)

# optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
# device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# model = model.to(device)

# # Train and Evaluate
# num_epochs = 10
# for epoch in range(num_epochs):
#     loss = train_model(model, train_loader, optimizer)
#     accuracy = evaluate_model(model, test_loader)
#     print(f"Epoch {epoch + 1}: Loss = {loss:.4f}, Accuracy = {accuracy * 100:.2f}%")

# print("Training complete!")



num_features = X.shape[-1]
models = {'GCN': GCN(num_features, 32, 2), 'GAT': GAT(num_features, 32, 2)}

results = {}
num_epochs = 50
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

for name, model in models.items():
    model.to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(num_epochs):
        loss = train_model(model, train_loader, optimizer)
        acc = evaluate_model(model, test_loader)
        print(f"{name} - Epoch {epoch + 1}: Loss = {loss:.4f}, Accuracy = {acc * 100:.2f}%")

    final_acc = evaluate_model(model, test_loader)
    results[name] = final_acc * 100

print("\n=== Final Model Performance ===")
for name, acc in results.items():
    print(f"{name}: {acc:.2f}% Accuracy")
```

```
GCN - Epoch 1: Loss = 0.6784, Accuracy = 71.60%
GCN - Epoch 2: Loss = 0.5876, Accuracy = 70.06%
GCN - Epoch 3: Loss = 0.5712, Accuracy = 73.75%
GCN - Epoch 4: Loss = 0.5594, Accuracy = 73.64%
GCN - Epoch 5: Loss = 0.5435, Accuracy = 73.98%
GCN - Epoch 6: Loss = 0.5361, Accuracy = 74.63%
GCN - Epoch 7: Loss = 0.5298, Accuracy = 74.05%
GCN - Epoch 8: Loss = 0.5258, Accuracy = 74.61%
GCN - Epoch 9: Loss = 0.5229, Accuracy = 75.16%
GCN - Epoch 10: Loss = 0.5227, Accuracy = 75.67%
GCN - Epoch 11: Loss = 0.5208, Accuracy = 75.21%
GCN - Epoch 12: Loss = 0.5188, Accuracy = 74.78%
GCN - Epoch 13: Loss = 0.5171, Accuracy = 75.14%
GCN - Epoch 14: Loss = 0.5154, Accuracy = 75.12%
GCN - Epoch 15: Loss = 0.5135, Accuracy = 74.59%
GCN - Epoch 16: Loss = 0.5138, Accuracy = 75.14%
GCN - Epoch 17: Loss = 0.5132, Accuracy = 75.79%
GCN - Epoch 18: Loss = 0.5129, Accuracy = 75.47%
GCN - Epoch 19: Loss = 0.5125, Accuracy = 75.62%
GCN - Epoch 20: Loss = 0.5105, Accuracy = 75.73%
GCN - Epoch 21: Loss = 0.5114, Accuracy = 75.69%
GCN - Epoch 22: Loss = 0.5108, Accuracy = 75.30%
GCN - Epoch 23: Loss = 0.5095, Accuracy = 75.30%
GCN - Epoch 24: Loss = 0.5087, Accuracy = 75.77%
GCN - Epoch 25: Loss = 0.5089, Accuracy = 75.47%
GCN - Epoch 26: Loss = 0.5085, Accuracy = 75.16%
GCN - Epoch 27: Loss = 0.5076, Accuracy = 75.97%
GCN - Epoch 28: Loss = 0.5070, Accuracy = 75.48%
GCN - Epoch 29: Loss = 0.5066, Accuracy = 75.16%
GCN - Epoch 30: Loss = 0.5064, Accuracy = 75.89%
GCN - Epoch 31: Loss = 0.5071, Accuracy = 75.73%
GCN - Epoch 32: Loss = 0.5074, Accuracy = 76.05%
GCN - Epoch 33: Loss = 0.5065, Accuracy = 75.81%
GCN - Epoch 34: Loss = 0.5056, Accuracy = 76.13%
GCN - Epoch 35: Loss = 0.5057, Accuracy = 76.33%
GCN - Epoch 36: Loss = 0.5042, Accuracy = 75.98%
GCN - Epoch 37: Loss = 0.5046, Accuracy = 75.41%
GCN - Epoch 38: Loss = 0.5036, Accuracy = 76.17%
GCN - Epoch 39: Loss = 0.5032, Accuracy = 75.94%
GCN - Epoch 40: Loss = 0.5042, Accuracy = 76.19%
GCN - Epoch 41: Loss = 0.5029, Accuracy = 76.44%
GCN - Epoch 42: Loss = 0.5033, Accuracy = 76.28%
GCN - Epoch 43: Loss = 0.5028, Accuracy = 76.28%
GCN - Epoch 44: Loss = 0.5023, Accuracy = 76.02%
GCN - Epoch 45: Loss = 0.5033, Accuracy = 76.05%
GCN - Epoch 46: Loss = 0.5024, Accuracy = 76.06%
GCN - Epoch 47: Loss = 0.5027, Accuracy = 76.44%
GCN - Epoch 48: Loss = 0.5019, Accuracy = 76.46%
GCN - Epoch 49: Loss = 0.5022, Accuracy = 75.69%
GCN - Epoch 50: Loss = 0.5021, Accuracy = 76.15%
```

```
GAT - Epoch 1: Loss = 0.7844, Accuracy = 52.38%
GAT - Epoch 2: Loss = 0.6829, Accuracy = 50.36%
GAT - Epoch 3: Loss = 0.6568, Accuracy = 51.35%
GAT - Epoch 4: Loss = 0.6358, Accuracy = 50.43%
GAT - Epoch 5: Loss = 0.6363, Accuracy = 52.34%
GAT - Epoch 6: Loss = 0.6218, Accuracy = 50.27%
GAT - Epoch 7: Loss = 0.5720, Accuracy = 55.94%
GAT - Epoch 8: Loss = 0.5772, Accuracy = 53.89%
```

Start coding or generate with AI.