Common_Task_02

quark/gluon Classification

In [1]:
```
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 2.0 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (3.11.12)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (2024.12.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (3.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (1.26.4)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (3.2.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (4.67.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (2.4.6)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (1.3.2)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (5.0.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (1.18.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch-geometric) (3.0.2)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (2025.0.1)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (2022.0.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (2.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (2025.1.31)
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from multidict<7.0,>=4.5->aiohttp->torch-geometric) (4.12.2)
Requirement already satisfied: intel-openmp>=2024 in /usr/local/lib/python3.10/dist-packages (from mkl->numpy->torch-geometric) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dist-packages (from mkl->numpy->torch-geometric) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl->numpy->torch-geometric) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.10/dist-packages (from mkl_umath->numpy->torch-geometric) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/lib/python3.10/dist-packages (from intel-openmp>=2024->mkl->numpy->torch-geometric) (2024.2.0)
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 19.8 MB/s eta 0:00:00
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.6.1
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```python
import h5py
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import networkx as nx
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
from torch_geometric.data import Data, Batch
from torch_geometric.loader import DataLoader
from torch.optim import Adam
import pytorch_lightning as pl
import torch.nn.functional as F
from torch.nn import Linear
```

In [3]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
file_path = "/kaggle/input/autoencoder-data/quark-gluon_data-set_n139306.hdf5"
```

In [4]:
```python
print(device)
```

```
cuda
```

In [5]:
```python
def explore_hdf5(file):
    with h5py.File(file, "r") as f:
        print("Keys in dataset:", list(f.keys()))
        for key in f.keys():
            print(f"Shape of {key}: {f[key].shape}")

explore_hdf5(file_path)
```

```
Keys in dataset: ['X_jets', 'm0', 'pt', 'y']
Shape of X_jets: (139306, 125, 125, 3)
Shape of m0: (139306,)
Shape of pt: (139306,)
Shape of y: (139306,)
```

In [6]:
```python
def load_data(file_name, sample_size):
    with h5py.File(file_name, 'r') as f:
        print("Dataset keys:", list(f.keys()))
        print("Total images:", len(f['X_jets']))
        print("Image dimensions:", f['X_jets'].shape[1:])
        return np.array(f['X_jets'][:sample_size]), np.array(f['y'][:sample_size])

X, y = load_data(file_path, 10000)
```

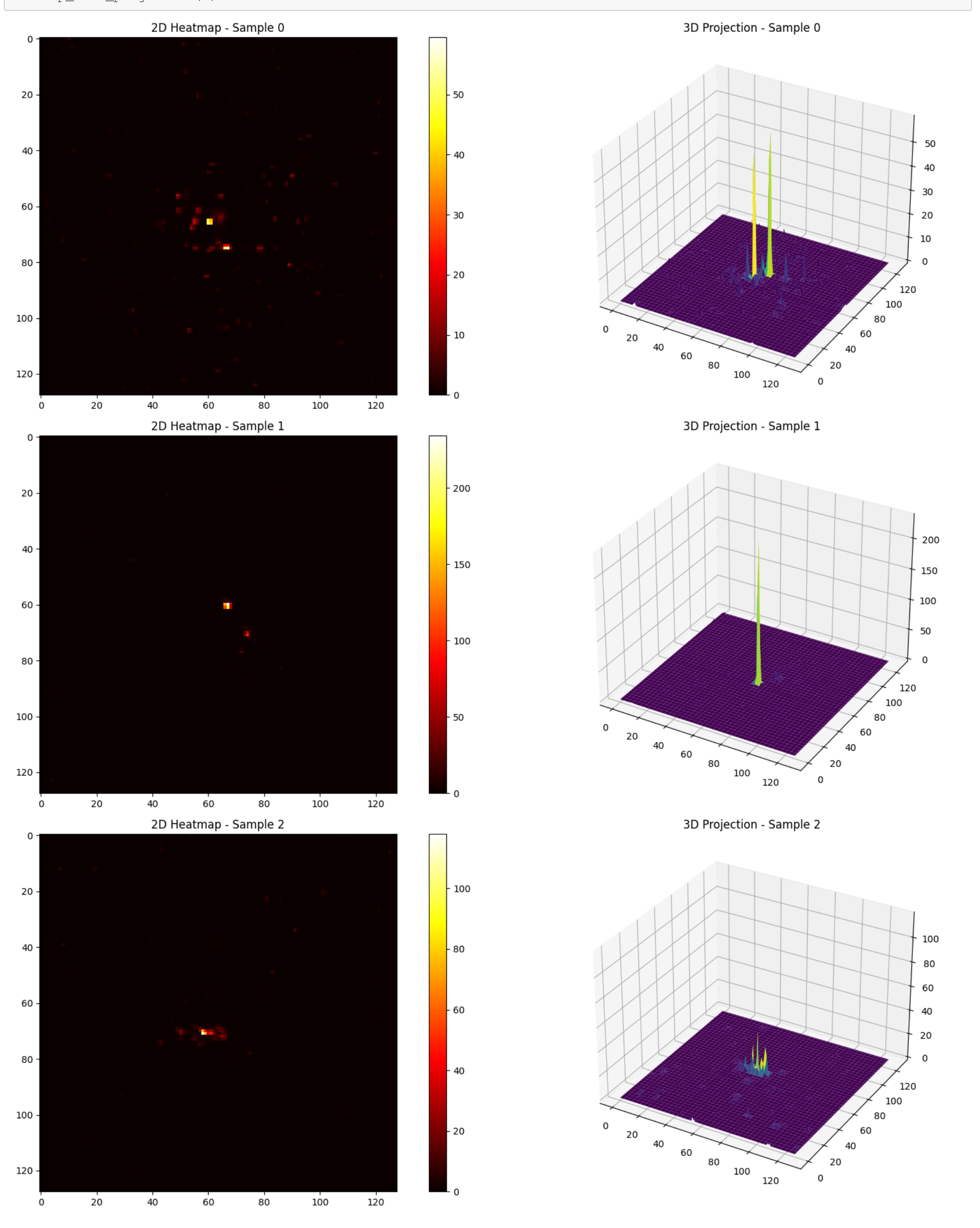
```
Dataset keys: ['X_jets', 'm0', 'pt', 'y']
Total images: 139306
Image dimensions: (125, 125, 3)
```

In [7]:
```python
def count_labels(labels):
    label_counts = np.bincount(labels.astype(np.int64))
    return {str(i): count for i, count in enumerate(label_counts)}

print(count_labels(y))
```

```
{'0': 4994, '1': 5006}
```

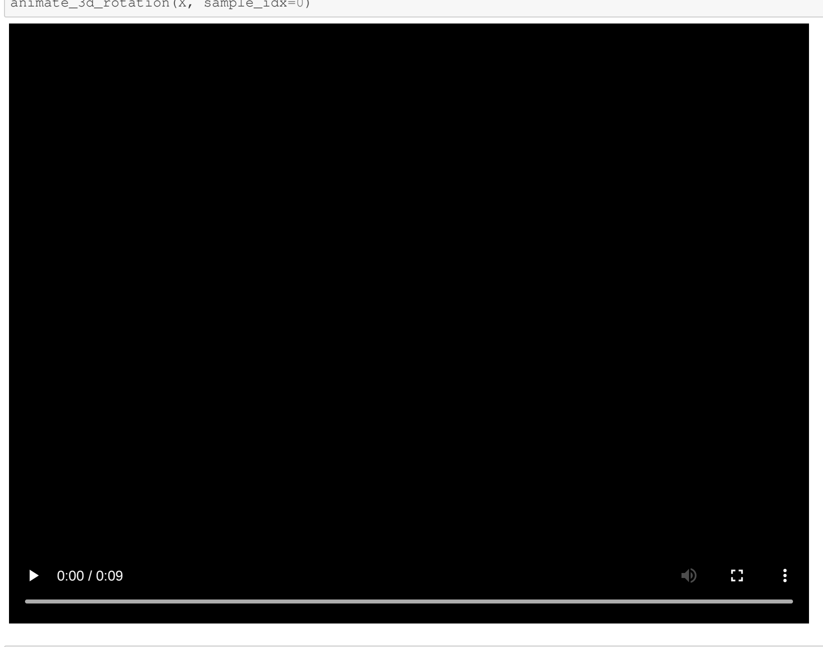
In [8]:
```python
def preprocess_images(images):
    from skimage.transform import resize
    # Resize to 128x128 and normalize
    processed = np.array([resize(img, (128, 128), anti_aliasing=True) for img in images], dtype=np.float32)
    mean, std = np.mean(processed), np.std(processed)
    return np.clip((processed - mean) / std, 0, None)

X = preprocess_images(X)
```

In [9]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def heatmap_with_projection(images, num_samples=3):
    fig = plt.figure(figsize=(18, 6 * num_samples))

    for idx in range(num_samples):
        # 2D Heatmap
        ax1 = fig.add_subplot(num_samples, 2, 2*idx + 1)
        combined_data = np.sum(images[idx], axis=-1)
        heatmap = ax1.imshow(combined_data, cmap='hot', interpolation='nearest')
        plt.colorbar(heatmap, ax=ax1)
        ax1.set_title(f'2D Heatmap - Sample {idx}')

        # 3D Projection
        ax2 = fig.add_subplot(num_samples, 2, 2*idx + 2, projection='3d')
        X, Y = np.meshgrid(np.arange(combined_data.shape[1]), np.arange(combined_data.shape[0]))
        ax2.plot_surface(X, Y, combined_data, cmap='viridis')
        ax2.set_title(f'3D Projection - Sample {idx}')

    plt.tight_layout()
    plt.show()

heatmap_with_projection(X)
```



In [10]:
```python
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

def animate_3d_rotation(images, sample_idx=0):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    combined_data = np.sum(images[sample_idx], axis=-1)
    X, Y = np.meshgrid(np.arange(combined_data.shape[1]), np.arange(combined_data.shape[0]))
    surf = ax.plot_surface(X, Y, combined_data, cmap='viridis')

    def update(frame):
        ax.view_init(elev=20, azim=frame)
        return fig,

    anim = FuncAnimation(fig, update, frames=np.arange(0, 360, 2), interval=50)
    plt.close()
    return HTML(anim.to_html5_video())

animate_3d_rotation(X, sample_idx=0)
```

Out[10]:



In [11]:
```python
def extract_nonzero_mask(images):
    reshaped = images.reshape((-1, images.shape[1] * images.shape[2], 3))
    return np.any(reshaped != [0., 0., 0.], axis=-1).reshape(images.shape[:3])

mask = extract_nonzero_mask(X)
```

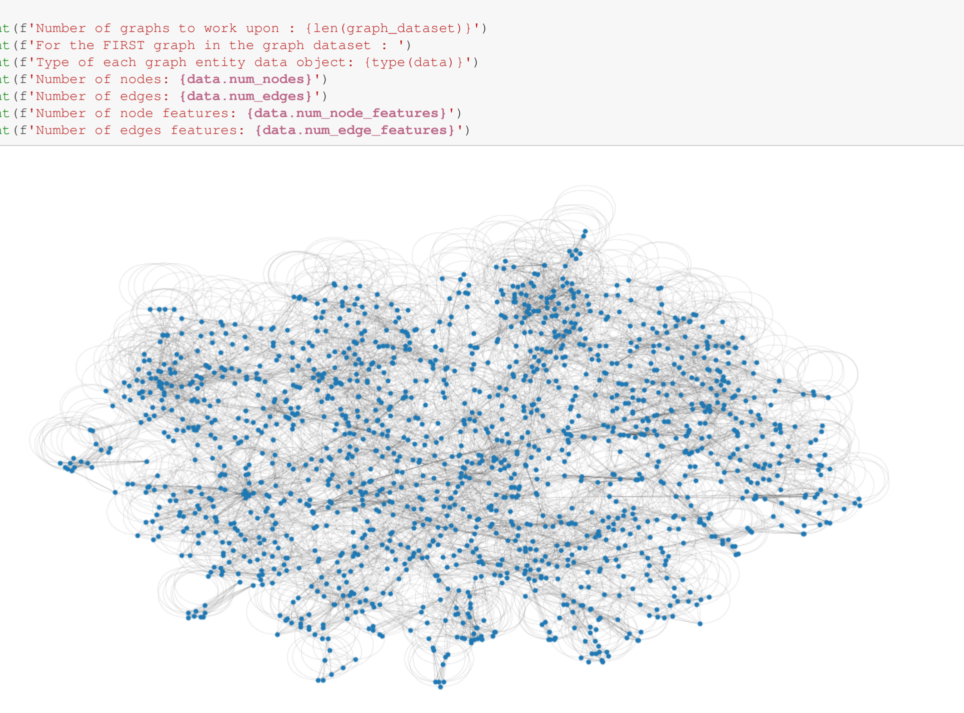
In [12]:
```python
def create_graph_features(masked_data):
    indices, features = [], []
    for img_idx, mask in enumerate(masked_data):
        coords = np.column_stack(np.where(mask))
        features.append(X[img_idx, coords[:, 0], coords[:, 1], :])
        indices.append(coords)
    return indices, features

indices_list, features_list = create_graph_features(mask)
```

In [13]:
```python
def build_graph_structure(coords, k=4):
    from scipy.spatial import cKDTree
    from scipy.sparse import coo_matrix
    tree = cKDTree(coords)
    dist, indices = tree.query(coords, k=k)
    sigma2 = np.mean(dist[:, -1])**2
    weights = np.exp(-dist**2 / sigma2)
    row, col = np.arange(len(coords)).repeat(k), indices.flatten()
    return coo_matrix((weights.flatten(), (row, col)), shape=(len(coords), len(coords)))
```

In [14]:
```python
def create_graph_dataset(indices_list, labels, neighbors=8):
    dataset = []
    for i, points in enumerate(indices_list):
        adjacency = build_graph_structure(points, k=neighbors)
        edge_idx = torch.from_numpy(np.vstack((adjacency.row, adjacency.col))).long()
        edge_weights = torch.from_numpy(adjacency.data).float().view(-1, 1)
        label = torch.tensor([int(labels[i])], dtype=torch.long)
        graph = Data(x=torch.from_numpy(features_list[i]), edge_index=edge_idx, edge_attr=edge_weights, y=label)
        dataset.append(graph)
    return dataset
```

In [15]:
```python
graph_dataset = create_graph_dataset(indices_list, y, neighbors=8)
```

In [16]:
```python
G = nx.Graph()
data = graph_dataset[0]
edge_tensor = data.edge_index
edge_list = [(edge_tensor[0, i].item(), edge_tensor[1, i].item()) for i in range(edge_tensor.shape[1])]
G.add_edges_from(edge_list)
pos = nx.spring_layout(G, iterations=15, seed=1721)
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
nx.draw_networkx(G, pos=pos, ax=ax, node_size=10, with_labels=False, width=0.05)
plt.show()

print(f'Number of graphs to work upon : {len(graph_dataset)}')
print(f'For the FIRST graph in the graph dataset : ')
print(f'Type of each graph entity data object: {type(data)}')
print(f'Number of nodes: {data.num_nodes}')
print(f'Number of edges: {data.num_edges}')
print(f'Number of node features: {data.num_node_features}')
print(f'Number of edges features: {data.num_edge_features}')
```



```
Number of graphs to work upon : 10000
For the FIRST graph in the graph dataset :
Type of each graph entity data object: <class 'torch_geometric.data.data.Data'>
Number of nodes: 1530
Number of edges: 12240
Number of node features: 3
Number of edges features: 1
```

In [17]:
```python
train_data, test_data = train_test_split(graph_dataset, test_size=0.2, random_state=14)
train_data, val_data = train_test_split(train_data, test_size=0.1, random_state=14)
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)
val_loader = DataLoader(val_data, batch_size=128, shuffle=False)
```

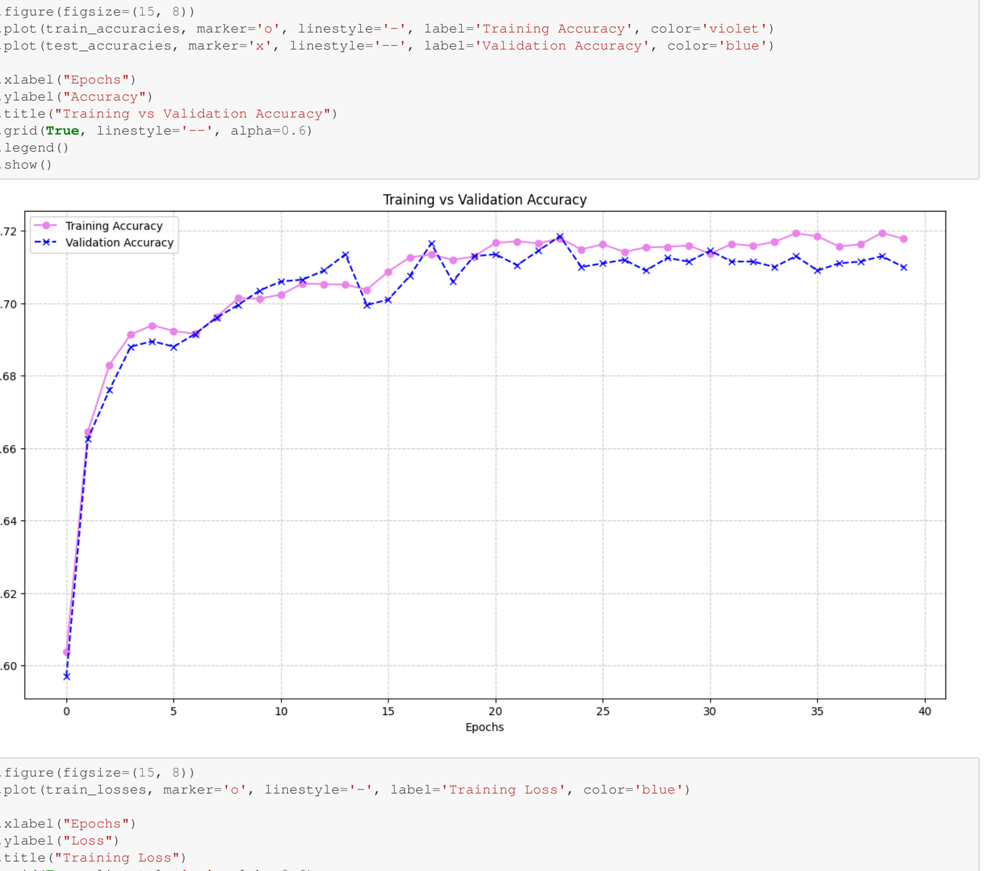
In [18]:
```python
from torch_geometric.nn import GCNConv,global_mean_pool
class GNN(torch.nn.Module):
    def __init__(self, in_features, hidden_dim, num_classes):
        super().__init__()
        self.layers = nn.ModuleList([
            GCNConv(in_features, hidden_dim),
            GCNConv(hidden_dim, 2 * hidden_dim),
            GCNConv(2 * hidden_dim, hidden_dim)
        ])
        self.linear = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim // 4),
            nn.ReLU(),
            nn.Linear(hidden_dim // 4, num_classes)
        )
        self.loss_fn = nn.CrossEntropyLoss()

    def forward(self, x, edge_index, batch):
        for layer in self.layers:
            x = F.relu(layer(x, edge_index))
        x = global_mean_pool(x, batch)
        return self.linear(x)
```

In [19]:
```python
def train(model, loader, optimizer, criterion):
    model.train()
    total_loss = 0
    num_batches = 0
    for data in loader:
        data = data.to(device)
        optimizer.zero_grad()
        loss = criterion(model(data.x, data.edge_index, data.batch), data.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    return total_loss

def evaluate(model, loader):
    model.eval()
    correct = 0
    for data in loader:
        data = data.to(device)
        pred = model(data.x, data.edge_index, data.batch).argmax(dim=1)
        correct += int((pred == data.y).sum())
    return correct / len(loader.dataset)

model = GNN(3, 64, 2).to(device)
optimizer = Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

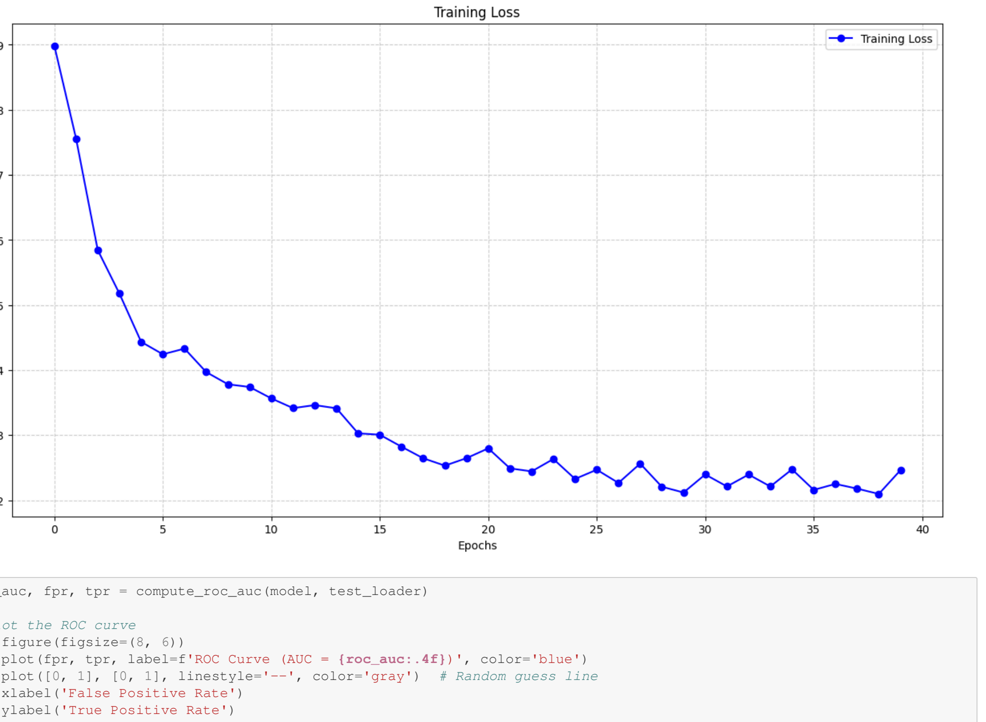
In [20]:
```python
def compute_roc_auc(model, loader):
    model.eval()
    y_true, y_scores = [], []
    for data in loader:
        data = data.to(device)
        outputs = model(data.x, data.edge_index, data.batch)
        probs = F.softmax(outputs, dim=1)[:, 1].cpu().detach().numpy()
        y_true.extend(data.y.cpu().numpy())
        y_scores.extend(probs)
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    return auc(fpr, tpr), fpr, tpr

train_losses = []
train_accuracies = []
test_accuracies = []
roc_auc_scores = []
best_test_acc = 0

for epoch in range(40):
    train_loss = train(model, train_loader, optimizer, criterion)
    train_acc = evaluate(model, train_loader)
    test_acc = evaluate(model, test_loader)
    roc_auc, fpr, tpr = compute_roc_auc(model, test_loader)

    train_losses.append(train_loss)
    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)
    roc_auc_scores.append(roc_auc)

    # Update best test accuracy
    if test_acc > best_test_acc:
        best_test_acc = test_acc

    print(f'Epoch {epoch}, Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}, Test Acc: {test_acc:.4f}, ROC AUC: {roc_auc:.4f}')

print(f'\nBest Test Accuracy: {best_test_acc:.4f}')
```

```
Epoch 0, Train Loss: 38.9843, Train Acc: 0.6038, Test Acc: 0.5970, ROC AUC: 0.7097
Epoch 1, Train Loss: 37.5590, Train Acc: 0.6646, Test Acc: 0.6625, ROC AUC: 0.7231
Epoch 2, Train Loss: 35.8449, Train Acc: 0.6829, Test Acc: 0.6760, ROC AUC: 0.7370
Epoch 3, Train Loss: 35.1756, Train Acc: 0.6914, Test Acc: 0.6880, ROC AUC: 0.7467
Epoch 4, Train Loss: 34.4349, Train Acc: 0.6939, Test Acc: 0.6895, ROC AUC: 0.7529
Epoch 5, Train Loss: 34.2439, Train Acc: 0.6924, Test Acc: 0.6880, ROC AUC: 0.7602
Epoch 6, Train Loss: 34.3309, Train Acc: 0.6915, Test Acc: 0.6915, ROC AUC: 0.7600
Epoch 7, Train Loss: 33.9701, Train Acc: 0.6963, Test Acc: 0.6960, ROC AUC: 0.7630
Epoch 8, Train Loss: 33.7831, Train Acc: 0.7014, Test Acc: 0.6995, ROC AUC: 0.7656
Epoch 9, Train Loss: 33.7409, Train Acc: 0.7013, Test Acc: 0.7035, ROC AUC: 0.7675
Epoch 10, Train Loss: 33.5658, Train Acc: 0.7024, Test Acc: 0.7060, ROC AUC: 0.7700
Epoch 11, Train Loss: 33.4167, Train Acc: 0.7054, Test Acc: 0.7065, ROC AUC: 0.7718
Epoch 12, Train Loss: 33.4624, Train Acc: 0.7053, Test Acc: 0.7090, ROC AUC: 0.7735
Epoch 13, Train Loss: 33.4143, Train Acc: 0.7051, Test Acc: 0.7135, ROC AUC: 0.7742
Epoch 14, Train Loss: 33.0292, Train Acc: 0.7037, Test Acc: 0.6995, ROC AUC: 0.7750
Epoch 15, Train Loss: 33.0074, Train Acc: 0.7087, Test Acc: 0.7010, ROC AUC: 0.7773
Epoch 16, Train Loss: 32.8237, Train Acc: 0.7126, Test Acc: 0.7075, ROC AUC: 0.7770
Epoch 17, Train Loss: 32.6460, Train Acc: 0.7135, Test Acc: 0.7165, ROC AUC: 0.7781
Epoch 18, Train Loss: 32.5327, Train Acc: 0.7119, Test Acc: 0.7060, ROC AUC: 0.7804
Epoch 19, Train Loss: 32.6482, Train Acc: 0.7129, Test Acc: 0.7130, ROC AUC: 0.7804
Epoch 20, Train Loss: 32.7982, Train Acc: 0.7167, Test Acc: 0.7135, ROC AUC: 0.7807
Epoch 21, Train Loss: 32.4910, Train Acc: 0.7171, Test Acc: 0.7105, ROC AUC: 0.7798
Epoch 22, Train Loss: 32.4442, Train Acc: 0.7165, Test Acc: 0.7145, ROC AUC: 0.7807
Epoch 23, Train Loss: 32.6344, Train Acc: 0.7179, Test Acc: 0.7185, ROC AUC: 0.7810
Epoch 24, Train Loss: 32.3283, Train Acc: 0.7149, Test Acc: 0.7100, ROC AUC: 0.7807
Epoch 25, Train Loss: 32.4701, Train Acc: 0.7163, Test Acc: 0.7110, ROC AUC: 0.7811
Epoch 26, Train Loss: 32.2670, Train Acc: 0.7142, Test Acc: 0.7120, ROC AUC: 0.7809
Epoch 27, Train Loss: 32.5638, Train Acc: 0.7154, Test Acc: 0.7090, ROC AUC: 0.7808
Epoch 28, Train Loss: 32.2082, Train Acc: 0.7156, Test Acc: 0.7125, ROC AUC: 0.7814
Epoch 29, Train Loss: 32.1198, Train Acc: 0.7160, Test Acc: 0.7115, ROC AUC: 0.7817
Epoch 30, Train Loss: 32.3968, Train Acc: 0.7136, Test Acc: 0.7145, ROC AUC: 0.7808
Epoch 31, Train Loss: 32.2128, Train Acc: 0.7164, Test Acc: 0.7115, ROC AUC: 0.7817
Epoch 32, Train Loss: 32.3957, Train Acc: 0.7158, Test Acc: 0.7115, ROC AUC: 0.7820
Epoch 33, Train Loss: 32.2124, Train Acc: 0.7169, Test Acc: 0.7100, ROC AUC: 0.7823
Epoch 34, Train Loss: 32.4755, Train Acc: 0.7193, Test Acc: 0.7130, ROC AUC: 0.7820
Epoch 35, Train Loss: 32.1599, Train Acc: 0.7185, Test Acc: 0.7090, ROC AUC: 0.7825
Epoch 36, Train Loss: 32.2516, Train Acc: 0.7157, Test Acc: 0.7110, ROC AUC: 0.7827
Epoch 37, Train Loss: 32.1794, Train Acc: 0.7163, Test Acc: 0.7115, ROC AUC: 0.7832
Epoch 38, Train Loss: 32.0975, Train Acc: 0.7194, Test Acc: 0.7130, ROC AUC: 0.7826
Epoch 39, Train Loss: 32.4635, Train Acc: 0.7178, Test Acc: 0.7100, ROC AUC: 0.7819

Best Test Accuracy: 0.7185
```

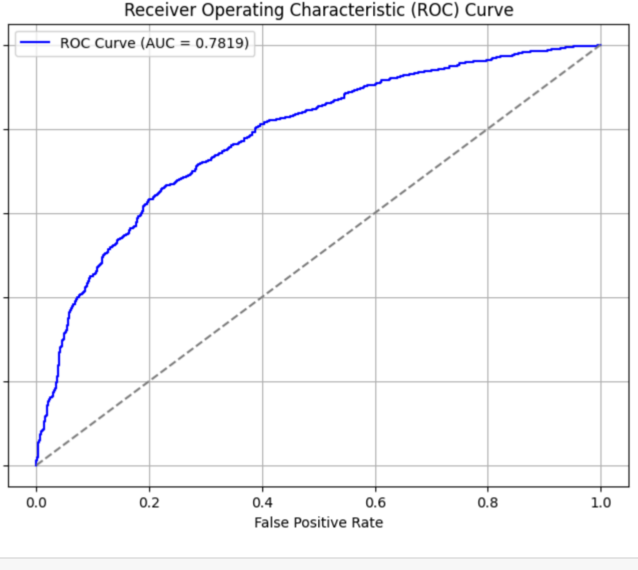
In [21]:
```python
plt.figure(figsize=(15, 8))
plt.plot(train_accuracies, marker='o', linestyle='-', label='Training Accuracy', color='violet')
plt.plot(test_accuracies, marker='x', linestyle='--', label='Validation Accuracy', color='blue')

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training vs Validation Accuracy")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```



In [22]:
```python
plt.figure(figsize=(15, 8))
plt.plot(train_losses, marker='o', linestyle='-', label='Training Loss', color='blue')

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```



In [23]:
```python
roc_auc, fpr, tpr = compute_roc_auc(model, test_loader)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.4f})', color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Random guess line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.grid()
plt.show()
```



In [ ]: