Specific_Task_01

quark/gluon Classification using Contrastive Loss

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 4.5 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 57.8 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:])
        X = np.array(f['X_jets'][:sample_size])
        y = np.array(f['y'][:sample_size], dtype=np.int64)
        metadata = np.column_stack((f['m0'][:sample_size], f['pt'][:sample_size]))
    return X, y, metadata

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

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

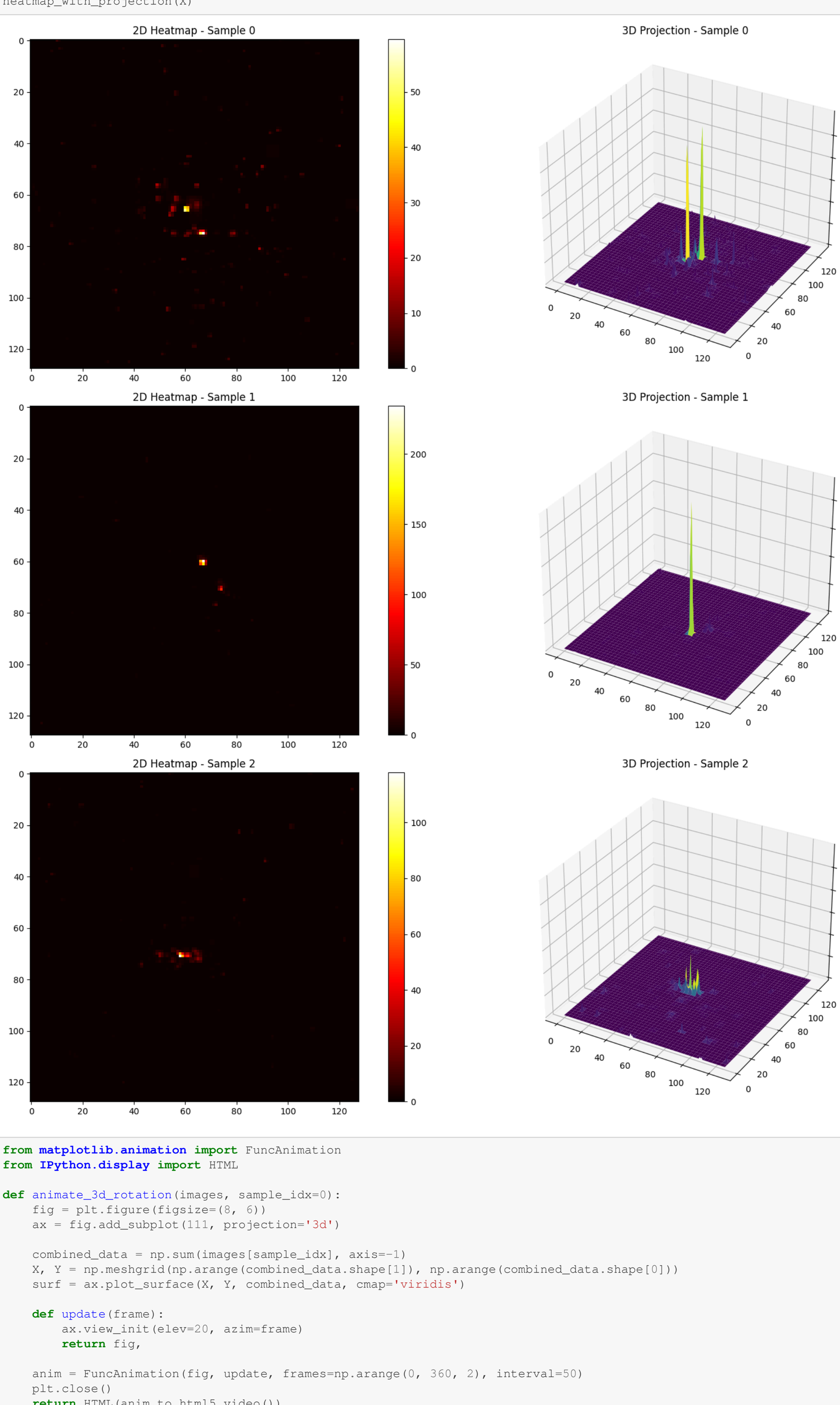
In [7]:
```python
def count_labels(labels):
    label_counts = np.bincount(labels.astype(np.int64))
    return {str(i): count for i, count in enumerate(label_counts)}

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)
```

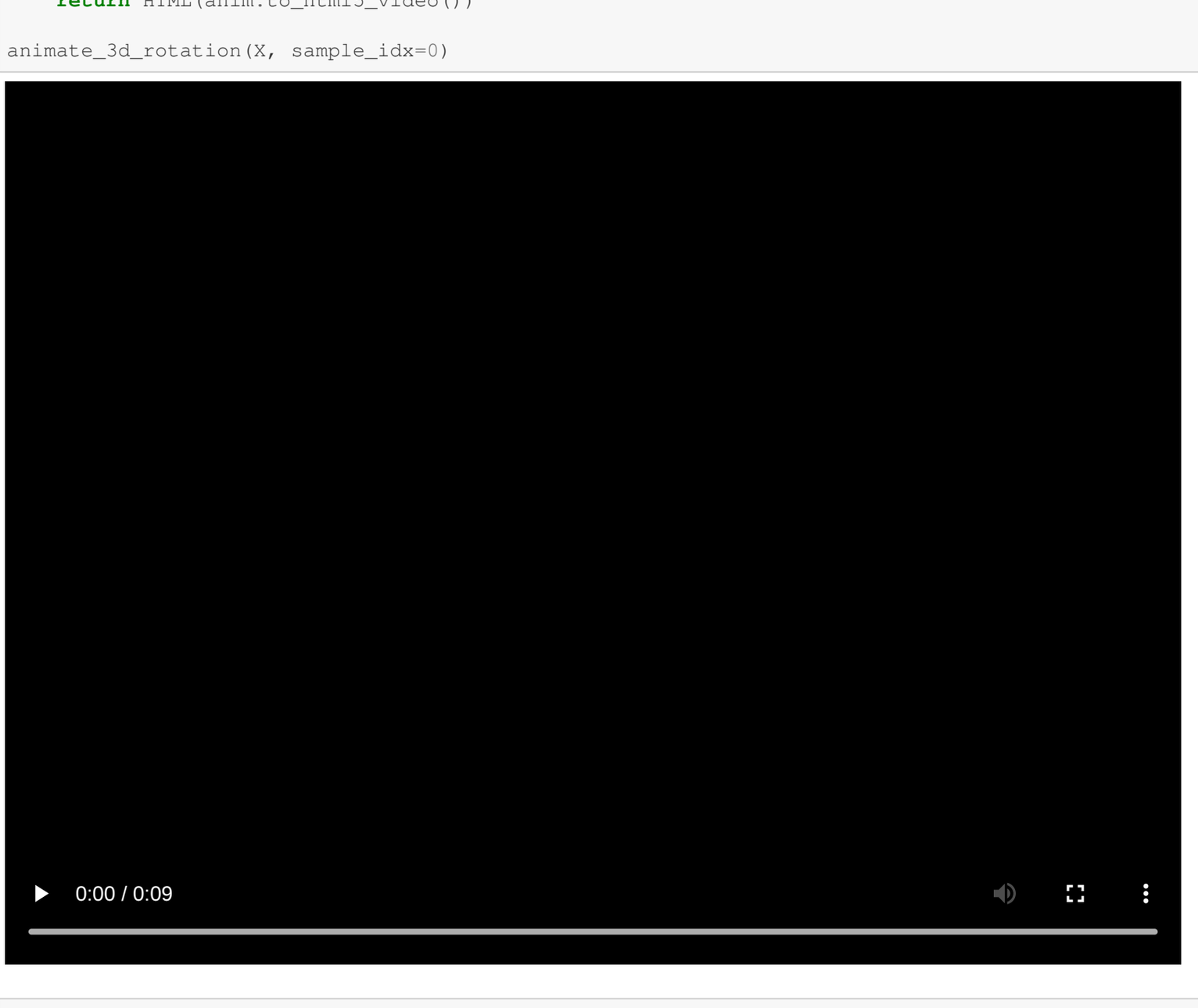
In [9]:
```python
def normalize_metadata(metadata):
    mean, std = np.mean(metadata, axis=0), np.std(metadata, axis=0)
    return (metadata - mean) / std

metadata = normalize_metadata(metadata)
```

In [10]:
```python
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 [11]:
```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[11]:



In [12]:
```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 [13]:
```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)
```

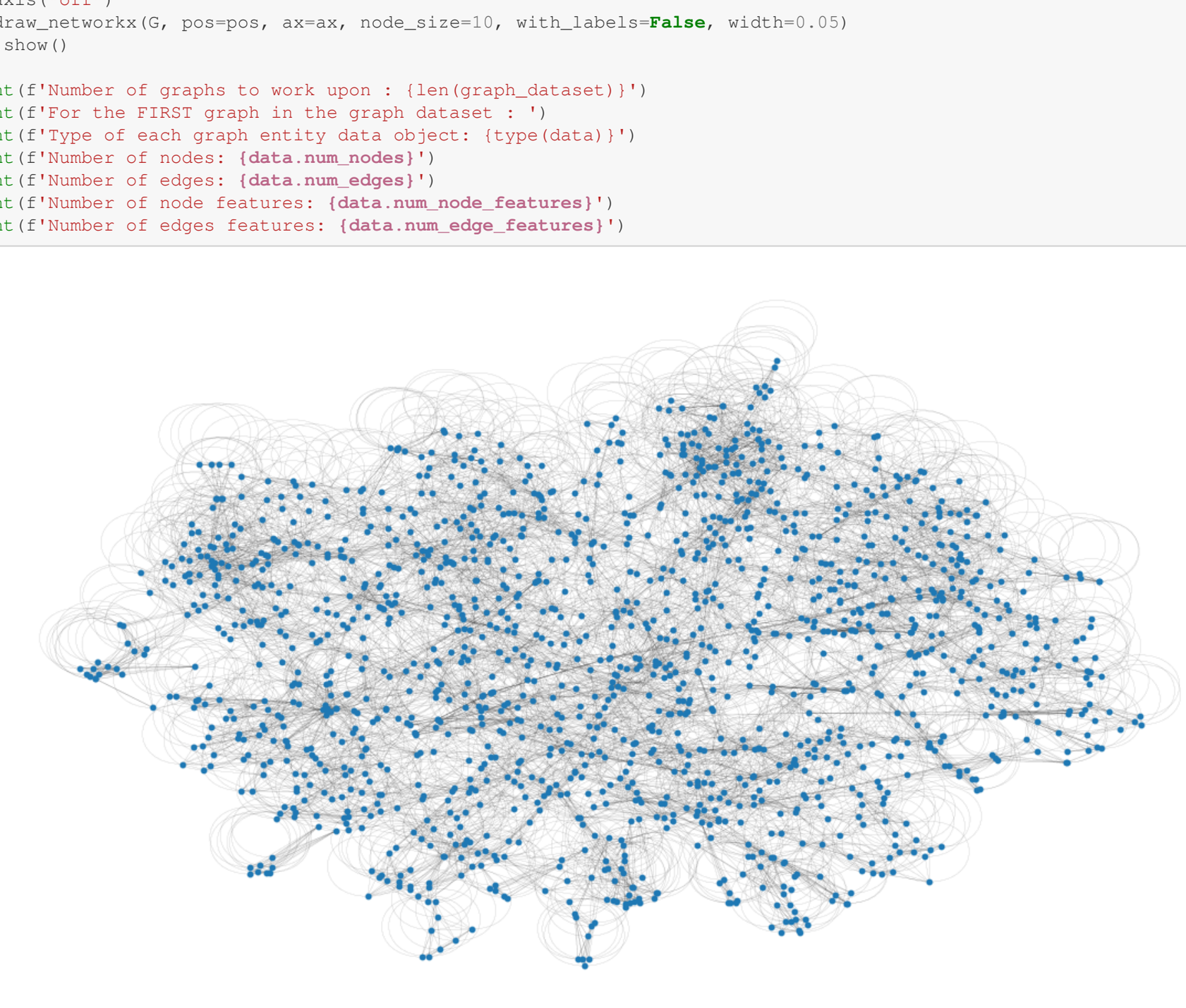
In [14]:
```python
def create_graph_features(masked_data):
    indices, features = [], []
    for img_idx, mask in enumerate(masked_data):
        coords = np.column_stack(np.where(mask))
        features.append(X[img_idx, coords[:, 0], coords[:, 1], :])
        indices.append(coords)
    return indices, features

indices_list, features_list = create_graph_features(mask)
```

In [15]:
```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 [16]:
```python
def create_graph_dataset(indices_list, features_list, metadata, 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)

        mass, momentum = metadata[i]
        mass_tensor = torch.full((features_list[i].shape[0], 1), mass, dtype=torch.float32)
        momentum_tensor = torch.full((features_list[i].shape[0], 1), momentum, dtype=torch.float32)

        node_features = torch.from_numpy(features_list[i]).float()
        node_features = torch.cat((node_features, mass_tensor, momentum_tensor), dim=1)

        label = torch.tensor([int(labels[i])], dtype=torch.long)
        graph = Data(x=node_features, edge_index=edge_idx, edge_attr=edge_weights, y=label)
        dataset.append(graph)

    return dataset
```

In [17]:
```python
graph_dataset = create_graph_dataset(indices_list, features_list, metadata, y, neighbors=8)
```

In [18]:
```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: 5
Number of edges features: 1
```

In [19]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import SAGEConv, global_mean_pool

class GraphEncoder(nn.Module):
    def __init__(self, point_dim=3, metadata_dim=2, hidden_dim=128, out_dim=64):
        super().__init__()
        self.point_encoder = nn.Sequential(
            nn.Linear(point_dim, hidden_dim // 2),
            nn.ReLU(),
        )

        self.metadata_encoder = nn.Sequential(
            nn.Linear(metadata_dim, hidden_dim // 4),
            nn.ReLU(),
            nn.Linear(hidden_dim // 4, hidden_dim // 2),
        )

        self.conv1 = SAGEConv(hidden_dim, hidden_dim)
        self.conv2 = SAGEConv(hidden_dim, hidden_dim)
        self.conv3 = SAGEConv(hidden_dim, out_dim)

    def forward(self, data):
        point_features = data.x[:, :3]
        metadata_features = data.x[:, 3:]

        point_emb = self.point_encoder(point_features)
        metadata_emb = self.metadata_encoder(metadata_features)

        x = torch.cat([point_emb, metadata_emb], dim=1)

        x = F.relu(self.conv1(x, data.edge_index))
        x = F.relu(self.conv2(x, data.edge_index))
        x = self.conv3(x, data.edge_index)


        return global_mean_pool(x, data.batch)
```

In [20]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self, encoder, projection_dim=32):
        super().__init__()
        self.encoder = encoder

        self.projector = nn.Sequential(
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, projection_dim),
        )

    def forward(self, graph):
        embedding = self.encoder(graph)
        projected = self.projector(embedding)
        return F.normalize(projected, dim=1)
```

In [21]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Loss(nn.Module):
    def __init__(self, temperature=0.1, hard_neg_weight=0.5):
        super().__init__()
        self.temp = temperature
        self.hard_neg_weight = hard_neg_weight

    def forward(self, features, labels):
        features = F.normalize(features, dim=1)

        sim_matrix = torch.matmul(features, features.T) / self.temp

        mask = torch.eq(labels.unsqueeze(1), labels.unsqueeze(0)).float()

        exp_sim = torch.exp(sim_matrix)

        hard_neg_mask = (1 - mask) * (sim_matrix > 0.5).float()

        log_prob = sim_matrix - torch.log(exp_sim.sum(dim=1, keepdim=True))

        weighted_log_prob = log_prob * (mask + self.hard_neg_weight * hard_neg_mask)

        mean_log_prob = weighted_log_prob.sum(1) / (mask.sum(1) + self.hard_neg_weight * hard_neg_mask.sum(1))
        return -mean_log_prob.mean()
```

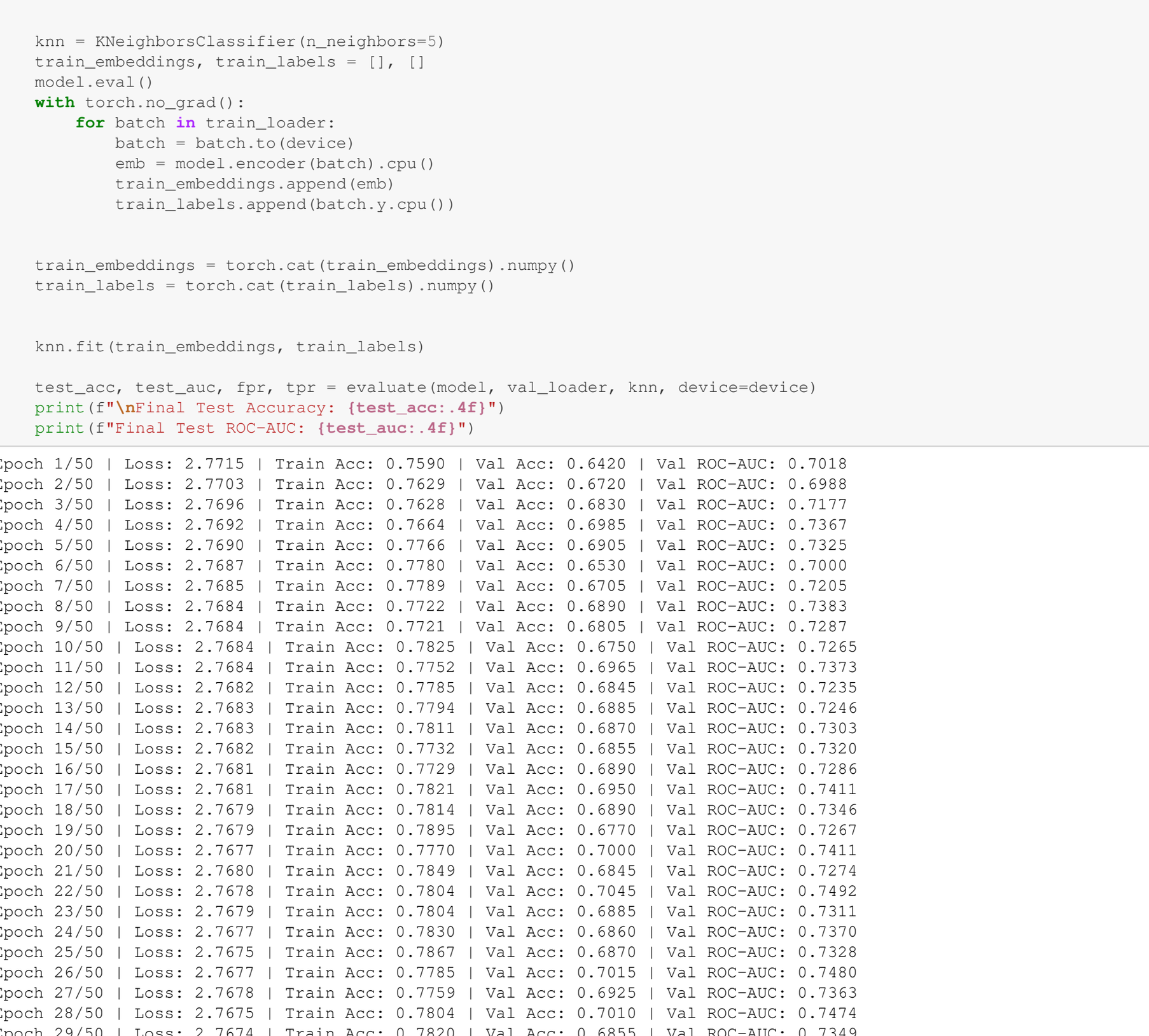
In [22]:
```python
import torch
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

def train(model, train_loader, val_loader, optimizer, criterion,schedular, num_epochs=50, device="cuda"):
    model.to(device)

    # Initialize KNN once
    knn = KNeighborsClassifier(n_neighbors=5)

    train_losses = []
    train_accuracies = []
    val_accuracies = []
    best_val_acc = 0.0

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        train_embeddings = []
        train_labels = []

        # Training phase
        for batch in train_loader:
            batch = batch.to(device)
            optimizer.zero_grad()

            projections = model(batch)
            loss = criterion(projections, batch.y.view(-1))
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

            with torch.no_grad():
                emb = model.encoder(batch).cpu()
                train_embeddings.append(emb)
                train_labels.append(batch.y.cpu())

        scheduler.step()
        current_lr = scheduler.get_last_lr()[0]


        train_embeddings = torch.cat(train_embeddings).numpy()
        train_labels = torch.cat(train_labels).numpy()

        knn.fit(train_embeddings, train_labels)
        train_acc = knn.score(train_embeddings, train_labels)

        val_acc, val_auc, _, _= evaluate(model, val_loader, knn) if val_loader else 0.0

        train_losses.append(total_loss / len(train_loader))
        train_accuracies.append(train_acc)
        val_accuracies.append(val_acc)

        if val_acc > best_val_acc:
            best_val_acc = val_acc

        print(f"Epoch {epoch+1}/{num_epochs} | "
              f"Loss: {total_loss / len(train_loader):.4f} | "
              f"Train Acc: {train_acc:.4f} | "
              f"Val Acc: {val_acc:.4f} | "
              f"Val ROC-AUC: {val_auc:.4f}")

    # Print best validation accuracy
    print(f"\n🔥 Best Validation Accuracy: {best_val_acc:.4f}")

    return train_losses, train_accuracies, val_accuracies
```

In [23]:
```python
import torch
import numpy as np

def evaluate(model, loader, knn, device="cuda"):
    model.eval()
    embeddings = []
    labels = []

    with torch.no_grad():
        for batch in loader:
            batch = batch.to(device)
            emb = model.encoder(batch).cpu()
            embeddings.append(emb)
            labels.append(batch.y.view(-1).cpu())

    if not embeddings:
        return 0.0

    embeddings = torch.cat(embeddings).numpy()
    labels = torch.cat(labels).numpy()

    val_acc = knn.score(embeddings, labels)

    probs = knn.predict_proba(embeddings)[:, 1]
    val_auc = roc_auc_score(labels, probs) if len(np.unique(labels)) > 1 else 0.0

    fpr, tpr, _ = roc_curve(labels, probs) if len(np.unique(labels)) > 1 else ([], [], [])

    return val_acc, val_auc, fpr, tpr
```

In [24]:
```python
import torch
from torch.utils.data import random_split
from torch_geometric.loader import DataLoader
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score, roc_curve

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

if __name__ == "__main__":
    train_size = int(0.8 * len(graph_dataset))
    val_size = len(graph_dataset) - train_size
    train_set, val_set = random_split(graph_dataset, [train_size, val_size], generator=torch.Generator().manual_seed(42))

    train_loader = DataLoader(train_set, batch_size=16, shuffle=True)
    val_loader = DataLoader(val_set, batch_size=16)

    encoder = GraphEncoder(point_dim=3, metadata_dim=2).to(device)
    model = Model(encoder).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=50, eta_min=1e-5)
    criterion = Loss()


    train_losses, train_accuracies, val_accuracies= train(
        model, train_loader, val_loader, optimizer, criterion, scheduler, num_epochs=50, device=device
    )


    knn = KNeighborsClassifier(n_neighbors=5)
    train_embeddings, train_labels = [], []
    model.eval()
    with torch.no_grad():
        for batch in train_loader:
            batch = batch.to(device)
            emb = model.encoder(batch).cpu()
            train_embeddings.append(emb)
            train_labels.append(batch.y.cpu())


    train_embeddings = torch.cat(train_embeddings).numpy()
    train_labels = torch.cat(train_labels).numpy()


    knn.fit(train_embeddings, train_labels)

    test_acc, test_auc, fpr, tpr = evaluate(model, val_loader, knn, device=device)
    print(f"\nFinal Test Accuracy: {test_acc:.4f}")
    print(f"Final Test ROC-AUC: {test_auc:.4f}")
```

```
Epoch 1/50 | Loss: 2.7715 | Train Acc: 0.7590 | Val Acc: 0.6420 | Val ROC-AUC: 0.7018
Epoch 2/50 | Loss: 2.7703 | Train Acc: 0.7629 | Val Acc: 0.6720 | Val ROC-AUC: 0.6988
Epoch 3/50 | Loss: 2.7696 | Train Acc: 0.7628 | Val Acc: 0.6830 | Val ROC-AUC: 0.7177
Epoch 4/50 | Loss: 2.7692 | Train Acc: 0.7664 | Val Acc: 0.6985 | Val ROC-AUC: 0.7367
Epoch 5/50 | Loss: 2.7690 | Train Acc: 0.7766 | Val Acc: 0.6905 | Val ROC-AUC: 0.7325
Epoch 6/50 | Loss: 2.7687 | Train Acc: 0.7780 | Val Acc: 0.6530 | Val ROC-AUC: 0.7000
Epoch 7/50 | Loss: 2.7685 | Train Acc: 0.7789 | Val Acc: 0.6705 | Val ROC-AUC: 0.7205
Epoch 8/50 | Loss: 2.7684 | Train Acc: 0.7722 | Val Acc: 0.6890 | Val ROC-AUC: 0.7383
Epoch 9/50 | Loss: 2.7684 | Train Acc: 0.7721 | Val Acc: 0.6805 | Val ROC-AUC: 0.7287
Epoch 10/50 | Loss: 2.7684 | Train Acc: 0.7825 | Val Acc: 0.6750 | Val ROC-AUC: 0.7265
Epoch 11/50 | Loss: 2.7684 | Train Acc: 0.7752 | Val Acc: 0.6965 | Val ROC-AUC: 0.7373
Epoch 12/50 | Loss: 2.7682 | Train Acc: 0.7785 | Val Acc: 0.6845 | Val ROC-AUC: 0.7235
Epoch 13/50 | Loss: 2.7683 | Train Acc: 0.7794 | Val Acc: 0.6885 | Val ROC-AUC: 0.7246
Epoch 14/50 | Loss: 2.7683 | Train Acc: 0.7811 | Val Acc: 0.6870 | Val ROC-AUC: 0.7303
Epoch 15/50 | Loss: 2.7682 | Train Acc: 0.7732 | Val Acc: 0.6855 | Val ROC-AUC: 0.7320
Epoch 16/50 | Loss: 2.7681 | Train Acc: 0.7729 | Val Acc: 0.6890 | Val ROC-AUC: 0.7286
Epoch 17/50 | Loss: 2.7681 | Train Acc: 0.7821 | Val Acc: 0.6950 | Val ROC-AUC: 0.7411
Epoch 18/50 | Loss: 2.7679 | Train Acc: 0.7814 | Val Acc: 0.6890 | Val ROC-AUC: 0.7346
Epoch 19/50 | Loss: 2.7679 | Train Acc: 0.7895 | Val Acc: 0.6770 | Val ROC-AUC: 0.7267
Epoch 20/50 | Loss: 2.7677 | Train Acc: 0.7770 | Val Acc: 0.7000 | Val ROC-AUC: 0.7411
Epoch 21/50 | Loss: 2.7680 | Train Acc: 0.7849 | Val Acc: 0.6845 | Val ROC-AUC: 0.7274
Epoch 22/50 | Loss: 2.7678 | Train Acc: 0.7804 | Val Acc: 0.7045 | Val ROC-AUC: 0.7492
Epoch 23/50 | Loss: 2.7679 | Train Acc: 0.7804 | Val Acc: 0.6885 | Val ROC-AUC: 0.7311
Epoch 24/50 | Loss: 2.7677 | Train Acc: 0.7830 | Val Acc: 0.6860 | Val ROC-AUC: 0.7370
Epoch 25/50 | Loss: 2.7675 | Train Acc: 0.7867 | Val Acc: 0.6870 | Val ROC-AUC: 0.7328
Epoch 26/50 | Loss: 2.7677 | Train Acc: 0.7785 | Val Acc: 0.7015 | Val ROC-AUC: 0.7480
Epoch 27/50 | Loss: 2.7678 | Train Acc: 0.7759 | Val Acc: 0.6925 | Val ROC-AUC: 0.7363
Epoch 28/50 | Loss: 2.7675 | Train Acc: 0.7804 | Val Acc: 0.7010 | Val ROC-AUC: 0.7474
Epoch 29/50 | Loss: 2.7674 | Train Acc: 0.7820 | Val Acc: 0.6855 | Val ROC-AUC: 0.7349
Epoch 30/50 | Loss: 2.7676 | Train Acc: 0.7929 | Val Acc: 0.6900 | Val ROC-AUC: 0.7318
Epoch 31/50 | Loss: 2.7675 | Train Acc: 0.7849 | Val Acc: 0.6960 | Val ROC-AUC: 0.7465
Epoch 32/50 | Loss: 2.7673 | Train Acc: 0.7816 | Val Acc: 0.6900 | Val ROC-AUC: 0.7369
Epoch 33/50 | Loss: 2.7675 | Train Acc: 0.7809 | Val Acc: 0.6790 | Val ROC-AUC: 0.7252
Epoch 34/50 | Loss: 2.7673 | Train Acc: 0.7802 | Val Acc: 0.6960 | Val ROC-AUC: 0.7336
Epoch 35/50 | Loss: 2.7671 | Train Acc: 0.7785 | Val Acc: 0.6895 | Val ROC-AUC: 0.7390
Epoch 36/50 | Loss: 2.7673 | Train Acc: 0.7856 | Val Acc: 0.6880 | Val ROC-AUC: 0.7301
Epoch 37/50 | Loss: 2.7674 | Train Acc: 0.7885 | Val Acc: 0.7000 | Val ROC-AUC: 0.7384
Epoch 38/50 | Loss: 2.7672 | Train Acc: 0.7843 | Val Acc: 0.6940 | Val ROC-AUC: 0.7502
Epoch 39/50 | Loss: 2.7671 | Train Acc: 0.7874 | Val Acc: 0.6905 | Val ROC-AUC: 0.7389
Epoch 40/50 | Loss: 2.7671 | Train Acc: 0.7919 | Val Acc: 0.6855 | Val ROC-AUC: 0.7333
Epoch 41/50 | Loss: 2.7673 | Train Acc: 0.7814 | Val Acc: 0.6910 | Val ROC-AUC: 0.7368
Epoch 42/50 | Loss: 2.7672 | Train Acc: 0.7896 | Val Acc: 0.6990 | Val ROC-AUC: 0.7395
Epoch 43/50 | Loss: 2.7671 | Train Acc: 0.7869 | Val Acc: 0.6965 | Val ROC-AUC: 0.7414
Epoch 44/50 | Loss: 2.7671 | Train Acc: 0.7893 | Val Acc: 0.6930 | Val ROC-AUC: 0.7505
Epoch 45/50 | Loss: 2.7670 | Train Acc: 0.7876 | Val Acc: 0.6940 | Val ROC-AUC: 0.7364
Epoch 46/50 | Loss: 2.7669 | Train Acc: 0.7866 | Val Acc: 0.6975 | Val ROC-AUC: 0.7472
Epoch 47/50 | Loss: 2.7669 | Train Acc: 0.7887 | Val Acc: 0.7000 | Val ROC-AUC: 0.7390
Epoch 48/50 | Loss: 2.7669 | Train Acc: 0.7849 | Val Acc: 0.6940 | Val ROC-AUC: 0.7349
Epoch 49/50 | Loss: 2.7668 | Train Acc: 0.7874 | Val Acc: 0.7015 | Val ROC-AUC: 0.7419
Epoch 50/50 | Loss: 2.7669 | Train Acc: 0.7897 | Val Acc: 0.6985 | Val ROC-AUC: 0.7377

🔥 Best Validation Accuracy: 0.7045

Final Test Accuracy: 0.6980
Final Test ROC-AUC: 0.7326
```

In [25]:
```python
plt.figure(figsize=(15, 8))
plt.plot(train_accuracies, marker='o', linestyle='-', label='Training Accuracy', color='violet')
plt.plot(val_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 [26]:
```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 [27]:
```python
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {test_auc:.4f})', color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.grid()
plt.show()
```