```
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geom
         etric) (1.18.3)
         Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch-geometr
         ic) (3.0.2)
         Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (1.3.
         Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric)
         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.
         Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (2022.
         Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric)
         Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->to
         rch-geometric) (3.4.1)
         Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometri
         Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torch-ge
         ometric) (2.3.0)
         Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torch-ge
         ometric) (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-geometr
         ic) (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-op
         enmp>=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 30.1 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]: 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]: | 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]: print(device)
         cuda
 In [5]: 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]: 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]: 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]: 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]: 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)
                                                                                               3D Projection - Sample 0
                           2D Heatmap - Sample 0
                                                                  50
                                                                  40
                                                                                                                             30
                                                                  30
                                                                                                                             20
                                                                 - 20
                                                                                                                        100
          100 -
                                                                                                                       80
                                                                                       20
                                                                                                                     60
                                                                  10
                                                                                           40
                                                                                              60
                                                                                                 80
                                                                                                     100
          120 -
                                               100
                                                      120
                           2D Heatmap - Sample 1
                                                                                               3D Projection - Sample 1
                                                                  200
                                                                                                                             200
                                                                  150
                                                                                                                             150
                                                                                                                             100
                                                                                                                             50
                                                                                                                          120
                                                                                                                        100
          100 -
                                                                  50
                                                                                                                       80
                                                                                          40
                                                                                              60
                                                                                                 80
                                                                                                     100
          120 -
                                                                                                         120
                                                       120
                           2D Heatmap - Sample 2
                                                                                               3D Projection - Sample 2
                                                                  100
                                                                  80
                                                                                                                             60
                                                                  60
                                                                                                                             40
                                                                                                                             20
                                                                                                                        100
                                                                                                                       80
          100 -
                                                                 - 20
                                                                                          40
                                                                                              60
                                                                                                  80
          120 -
                                                                                                     100
             0
                                                100
                   20
                           40
                                  60
                                         80
                                                       120
In [10]: 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]:
           0:00 / 0:09
In [11]: 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]: 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]: 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]: 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]: graph_dataset = create_graph_dataset(indices_list, y, neighbors=8)
In [16]: 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]: | 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]: from torch_geometric.nn import SAGEConv,global_mean_pool
         class GNN(torch.nn.Module):
             def __init__(self, in_features, hidden_dim, num_classes):
                 super().__init__()
                 self.layers = nn.ModuleList([
                     SAGEConv(in_features, hidden_dim),
                      SAGEConv(hidden_dim, 2 * hidden_dim),
                     SAGEConv(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]: 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]: 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: 37.6869, Train Acc: 0.6500, Test Acc: 0.6415, ROC AUC: 0.7268
         Epoch 1, Train Loss: 35.5801, Train Acc: 0.6754, Test Acc: 0.6860, ROC AUC: 0.7422
         Epoch 2, Train Loss: 34.7762, Train Acc: 0.6797, Test Acc: 0.6790, ROC AUC: 0.7540
         Epoch 3, Train Loss: 34.3511, Train Acc: 0.6917, Test Acc: 0.6875, ROC AUC: 0.7607
         Epoch 4, Train Loss: 33.8922, Train Acc: 0.6926, Test Acc: 0.6935, ROC AUC: 0.7685
         Epoch 5, Train Loss: 33.6760, Train Acc: 0.6953, Test Acc: 0.6945, ROC AUC: 0.7712
         Epoch 6, Train Loss: 33.6043, Train Acc: 0.7003, Test Acc: 0.7050, ROC AUC: 0.7723
         Epoch 7, Train Loss: 33.0549, Train Acc: 0.7029, Test Acc: 0.7060, ROC AUC: 0.7740
         Epoch 8, Train Loss: 33.1737, Train Acc: 0.7011, Test Acc: 0.7030, ROC AUC: 0.7767
         Epoch 9, Train Loss: 33.2388, Train Acc: 0.7197, Test Acc: 0.7115, ROC AUC: 0.7786
         Epoch 10, Train Loss: 32.5883, Train Acc: 0.6979, Test Acc: 0.6950, ROC AUC: 0.7790
         Epoch 11, Train Loss: 34.0718, Train Acc: 0.7029, Test Acc: 0.7055, ROC AUC: 0.7757
         Epoch 12, Train Loss: 32.8650, Train Acc: 0.7222, Test Acc: 0.7135, ROC AUC: 0.7802
         Epoch 13, Train Loss: 32.5983, Train Acc: 0.7151, Test Acc: 0.7150, ROC AUC: 0.7798
         Epoch 14, Train Loss: 32.9276, Train Acc: 0.6882, Test Acc: 0.6835, ROC AUC: 0.7815
         Epoch 15, Train Loss: 32.4835, Train Acc: 0.7074, Test Acc: 0.7065, ROC AUC: 0.7814
         Epoch 16, Train Loss: 32.3610, Train Acc: 0.7151, Test Acc: 0.7105, ROC AUC: 0.7813
         Epoch 17, Train Loss: 32.6768, Train Acc: 0.7071, Test Acc: 0.7075, ROC AUC: 0.7812
         Epoch 18, Train Loss: 32.4731, Train Acc: 0.7197, Test Acc: 0.7115, ROC AUC: 0.7811
         Epoch 19, Train Loss: 32.4254, Train Acc: 0.7185, Test Acc: 0.7055, ROC AUC: 0.7826
         Epoch 20, Train Loss: 32.2632, Train Acc: 0.7139, Test Acc: 0.7110, ROC AUC: 0.7832
         Epoch 21, Train Loss: 32.4083, Train Acc: 0.7151, Test Acc: 0.7120, ROC AUC: 0.7833
         Epoch 22, Train Loss: 32.6549, Train Acc: 0.7211, Test Acc: 0.7205, ROC AUC: 0.7831
         Epoch 23, Train Loss: 32.8262, Train Acc: 0.7086, Test Acc: 0.7045, ROC AUC: 0.7829
         Epoch 24, Train Loss: 32.6085, Train Acc: 0.7010, Test Acc: 0.7000, ROC AUC: 0.7831
         Epoch 25, Train Loss: 32.4665, Train Acc: 0.7231, Test Acc: 0.7160, ROC AUC: 0.7836
         Epoch 26, Train Loss: 32.4097, Train Acc: 0.7210, Test Acc: 0.7115, ROC AUC: 0.7838
         Epoch 27, Train Loss: 32.1999, Train Acc: 0.7219, Test Acc: 0.7110, ROC AUC: 0.7831
         Epoch 28, Train Loss: 32.1292, Train Acc: 0.7186, Test Acc: 0.7105, ROC AUC: 0.7831
         Epoch 29, Train Loss: 32.2881, Train Acc: 0.7004, Test Acc: 0.6980, ROC AUC: 0.7849
         Epoch 30, Train Loss: 32.3895, Train Acc: 0.7071, Test Acc: 0.7005, ROC AUC: 0.7840
         Epoch 31, Train Loss: 32.2274, Train Acc: 0.7211, Test Acc: 0.7160, ROC AUC: 0.7837
         Epoch 32, Train Loss: 31.9620, Train Acc: 0.7231, Test Acc: 0.7150, ROC AUC: 0.7842
         Epoch 33, Train Loss: 32.2410, Train Acc: 0.6958, Test Acc: 0.6955, ROC AUC: 0.7839
         Epoch 34, Train Loss: 32.0987, Train Acc: 0.7157, Test Acc: 0.7135, ROC AUC: 0.7847
         Epoch 35, Train Loss: 32.3820, Train Acc: 0.7190, Test Acc: 0.7135, ROC AUC: 0.7852
         Epoch 36, Train Loss: 31.9313, Train Acc: 0.7212, Test Acc: 0.7125, ROC AUC: 0.7855
         Epoch 37, Train Loss: 32.0348, Train Acc: 0.7222, Test Acc: 0.7100, ROC AUC: 0.7841
         Epoch 38, Train Loss: 31.9112, Train Acc: 0.7221, Test Acc: 0.7190, ROC AUC: 0.7851
         Epoch 39, Train Loss: 32.0577, Train Acc: 0.7250, Test Acc: 0.7150, ROC AUC: 0.7840
         Best Test Accuracy: 0.7205
In [21]: 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()
                                                            Training vs Validation Accuracy
                  Training Accuracy
                 -X- Validation Accuracy
            0.72
            0.70
            0.66
            0.64
                                                                        20
                                                                                     25
                                                                                                               35
                                                                     Epochs
In [22]: 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()
                                                                  Training Loss
                                                                                                                  Training Loss
            37
            36
            34
            33
            32
                                             10
                                                                       20
                                                                     Epochs
In [23]: 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='red')
         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()
                               Receiver Operating Characteristic (ROC) Curve
                       ROC Curve (AUC = 0.7840)
             0.8
          Frue Positive Rate
             0.6
             0.2
             0.0
                                              0.4
                   0.0
                                0.2
                                                            0.6
                                                                         0.8
                                                                                       1.0
                                              False Positive Rate
```

Common_Task_02

Collecting torch-geometric

In [1]: pip install torch-geometric

h-geometric) (2.4.6)

h-geometric) (5.0.1)

tric) (1.3.2)

c) (25.1.0)

etric) (1.5.0)

ometric) (6.1.0)

quark/gluon Classification

Downloading torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)

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.

Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torc

Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geome

Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torc

Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometri

Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geom

Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-ge

Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geome

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)

- 63.1/63.1 kB 2.5 MB/s eta 0:00:00