mean, std = np.mean(processed), np.std(processed) return np.clip((processed - mean) / std, 0, None) X = preprocess_images(X) In [8]: 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 [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) 2D Heatmap - Sample 0 3D Projection - Sample 0 50 40 30 30 20 - 20 20 40 10 60 120 -100 0 120 2D Heatmap - Sample 1 3D Projection - Sample 1 200 200 150 150 100 100 100 100 -20 40 60 80 120 -120 3D Projection - Sample 2 2D Heatmap - Sample 2 100 80 60 60 40 20 100 20 60 120 -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) 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])**2weights = 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, 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) In [15]: graph_dataset = create_graph_dataset(indices_list, features_list, metadata, 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: 5 Number of edges features: 1 In [17]: import torch import torch.nn as nn import torch.nn.functional as F from torch_geometric.nn import GATConv, 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 = GATConv(hidden_dim, hidden_dim) self.conv2 = GATConv(hidden_dim, hidden_dim) self.conv3 = GATConv(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 [18]: 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 [19]: 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 [20]: 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 [21]: 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 [22]: 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(4) 2)) 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: 2.7707 | Train Acc: 0.7542 | Val Acc: 0.6430 | Val ROC-AUC: 0.6625 Epoch 2/50 | Loss: 2.7701 | Train Acc: 0.7599 | Val Acc: 0.6635 | Val ROC-AUC: 0.7037 Epoch 3/50 | Loss: 2.7700 | Train Acc: 0.7569 | Val Acc: 0.6500 | Val ROC-AUC: 0.6932 Epoch 4/50 | Loss: 2.7697 | Train Acc: 0.7611 | Val Acc: 0.6345 | Val ROC-AUC: 0.6727 Epoch 5/50 | Loss: 2.7697 | Train Acc: 0.7635 | Val Acc: 0.6935 | Val ROC-AUC: 0.7307 Epoch 6/50 | Loss: 2.7695 | Train Acc: 0.7678 | Val Acc: 0.6425 | Val ROC-AUC: 0.6872 Epoch 7/50 | Loss: 2.7695 | Train Acc: 0.7686 | Val Acc: 0.6520 | Val ROC-AUC: 0.7057 Epoch 8/50 | Loss: 2.7693 | Train Acc: 0.7639 | Val Acc: 0.6690 | Val ROC-AUC: 0.7086 Epoch 9/50 | Loss: 2.7691 | Train Acc: 0.7671 | Val Acc: 0.6390 | Val ROC-AUC: 0.6835 Epoch 10/50 | Loss: 2.7690 | Train Acc: 0.7719 | Val Acc: 0.6750 | Val ROC-AUC: 0.7157 Epoch 11/50 | Loss: 2.7689 | Train Acc: 0.7752 | Val Acc: 0.6835 | Val ROC-AUC: 0.7248 Epoch 12/50 | Loss: 2.7688 | Train Acc: 0.7684 | Val Acc: 0.6580 | Val ROC-AUC: 0.7029 Epoch 13/50 | Loss: 2.7687 | Train Acc: 0.7672 | Val Acc: 0.6540 | Val ROC-AUC: 0.6969 Epoch 14/50 | Loss: 2.7686 | Train Acc: 0.7731 | Val Acc: 0.6705 | Val ROC-AUC: 0.7132 Epoch 15/50 | Loss: 2.7685 | Train Acc: 0.7679 | Val Acc: 0.6840 | Val ROC-AUC: 0.7249 Epoch 16/50 | Loss: 2.7685 | Train Acc: 0.7705 | Val Acc: 0.6650 | Val ROC-AUC: 0.7197 Epoch 17/50 | Loss: 2.7684 | Train Acc: 0.7731 | Val Acc: 0.6795 | Val ROC-AUC: 0.7109 Epoch 18/50 | Loss: 2.7686 | Train Acc: 0.7752 | Val Acc: 0.6525 | Val ROC-AUC: 0.6978 Epoch 19/50 | Loss: 2.7683 | Train Acc: 0.7744 | Val Acc: 0.6720 | Val ROC-AUC: 0.7072 Epoch 20/50 | Loss: 2.7683 | Train Acc: 0.7651 | Val Acc: 0.6640 | Val ROC-AUC: 0.7001 Epoch 21/50 | Loss: 2.7683 | Train Acc: 0.7750 | Val Acc: 0.6625 | Val ROC-AUC: 0.7099 Epoch 22/50 | Loss: 2.7683 | Train Acc: 0.7778 | Val Acc: 0.6790 | Val ROC-AUC: 0.7273 Epoch 23/50 | Loss: 2.7681 | Train Acc: 0.7680 | Val Acc: 0.6765 | Val ROC-AUC: 0.7136 Epoch 24/50 | Loss: 2.7679 | Train Acc: 0.7694 | Val Acc: 0.6930 | Val ROC-AUC: 0.7314 Epoch 25/50 | Loss: 2.7680 | Train Acc: 0.7748 | Val Acc: 0.6730 | Val ROC-AUC: 0.7153 Epoch 26/50 | Loss: 2.7679 | Train Acc: 0.7715 | Val Acc: 0.6885 | Val ROC-AUC: 0.7378 Epoch 27/50 | Loss: 2.7681 | Train Acc: 0.7702 | Val Acc: 0.6800 | Val ROC-AUC: 0.7210 Epoch 28/50 | Loss: 2.7680 | Train Acc: 0.7771 | Val Acc: 0.6755 | Val ROC-AUC: 0.7220 Epoch 29/50 | Loss: 2.7678 | Train Acc: 0.7749 | Val Acc: 0.6795 | Val ROC-AUC: 0.7189 Epoch 30/50 | Loss: 2.7675 | Train Acc: 0.7732 | Val Acc: 0.6775 | Val ROC-AUC: 0.7138 Epoch 31/50 | Loss: 2.7679 | Train Acc: 0.7774 | Val Acc: 0.6760 | Val ROC-AUC: 0.7161 Epoch 32/50 | Loss: 2.7677 | Train Acc: 0.7735 | Val Acc: 0.6835 | Val ROC-AUC: 0.7177 Epoch 33/50 | Loss: 2.7674 | Train Acc: 0.7789 | Val Acc: 0.6615 | Val ROC-AUC: 0.7106 Epoch 34/50 | Loss: 2.7674 | Train Acc: 0.7778 | Val Acc: 0.6930 | Val ROC-AUC: 0.7374 Epoch 35/50 | Loss: 2.7676 | Train Acc: 0.7751 | Val Acc: 0.6725 | Val ROC-AUC: 0.7203 Epoch 36/50 | Loss: 2.7672 | Train Acc: 0.7804 | Val Acc: 0.6740 | Val ROC-AUC: 0.7196 Epoch 37/50 | Loss: 2.7673 | Train Acc: 0.7799 | Val Acc: 0.6850 | Val ROC-AUC: 0.7312 Epoch 38/50 | Loss: 2.7672 | Train Acc: 0.7785 | Val Acc: 0.6820 | Val ROC-AUC: 0.7244 Epoch 39/50 | Loss: 2.7672 | Train Acc: 0.7768 | Val Acc: 0.6905 | Val ROC-AUC: 0.7276 Epoch 40/50 | Loss: 2.7672 | Train Acc: 0.7779 | Val Acc: 0.6970 | Val ROC-AUC: 0.7378 Epoch 41/50 | Loss: 2.7669 | Train Acc: 0.7804 | Val Acc: 0.6755 | Val ROC-AUC: 0.7230 Epoch 42/50 | Loss: 2.7669 | Train Acc: 0.7883 | Val Acc: 0.6815 | Val ROC-AUC: 0.7231 Epoch 43/50 | Loss: 2.7669 | Train Acc: 0.7823 | Val Acc: 0.6750 | Val ROC-AUC: 0.7251 Epoch 44/50 | Loss: 2.7670 | Train Acc: 0.7812 | Val Acc: 0.6800 | Val ROC-AUC: 0.7264 Epoch 45/50 | Loss: 2.7668 | Train Acc: 0.7805 | Val Acc: 0.6800 | Val ROC-AUC: 0.7269 Epoch 46/50 | Loss: 2.7668 | Train Acc: 0.7790 | Val Acc: 0.6760 | Val ROC-AUC: 0.7229 Epoch 47/50 | Loss: 2.7668 | Train Acc: 0.7826 | Val Acc: 0.6815 | Val ROC-AUC: 0.7273 Epoch 48/50 | Loss: 2.7669 | Train Acc: 0.7847 | Val Acc: 0.6865 | Val ROC-AUC: 0.7319 Epoch 49/50 | Loss: 2.7668 | Train Acc: 0.7829 | Val Acc: 0.6835 | Val ROC-AUC: 0.7302 Epoch 50/50 | Loss: 2.7668 | Train Acc: 0.7825 | Val Acc: 0.6810 | Val ROC-AUC: 0.7301 Best Validation Accuracy: 0.6970 Final Test Accuracy: 0.6810 Final Test ROC-AUC: 0.7336 In [23]: 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() Training vs Validation Accuracy Training Accuracy -x- Validation Accuracy 0.78 0.76 0.74 0.72 Accuracy 0.70 0.68 0.66 0.64 20 30 Epochs In [24]: 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 2.7705 2.7700 2.7695 2.7690 2.7685 2.7680 2.7675 2.7670 Epochs In [25]: plt.figure(figsize=(8, 6)) plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {test_auc:.4f})', color='green') 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() Receiver Operating Characteristic (ROC) Curve ROC Curve (AUC = 0.7336) 0.8 **True Positive Rate** 0.6 0.2 0.0 0.2 0.0 0.8 1.0 False Positive Rate In []:

Specific_Task_01

Collecting torch-geometric

In [1]: pip install torch-geometric

h-geometric) (2.4.6)

c) (25.1.0)

ometric) (6.1.0)

etric) (1.18.3)

rch-geometric) (3.4.1)

ometric) (2.3.0)

ometric) (2025.1.31)

geometric) (2024.2.0)

import numpy as np

import torch.nn as nn

import networkx as nx

import torch.optim as optim
import matplotlib.pyplot as plt

from torch.optim import Adam
import pytorch_lightning as pl
import torch.nn.functional as F
from torch.nn import Linear

from mpl_toolkits.mplot3d import Axes3D

with h5py.File(file, "r") as f:

Keys in dataset: ['X_jets', 'm0', 'pt', 'y']

with h5py.File(file_name, 'r') as f:

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

from skimage.transform import resize
Resize to 128x128 and normalize

Dataset keys: ['X_jets', 'm0', 'pt', 'y']

print("Dataset keys:", list(f.keys()))
print("Total images:", len(f['X_jets']))

X = np.array(f['X_jets'][:sample_size])

print("Image dimensions:", f['X_jets'].shape[1:])

y = np.array(f['y'][:sample_size], dtype=np.int64)

metadata = np.column_stack((f['m0'][:sample_size], f['pt'][:sample_size]))

processed = np.array([resize(img, (128, 128), anti_aliasing=True) for img in images], dtype=np.float32)

for key in f.keys():

Shape of X_jets: (139306, 125, 125, 3)

from sklearn.metrics import roc_curve, auc
from torch_geometric.data import Data, Batch
from torch_geometric.loader import DataLoader

from sklearn.model_selection import train_test_split

In [3]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print("Keys in dataset:", list(f.keys()))

print(f"Shape of {key}: {f[key].shape}")

file_path = "/kaggle/input/autoencoder-data/quark-gluon_data-set_n139306.hdf5"

import torch

torch-geometric) (1.2.0)

torch-geometric) (2024.2.0)

0, >=4.5-aiohttp->torch-geometric) (4.12.2)

enmp>=2024->mkl->numpy->torch-geometric) (2024.2.0)

Installing collected packages: torch-geometric Successfully installed torch-geometric-2.6.1

Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)

Note: you may need to restart the kernel to use updated packages.

1.1)

In [2]: import h5py

In [4]: print(device)

cuda

In [5]: def explore_hdf5(file):

explore_hdf5(file_path)

Shape of m0: (139306,) Shape of pt: (139306,) Shape of y: (139306,)

In [6]: def load_data(file_name, sample_size):

return X, y, metadata

Image dimensions: (125, 125, 3)

Total images: 139306

In [7]: def preprocess_images(images):

quark/gluon Classification using Contrastive Loss

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: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geom

Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch-geometr

Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (1.3.

Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric) (0.

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: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->to

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

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torch-ge

Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from multidict<7.

Requirement already satisfied: intel-openmp>=2024 in /usr/local/lib/python3.10/dist-packages (from mkl->numpy->torch-

Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dist-packages (from mkl->numpy->torch-geometr

Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl->numpy->

Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.10/dist-packages (from mkl_umath->numpy->

Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/lib/python3.10/dist-packages (from intel-op

--- 1.1/1.1 MB 22.7 MB/s eta 0:00:00

Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric)

Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist-packages (from numpy->torch-geometric)

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 2.9 MB/s eta 0:00:00