```
# Code originally written on kaggle
!pip install torch-scatter torch-sparse torch-cluster torch-spline-
conv -f https://data.pyg.org/whl/torch-2.5.0+cu121.html
!pip install torch-geometric
!pip install torch-cluster -f https://data.pyg.org/whl/torch-
2.5.0+cu121.html

import h5py
import numpy as np
import torch
import random
from tqdm import tqdm
from torch_geometric.data import Data
from torch_geometric.nn import knn_graph

# Select device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Dataset paths
file_path = "/kaggle/input/quark-gluon/quark-gluon_data-
set_n139306.hdf5"
output_file = "/kaggle/working/graph_gae_dataset.pt"

# Image dimensions
H, W = 125, 125
x_coords_np, y_coords_np = np.meshgrid(np.linspace(0, 1, W),
np.linspace(0, 1, H))

x_coords = torch.tensor(x_coords_np, dtype=torch.float32,
device=device)
y_coords = torch.tensor(y_coords_np, dtype=torch.float32,
device=device)

# Load HDF5 file
with h5py.File(file_path, "r") as f:
    X_jets = f["X_jets"]
    m0 = f["m0"]
    pt = f["pt"]
    y = f["y"]

    total_events = X_jets.shape[0]
    num_samples = min(50000, total_events)

    # Randomly select exactly 50,000 indices (sorted for consistency)
    random.seed(42)
    selected_indices = sorted(random.sample(range(total_events),
num_samples))

    print(f"Processing {num_samples} randomly selected samples out of
{total_events} available.")

    graph_dataset = []  # List to store PyG Data objects
    batch_size = 5000

    for start in tqdm(range(0, num_samples, batch_size),
desc="Processing Batches"):
        end = min(start + batch_size, num_samples)
        batch_indices = selected_indices[start:end]

        # Load a batch and move to GPU if available
        X_batch = torch.tensor(X_jets[batch_indices],
dtype=torch.float32, device=device)
        m0_batch = torch.tensor(m0[batch_indices],
dtype=torch.float32, device=device)
        pt_batch = torch.tensor(pt[batch_indices],
dtype=torch.float32, device=device)
        y_batch = torch.tensor(y[batch_indices], dtype=torch.float32,
device=device)

        for i in range(end - start):  # Removed tqdm from this loop
            nonzero_mask = torch.any(X_batch[i] > 0, dim=-1)

            nonzero_x = x_coords[nonzero_mask]
            nonzero_y = y_coords[nonzero_mask]
            nonzero_features = X_batch[i][nonzero_mask]

            if nonzero_x.numel() == 0:  # Skip empty events
                continue

            num_points = nonzero_x.shape[0]
            node_features = torch.stack([
                nonzero_x, nonzero_y,
                nonzero_features[:, 0], nonzero_features[:, 1],
nonzero_features[:, 2],
                torch.full((num_points,), m0_batch[i],
dtype=torch.float32, device=device),
                torch.full((num_points,), pt_batch[i],
dtype=torch.float32, device=device)
            ], dim=1)

            # Construct edges using k-NN
            k = 8
            edge_index = knn_graph(node_features[:, :5], k=k)

            # Create PyG Data object
            data = Data(x=node_features, edge_index=edge_index,
y=y_batch[i])
            graph_dataset.append(data)
```

```python
# Save graph dataset
torch.save(graph_dataset, output_file)
print(f"Graph dataset saved to '{output_file}'.")
```

```
Looking in links: https://data.pyg.org/whl/torch-2.5.0+cu121.html
Collecting torch-scatter
  Downloading
https://data.pyg.org/whl/torch-2.5.0%2Bcu121/torch_scatter-
2.1.2%2Bpt25cu121-cp310-cp310-linux_x86_64.whl (10.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.9/10.9 MB 83.4 MB/s eta
0:00:00:00:01:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.1/5.1 MB 90.8 MB/s eta
0:00:00:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 35.0 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 991.6/991.6 kB 10.9 MB/s eta
0:00:0000:0100:01
ent already satisfied: scipy in /usr/local/lib/python3.10/dist-
packages (from torch-sparse) (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in
/usr/local/lib/python3.10/dist-packages (from scipy->torch-sparse)
(1.26.4)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-
packages (from numpy<2.3,>=1.22.4->scipy->torch-sparse) (2025.0.1)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (2022.0.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (2.4.1)
Requirement already satisfied: intel-openmp>=2024 in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy<2.3,>=1.22.4-
>scipy->torch-sparse) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl-
>numpy<2.3,>=1.22.4->scipy->torch-sparse) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.10/dist-packages (from mkl_umath-
>numpy<2.3,>=1.22.4->scipy->torch-sparse) (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<2.3,>=1.22.4->scipy->torch-sparse) (2024.2.0)
Installing collected packages: torch-spline-conv, torch-scatter,
torch-sparse, torch-cluster
Successfully installed torch-cluster-1.6.3+pt25cu121 torch-scatter-
2.1.2+pt25cu121 torch-sparse-0.6.18+pt25cu121 torch-spline-conv-
1.2.2+pt25cu121
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
ent 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 21.7 MB/s eta
0:00:0000:01
etric
Successfully installed torch-geometric-2.6.1
Looking in links: https://data.pyg.org/whl/torch-2.5.0+cu121.html
Requirement already satisfied: torch-cluster in
/usr/local/lib/python3.10/dist-packages (1.6.3+pt25cu121)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from torch-cluster) (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in
/usr/local/lib/python3.10/dist-packages (from scipy->torch-cluster)
(1.26.4)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-
packages (from numpy<2.3,>=1.22.4->scipy->torch-cluster) (2025.0.1)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (2022.0.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.10/dist-packages (from numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (2.4.1)
Requirement already satisfied: intel-openmp>=2024 in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy<2.3,>=1.22.4-
>scipy->torch-cluster) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl-
>numpy<2.3,>=1.22.4->scipy->torch-cluster) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.10/dist-packages (from mkl_umath-
>numpy<2.3,>=1.22.4->scipy->torch-cluster) (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<2.3,>=1.22.4->scipy->torch-cluster) (2024.2.0)
Using device: cuda
Processing 50000 randomly selected samples out of 139306 available.

Processing Batches: 100%|██████████| 10/10 [12:08<00:00, 72.81s/it]

Graph dataset saved to '/kaggle/working/graph_gae_dataset.pt'.
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, GAE
from torch_geometric.loader import DataLoader
import torch_geometric.utils as pyg_utils
from tqdm import tqdm
import time
from torch.cuda.amp import autocast, GradScaler

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

try:
    graph_dataset = torch.load("/kaggle/working/graph_gae_dataset.pt")
    print(f"Dataset loaded successfully with {len(graph_dataset)}
samples")
except Exception as e:
    raise RuntimeError(f"Error loading dataset: {e}")

# Ensure dataset is valid
if not graph_dataset or not isinstance(graph_dataset, list):
    raise ValueError("Error: Loaded dataset is empty or incorrectly
formatted!")

graph_dataset = [data.to('cpu') for data in graph_dataset]

in_channels = graph_dataset[0].x.shape[1]
print(f"Node feature dimensions: {in_channels}")

train_size = int(0.8 * len(graph_dataset))
train_dataset = graph_dataset[:train_size]
val_dataset = graph_dataset[train_size:]
print(f"Training samples: {len(train_dataset)}, Validation samples:
{len(val_dataset)}")

train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True,
num_workers=4,
                          pin_memory=(device.type == "cuda"))
val_loader = DataLoader(val_dataset, batch_size=256, shuffle=False,
num_workers=2,
                       pin_memory=(device.type == "cuda"))

class GraphAutoencoder(nn.Module):
    def __init__(self, in_channels, hidden_dim=128, latent_dim=64):
        super(GraphAutoencoder, self).__init__()
        # Encoder
        self.conv1 = GCNConv(in_channels, hidden_dim)
        self.bn1 = nn.BatchNorm1d(hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.bn2 = nn.BatchNorm1d(hidden_dim)
        self.conv3 = GCNConv(hidden_dim, latent_dim)

    def encode(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = self.bn1(x)
        x = F.relu(x)
        x = F.dropout(x, p=0.2, training=self.training)

        x = self.conv2(x, edge_index)
        x = self.bn2(x)
        x = F.relu(x)
        x = F.dropout(x, p=0.2, training=self.training)

        x = self.conv3(x, edge_index)
        return x  # Latent representation

    def decode(self, z, edge_index):
        # Inner product decoder with clamping to prevent NaNs
        score = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)
        return torch.clamp(score, -10, 10)

    def forward(self, x, edge_index):
        return self.encode(x, edge_index)

model = GAE(GraphAutoencoder(in_channels)).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.003,
weight_decay=1e-5)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
'min', factor=0.5, patience=3, verbose=True)

scaler = GradScaler()

def train(epoch):
    model.train()
    total_loss = 0
    total_examples = 0

    start_time = time.time()
    for batch in tqdm(train_loader, desc=f"Epoch {epoch}/{num_epochs}
[Train]", leave=False):
        batch = batch.to(device)
        optimizer.zero_grad()

        with autocast(enabled=device.type == "cuda"):
            # Encode graph
            z = model.encode(batch.x, batch.edge_index)

            pos_edge_index = batch.edge_index

            neg_edge_index = pyg_utils.negative_sampling(
                edge_index=pos_edge_index.cpu(),
                num_nodes=batch.x.size(0),
                num_neg_samples=pos_edge_index.shape[1]
            ).to(device)

            edge_index = torch.cat([pos_edge_index, neg_edge_index],
dim=1)
            edge_label = torch.cat([
                torch.ones(pos_edge_index.shape[1]),
                torch.zeros(neg_edge_index.shape[1])
            ]).to(device)

            # Decode
            edge_pred = model.decode(z, edge_index)

            # BCE loss
            loss = F.binary_cross_entropy_with_logits(edge_pred,
edge_label)

        scaler.scale(loss).backward()

        scaler.unscale_(optimizer)
        torch.nn.utils.clip_grad_norm_(model.parameters(),
max_norm=1.0)

        # Step with scaler
        scaler.step(optimizer)
        scaler.update()

        total_loss += loss.item() * batch.num_graphs
        total_examples += batch.num_graphs

    epoch_time = time.time() - start_time
    return total_loss / total_examples, epoch_time

def validate():
    model.eval()
    total_loss = 0
    total_examples = 0

    with torch.no_grad():
        for batch in tqdm(val_loader, desc="Validation", leave=False):
            batch = batch.to(device)

            with autocast(enabled=device.type == "cuda"):
                # Encode graph
                z = model.encode(batch.x, batch.edge_index)

                pos_edge_index = batch.edge_index

                neg_edge_index = pyg_utils.negative_sampling(
                    edge_index=pos_edge_index.cpu(),
                    num_nodes=batch.x.size(0),
                    num_neg_samples=pos_edge_index.shape[1]
                ).to(device)

                edge_index = torch.cat([pos_edge_index,
neg_edge_index], dim=1)
                edge_label = torch.cat([
                    torch.ones(pos_edge_index.shape[1]),
                    torch.zeros(neg_edge_index.shape[1])
                ]).to(device)

                # Decode
                edge_pred = model.decode(z, edge_index)

                # BCE loss
                loss = F.binary_cross_entropy_with_logits(edge_pred,
edge_label)

            total_loss += loss.item() * batch.num_graphs
            total_examples += batch.num_graphs

    return total_loss / total_examples

num_epochs = 20
best_val_loss = float('inf')
patience = 5
counter = 0

print(f"Starting training with mixed precision {'enabled' if
device.type == 'cuda' else 'disabled (CUDA required)'}")

for epoch in range(1, num_epochs + 1):
    train_loss, epoch_time = train(epoch)

    val_loss = validate()
```

```python
    print(f"Epoch {epoch}/{num_epochs} - Train Loss: {train_loss:.4f}, 
Val Loss: {val_loss:.4f}, Time: {epoch_time:.2f}s")

    scheduler.step(val_loss)

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), 
"/kaggle/working/gae_model_best.pt")
        print(f"New best model saved with validation loss: 
{best_val_loss:.4f}")
        counter = 0
    else:
        counter += 1

    if counter >= patience:
        print(f"Early stopping after {epoch} epochs without 
improvement")
        break

torch.save(model.state_dict(), "/kaggle/working/gae_model_final.pt")
print("Training completed. Final model saved.")

model.load_state_dict(torch.load("/kaggle/working/gae_model_best.pt"))

val_loss = validate()
print(f"Best model validation loss: {val_loss:.4f}")
```

```
Using device: cuda
```

```
<ipython-input-3-108ed1b82b94>:16: FutureWarning: You are using 
`torch.load` with `weights_only=False` (the current default value), 
which uses the default pickle module implicitly. It is possible to 
construct malicious pickle data which will execute arbitrary code 
during unpickling (See 
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for 
`weights_only` will be flipped to `True`. This limits the functions 
that could be executed during unpickling. Arbitrary objects will no 
longer be allowed to be loaded via this mode unless they are 
explicitly allowlisted by the user via 
`torch.serialization.add_safe_globals`. We recommend you start setting 
`weights_only=True` for any use case where you don't have full control 
of the loaded file. Please open an issue on GitHub for any issues 
related to this experimental feature.
  graph_dataset = torch.load("/kaggle/working/graph_gae_dataset.pt")
```

```
Dataset loaded successfully with 50000 samples
Node feature dimensions: 7
Training samples: 40000, Validation samples: 10000
```

```
<ipython-input-3-108ed1b82b94>:86: FutureWarning:
`torch.cuda.amp.GradScaler(args...)` is deprecated. Please use
`torch.amp.GradScaler('cuda', args...)` instead.
  scaler = GradScaler()

Starting training with mixed precision enabled

Epoch 1/20 [Train]:   0%|          | 0/157 [00:00<?, ?it/s]<ipython-
input-3-108ed1b82b94>:100: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
  with autocast(enabled=device.type == "cuda"):
Validation:   0%|          | 0/40 [00:00<?, ?it/s]
<ipython-input-3-108ed1b82b94>:155: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
  with autocast(enabled=device.type == "cuda"):


Epoch 1/20 - Train Loss: 0.5996, Val Loss: 0.5911, Time: 329.24s
New best model saved with validation loss: 0.5911



Epoch 2/20 - Train Loss: 0.5740, Val Loss: 0.5916, Time: 318.55s



Epoch 3/20 - Train Loss: 0.5651, Val Loss: 0.5716, Time: 319.09s
New best model saved with validation loss: 0.5716



Epoch 4/20 - Train Loss: 0.5602, Val Loss: 0.5904, Time: 319.35s



Epoch 5/20 - Train Loss: 0.5564, Val Loss: 0.5798, Time: 319.93s



Epoch 6/20 - Train Loss: 0.5541, Val Loss: 0.5829, Time: 317.25s



Epoch 7/20 - Train Loss: 0.5525, Val Loss: 0.5772, Time: 321.21s



Epoch 8/20 - Train Loss: 0.5506, Val Loss: 0.5524, Time: 318.57s
New best model saved with validation loss: 0.5524
```

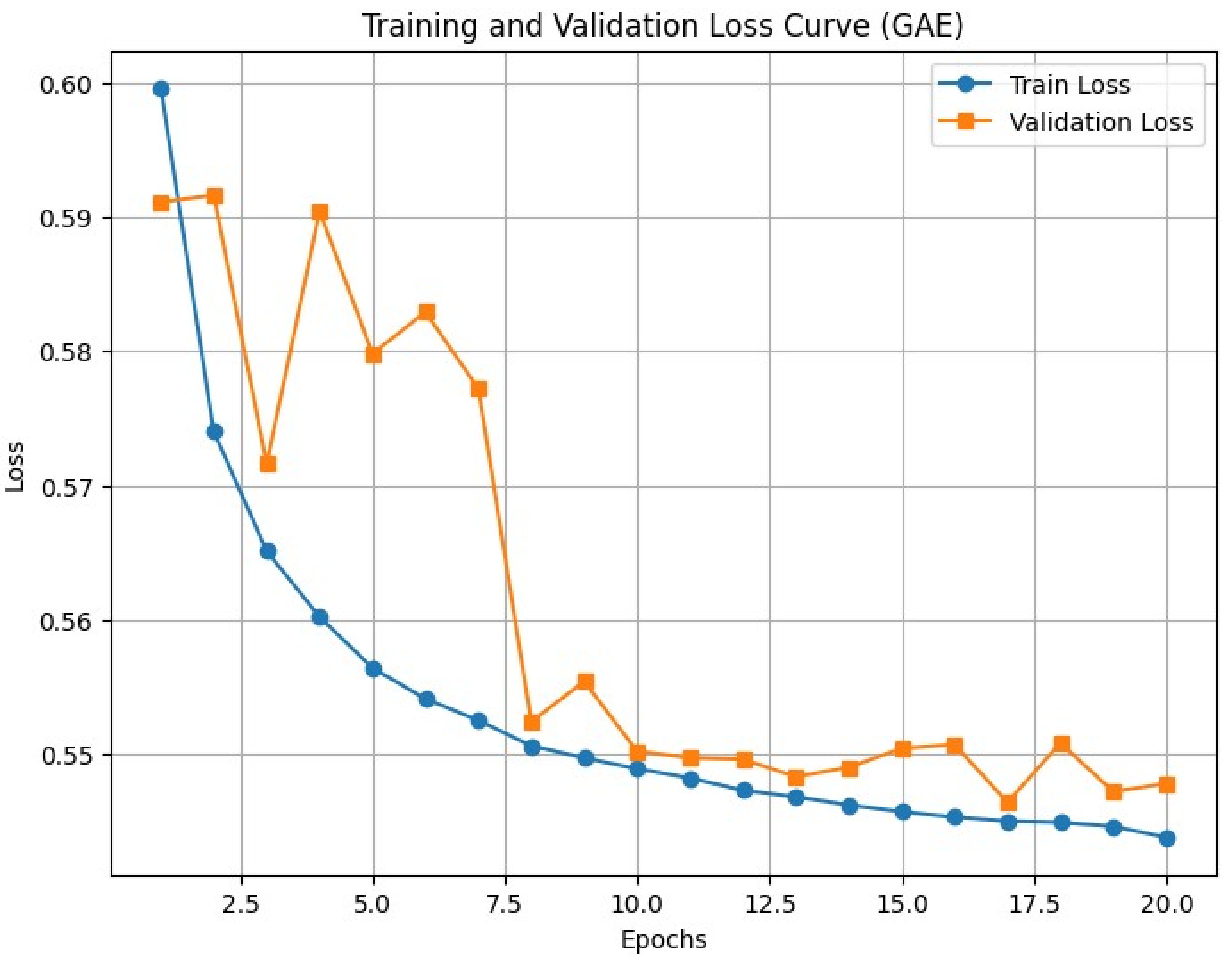
```
Epoch 9/20 - Train Loss: 0.5497, Val Loss: 0.5554, Time: 332.44s



Epoch 10/20 - Train Loss: 0.5489, Val Loss: 0.5502, Time: 334.55s
New best model saved with validation loss: 0.5502



Epoch 11/20 - Train Loss: 0.5482, Val Loss: 0.5497, Time: 335.07s
New best model saved with validation loss: 0.5497



Epoch 12/20 - Train Loss: 0.5473, Val Loss: 0.5496, Time: 335.31s
New best model saved with validation loss: 0.5496



Epoch 13/20 - Train Loss: 0.5468, Val Loss: 0.5483, Time: 337.69s
New best model saved with validation loss: 0.5483



Epoch 14/20 - Train Loss: 0.5462, Val Loss: 0.5490, Time: 338.61s



Epoch 15/20 - Train Loss: 0.5457, Val Loss: 0.5504, Time: 334.22s



Epoch 16/20 - Train Loss: 0.5453, Val Loss: 0.5507, Time: 337.97s



Epoch 17/20 - Train Loss: 0.5450, Val Loss: 0.5464, Time: 333.34s
New best model saved with validation loss: 0.5464



Epoch 18/20 - Train Loss: 0.5449, Val Loss: 0.5508, Time: 331.72s



Epoch 19/20 - Train Loss: 0.5446, Val Loss: 0.5472, Time: 337.77s

<ipython-input-3-108ed1b82b94>:227: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
```

```
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.

model.load_state_dict(torch.load("/kaggle/working/gae_model_best.pt"))
```

```
Epoch 20/20 - Train Loss: 0.5438, Val Loss: 0.5478, Time: 336.24s
Training completed. Final model saved.
```

```
Best model validation loss: 0.5464
```

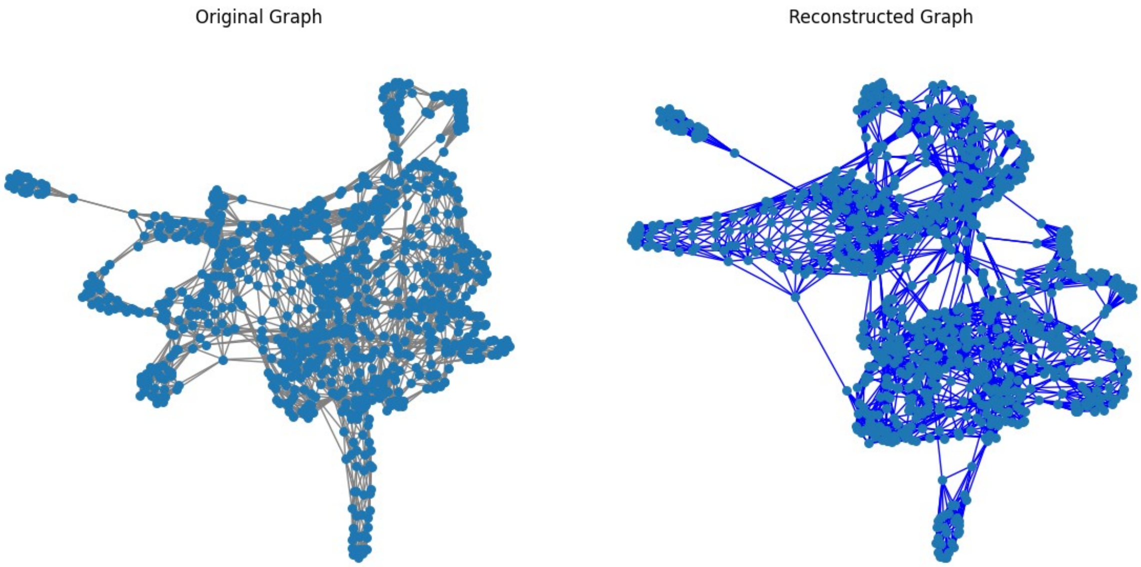
```python
import matplotlib.pyplot as plt

# Training and validation loss values from the logs
epochs = list(range(1, 21))
train_loss = [0.5996, 0.5740, 0.5651, 0.5602, 0.5564, 0.5541, 0.5525,
0.5506, 0.5497, 0.5489, 0.5482, 0.5473, 0.5468, 0.5462, 0.5457,
0.5453, 0.5450, 0.5449, 0.5446, 0.5438]

val_loss = [0.5911, 0.5916, 0.5716, 0.5904, 0.5798, 0.5829, 0.5772,
0.5524, 0.5554, 0.5502, 0.5497, 0.5496, 0.5483, 0.5490, 0.5504,
0.5507, 0.5464, 0.5508, 0.5472, 0.5478]

plt.figure(figsize=(8, 6))
plt.plot(epochs, train_loss, label="Train Loss", marker="o")
plt.plot(epochs, val_loss, label="Validation Loss", marker="s")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss Curve (GAE)")
plt.legend()
plt.grid(True)
plt.show()
```



```
import torch
import matplotlib.pyplot as plt
import networkx as nx
from torch_geometric.utils import to_networkx

model_path = "/kaggle/working/gae_model_best.pt"
model.load_state_dict(torch.load(model_path, map_location=device))
model.eval()

sample_idx = torch.randint(0, len(val_dataset), (1,)).item()

data = val_dataset[sample_idx].to(device)

# Encode and reconstruct
with torch.no_grad():
    z = model.encode(data.x, data.edge_index)
    reconstructed_edge_scores = model.decode(z,
data.edge_index).cpu().numpy()

original_graph = to_networkx(data, to_undirected=True)
```

```
threshold = 0
edges = [(u, v) for (u, v), score in
zip(data.edge_index.t().cpu().numpy(), reconstructed_edge_scores) if
score > threshold]
reconstructed_graph = nx.Graph()
reconstructed_graph.add_edges_from(edges)

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plot Original Graph
ax = axes[0]
ax.set_title("Original Graph")
nx.draw(original_graph, pos=nx.spring_layout(original_graph),
node_size=30, edge_color='gray', ax=ax)

# Plot Reconstructed Graph
ax = axes[1]
ax.set_title("Reconstructed Graph")
nx.draw(reconstructed_graph,
pos=nx.spring_layout(reconstructed_graph), node_size=30,
edge_color='blue', ax=ax)

plt.tight_layout()
plt.show()
```

```
<ipython-input-23-468b46e6cb67>:7: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load(model_path, map_location=device))
```



Original Graph

Reconstructed Graph