```
# Code originally written on kaggle
import h5py
import numpy as np
import torch
import random
from tqdm import tqdm

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

# Paths
file_path = "/kaggle/input/quark-gluon/quark-gluon_data-
set_n139306.hdf5"
output_file = "/kaggle/working/point_cloud_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)

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)  # Limiting to 50,000
samples

    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.")

    dataset = {"point_clouds": [], "labels": []}
    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]
```

```
        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):
            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:
                continue  # Skip empty point clouds

            num_points = nonzero_x.shape[0]
            jet_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)

            dataset["point_clouds"].append(jet_features.cpu())
            dataset["labels"].append(y_batch[i].cpu())

    torch.save(dataset, output_file)
    print(f"Point cloud dataset saved to '{output_file}'.")
```

```
Using device: cuda
Processing 50000 randomly selected samples out of 139306 available.

Processing Batches: 100%|██████████| 10/10 [10:13<00:00, 61.36s/it]

Point cloud dataset saved to '/kaggle/working/point_cloud_dataset.pt'.
```

```
# □ STEP 2: Convert Point Clouds to Graph Dataset
!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 torch
```

```
from torch_geometric.data import Data, InMemoryDataset
from torch_geometric.nn import knn_graph
from tqdm.notebook import tqdm

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

# Paths
input_file = "/kaggle/working/point_cloud_dataset.pt"
output_file = "/kaggle/working/graph_dataset.pt"

# Load point cloud dataset
dataset = torch.load(input_file)

graph_data_list = []

for i in tqdm(range(len(dataset["point_clouds"])), desc="Processing
Graphs"):
    point_cloud = dataset["point_clouds"][i]
    label = dataset["labels"][i]

    if point_cloud.numel() == 0:
        continue  # Skip empty point clouds

    # k-NN Graph Construction (use only spatial features `x, y`)
    k = 8
    edge_index = knn_graph(point_cloud[:, :2], k=k)  # Use only (x, y)
for graph structure

    data = Data(x=point_cloud, edge_index=edge_index, y=label)
    graph_data_list.append(data)

# ✅ Define InMemoryDataset wrapper
class GraphDataset(InMemoryDataset):
    def __init__(self, root=None, data_list=None):
        super().__init__(root)
        if data_list:
            self.data, self.slices = self.collate(data_list)

    def __len__(self):
        return len(self.slices["x"]) - 1

# ✅ Save dataset correctly
graph_dataset = GraphDataset(data_list=graph_data_list)
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 54.6 MB/s eta
0:00:0000:010:01
 5.1/5.1 MB 32.8 MB/s eta
0:00:0000:0100:01
 3.4/3.4 MB 17.1 MB/s eta
0:00:00a 0:00:01
 991.6/991.6 kB 8.2 MB/s eta
0:00:00a 0:00:01
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 2.9 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 21.9 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
```

```
<ipython-input-2-ebd000c1c252>:19: 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.
  dataset = torch.load(input_file)
```

{"model_id":"b75f42077d434e4681d1cbe48ecc2bc0","version_major":2,"version_minor":0}

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

```
dataset = torch.load("/kaggle/working/graph_dataset.pt",
map_location="cpu")
print(dataset)  # Should print dataset details

# Check dataset samples
print(len(dataset))
print(dataset[0])  # Check a sample graph
```

```
<ipython-input-3-300146acece1>:1: 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.
  dataset = torch.load("/kaggle/working/graph_dataset.pt",
map_location="cpu")
```

```
GraphDataset(50000)
50000
Data(x=[515, 7], edge_index=[2, 4120], y=[1])
```

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GINConv, global_mean_pool
from torch_geometric.data import DataLoader, InMemoryDataset
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.utils.class_weight import compute_class_weight
from tqdm import tqdm
import numpy as np
from sklearn.metrics import roc_auc_score
import gc

# Ensure GPU Memory is Cleared
torch.cuda.empty_cache()
gc.collect()

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

# Load Dataset
dataset_path = "/kaggle/working/graph_dataset.pt"
dataset = torch.load(dataset_path, map_location="cpu")

# Remove Empty Graphs
dataset = [data for data in dataset if data.edge_index.numel() > 0 and
data.x.numel() > 0]
assert len(dataset) > 0, "All graphs were found empty after
filtering."

# Split Dataset with Stratification
labels = np.array([data.y.item() for data in dataset])
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
random_state=42)
train_idx, temp_idx = next(splitter.split(np.zeros(len(labels)),
labels))

temp_labels = labels[temp_idx]
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.5,
random_state=42)
val_idx, test_idx = next(splitter.split(np.zeros(len(temp_labels)),
temp_labels))

train_dataset = [dataset[i] for i in train_idx]
val_dataset = [dataset[i] for i in val_idx]
test_dataset = [dataset[i] for i in test_idx]
```

```python
# Feature Normalization (Only Fit on Train Data)
scaler = StandardScaler()
train_features = np.vstack([data.x.cpu().numpy().astype(np.float32)
for data in train_dataset])
scaler.fit(train_features)

for data in dataset:
    data.x = torch.tensor(scaler.transform(data.x.cpu().numpy()),
dtype=torch.float32)

# Compute Class Weights
class_weights = compute_class_weight("balanced",
classes=np.unique(labels), y=labels)
class_weights = torch.tensor(class_weights,
dtype=torch.float32).to(device)

# Data Loaders
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, num_workers=2, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size,
num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
num_workers=2, pin_memory=True)

# GIN Model
class GIN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels=128,
out_channels=1, num_layers=3):
        super(GIN, self).__init__()
        self.convs = torch.nn.ModuleList()
        self.batch_norms = torch.nn.ModuleList()
        self.dropouts = torch.nn.ModuleList()

        nn1 = torch.nn.Sequential(
            torch.nn.Linear(in_channels, hidden_channels),
            torch.nn.LeakyReLU(0.1),
            torch.nn.Linear(hidden_channels, hidden_channels)
        )
        self.convs.append(GINConv(nn1))
        self.batch_norms.append(torch.nn.BatchNorm1d(hidden_channels))
        self.dropouts.append(torch.nn.Dropout(0.2))

        for _ in range(num_layers - 1):
            nn_layer = torch.nn.Sequential(
                torch.nn.Linear(hidden_channels, hidden_channels),
                torch.nn.LeakyReLU(0.1),
                torch.nn.Linear(hidden_channels, hidden_channels)
            )
            self.convs.append(GINConv(nn_layer))
```

```
self.batch_norms.append(torch.nn.BatchNorm1d(hidden_channels))
            self.dropouts.append(torch.nn.Dropout(0.2))

        self.fc = torch.nn.Linear(hidden_channels, out_channels)

    def forward(self, x, edge_index, batch):
        for conv, bn, do in zip(self.convs, self.batch_norms,
self.dropouts):
            x = conv(x, edge_index)
            x = bn(x)
            x = F.leaky_relu(x, negative_slope=0.1)
            x = do(x)

        x = global_mean_pool(x, batch)
        return self.fc(x).squeeze()

# Initialize Model
model = GIN(in_channels=7).to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=0.0015,
weight_decay=5e-5)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode="max", factor=0.5, patience=5, verbose=True)
criterion = torch.nn.BCEWithLogitsLoss(pos_weight=class_weights[1])

# Training Function
def train():
    model.train()
    total_loss = 0
    pbar = tqdm(train_loader, desc="Training", leave=False)
    for data in pbar:
        data = data.to(device)

        optimizer.zero_grad()
        output = model(data.x, data.edge_index, data.batch)
        loss = criterion(output, data.y.float().view(-1))

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(),
max_norm=1.0)  # Dynamic Gradient Clipping
        optimizer.step()

        total_loss += loss.item()
        pbar.set_postfix(loss=loss.item())

    return total_loss / len(train_loader)

# Evaluation Function
def evaluate(loader):
    model.eval()
```

```python
    total_loss = 0
    preds, labels = [], []
    with torch.inference_mode():
        pbar = tqdm(loader, desc="Evaluating", leave=False)
        for data in pbar:
            data = data.to(device)
            output = model(data.x, data.edge_index, data.batch)
            loss = criterion(output, data.y.float().view(-1))
            total_loss += loss.item()
            preds.append(output.sigmoid().cpu())
            labels.append(data.y.cpu())

    if not preds:
        return total_loss / len(loader), None, None

    preds = torch.cat(preds).numpy()
    labels = torch.cat(labels).numpy()
    acc = ((preds > 0.5) == labels).mean()
    try:
        auc = roc_auc_score(labels, preds) if len(set(labels)) > 1
else 0.5
    except ValueError:
        auc = 0.5
    return total_loss / len(loader), acc, auc

# Training Loop
num_epochs = 50
best_val_auc = 0
patience = 5
no_improve_epochs = 0

for epoch in range(num_epochs):
    train_loss = train()
    val_loss, val_acc, val_auc = evaluate(val_loader)

    print(f"Epoch {epoch+1}/{num_epochs} - Train Loss:
{train_loss:.4f} - Val Loss: {val_loss:.4f} - Val Acc: {val_acc:.4f} -
Val AUC: {val_auc:.4f}")

    scheduler.step(val_auc)

    if val_auc and val_auc > best_val_auc:
        best_val_auc = val_auc
        torch.save(model.state_dict(),
"/kaggle/working/best_gin_model.pt")
        no_improve_epochs = 0
    else:
        no_improve_epochs += 1

    if no_improve_epochs >= patience:
```

```
        print("Early stopping triggered")
        break

torch.cuda.empty_cache()
```

```
Using device: cuda
```

```
<ipython-input-3-856d2b5040e3>:22: 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.
  dataset = torch.load(dataset_path, map_location="cpu")
/usr/local/lib/python3.10/dist-packages/torch_geometric/deprecation.py
:26: UserWarning: 'data.DataLoader' is deprecated, use
'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62
: UserWarning: The verbose parameter is deprecated. Please use
get_last_lr() to access the learning rate.
  warnings.warn(
```

```
Epoch 1/50 - Train Loss: 0.5895 - Val Loss: 0.5887 - Val Acc: 0.7094 -
Val AUC: 0.7759
```

```
Epoch 2/50 - Train Loss: 0.5812 - Val Loss: 0.5852 - Val Acc: 0.7140 -
Val AUC: 0.7780
```

```
Epoch 3/50 - Train Loss: 0.5771 - Val Loss: 0.5867 - Val Acc: 0.7076 -
Val AUC: 0.7791
```

```
Epoch 4/50 - Train Loss: 0.5737 - Val Loss: 0.5742 - Val Acc: 0.7128 -
Val AUC: 0.7812
```

```
Epoch 5/50 - Train Loss: 0.5720 - Val Loss: 0.5723 - Val Acc: 0.7182 - Val AUC: 0.7830

Epoch 6/50 - Train Loss: 0.5693 - Val Loss: 0.5704 - Val Acc: 0.7140 - Val AUC: 0.7812

Epoch 7/50 - Train Loss: 0.5675 - Val Loss: 0.5714 - Val Acc: 0.7176 - Val AUC: 0.7817

Epoch 8/50 - Train Loss: 0.5665 - Val Loss: 0.5759 - Val Acc: 0.7188 - Val AUC: 0.7832

Epoch 9/50 - Train Loss: 0.5661 - Val Loss: 0.5646 - Val Acc: 0.7174 - Val AUC: 0.7858

Epoch 10/50 - Train Loss: 0.5647 - Val Loss: 0.7402 - Val Acc: 0.5184 - Val AUC: 0.7588

Epoch 11/50 - Train Loss: 0.5639 - Val Loss: 0.5735 - Val Acc: 0.7184 - Val AUC: 0.7825

Epoch 12/50 - Train Loss: 0.5637 - Val Loss: 0.5710 - Val Acc: 0.7178 - Val AUC: 0.7843

Epoch 13/50 - Train Loss: 0.5624 - Val Loss: 0.5646 - Val Acc: 0.7208 - Val AUC: 0.7880

Epoch 14/50 - Train Loss: 0.5603 - Val Loss: 0.5627 - Val Acc: 0.7246 - Val AUC: 0.7874

Epoch 15/50 - Train Loss: 0.5612 - Val Loss: 0.5644 - Val Acc: 0.7248 - Val AUC: 0.7884
```

```
Epoch 16/50 - Train Loss: 0.5603 - Val Loss: 0.5621 - Val Acc: 0.7256
- Val AUC: 0.7883

Epoch 17/50 - Train Loss: 0.5602 - Val Loss: 0.5664 - Val Acc: 0.7202
- Val AUC: 0.7886

Epoch 18/50 - Train Loss: 0.5594 - Val Loss: 0.5691 - Val Acc: 0.7216
- Val AUC: 0.7882

Epoch 19/50 - Train Loss: 0.5580 - Val Loss: 0.5685 - Val Acc: 0.7240
- Val AUC: 0.7878

Epoch 20/50 - Train Loss: 0.5587 - Val Loss: 0.5680 - Val Acc: 0.7270
- Val AUC: 0.7881

Epoch 21/50 - Train Loss: 0.5578 - Val Loss: 0.5591 - Val Acc: 0.7236
- Val AUC: 0.7904

Epoch 22/50 - Train Loss: 0.5578 - Val Loss: 0.5654 - Val Acc: 0.7226
- Val AUC: 0.7880

Epoch 23/50 - Train Loss: 0.5570 - Val Loss: 0.5590 - Val Acc: 0.7236
- Val AUC: 0.7905

Epoch 24/50 - Train Loss: 0.5577 - Val Loss: 0.5613 - Val Acc: 0.7214
- Val AUC: 0.7905

Epoch 25/50 - Train Loss: 0.5565 - Val Loss: 0.5645 - Val Acc: 0.7216
- Val AUC: 0.7898

Epoch 26/50 - Train Loss: 0.5565 - Val Loss: 0.5634 - Val Acc: 0.7268
- Val AUC: 0.7880
```

```
Epoch 27/50 - Train Loss: 0.5564 - Val Loss: 0.5647 - Val Acc: 0.7268
- Val AUC: 0.7915

Epoch 28/50 - Train Loss: 0.5561 - Val Loss: 0.5603 - Val Acc: 0.7274
- Val AUC: 0.7901

Epoch 29/50 - Train Loss: 0.5560 - Val Loss: 0.5628 - Val Acc: 0.7206
- Val AUC: 0.7892

Epoch 30/50 - Train Loss: 0.5554 - Val Loss: 0.5637 - Val Acc: 0.7138
- Val AUC: 0.7869

Epoch 31/50 - Train Loss: 0.5554 - Val Loss: 0.5619 - Val Acc: 0.7254
- Val AUC: 0.7925

Epoch 32/50 - Train Loss: 0.5556 - Val Loss: 0.5612 - Val Acc: 0.7246
- Val AUC: 0.7909

Epoch 33/50 - Train Loss: 0.5552 - Val Loss: 0.5621 - Val Acc: 0.7260
- Val AUC: 0.7910

Epoch 34/50 - Train Loss: 0.5555 - Val Loss: 0.5575 - Val Acc: 0.7288
- Val AUC: 0.7922

Epoch 35/50 - Train Loss: 0.5556 - Val Loss: 0.5559 - Val Acc: 0.7254
- Val AUC: 0.7915

Epoch 36/50 - Train Loss: 0.5542 - Val Loss: 0.5660 - Val Acc: 0.7238
- Val AUC: 0.7901
Early stopping triggered
```

```python
import matplotlib.pyplot as plt
import numpy as np

epochs = np.arange(1, 37)
```

```
train_loss = [0.5895, 0.5812, 0.5771, 0.5737, 0.5720, 0.5693, 0.5675,
0.5665, 0.5661, 0.5647,
              0.5639, 0.5637, 0.5624, 0.5603, 0.5612, 0.5603, 0.5602,
0.5594, 0.5580, 0.5587,
              0.5578, 0.5578, 0.5570, 0.5577, 0.5565, 0.5565, 0.5564,
0.5561, 0.5560, 0.5554,
              0.5554, 0.5556, 0.5552, 0.5555, 0.5556, 0.5542]

val_loss = [0.5887, 0.5852, 0.5867, 0.5742, 0.5723, 0.5704, 0.5714,
0.5759, 0.5646, 0.7402,
            0.5735, 0.5710, 0.5646, 0.5627, 0.5644, 0.5621, 0.5664,
0.5691, 0.5685, 0.5680,
            0.5591, 0.5654, 0.5590, 0.5613, 0.5645, 0.5634, 0.5647,
0.5603, 0.5628, 0.5637,
            0.5619, 0.5612, 0.5621, 0.5575, 0.5559, 0.5660]

val_acc = [0.7094, 0.7140, 0.7076, 0.7128, 0.7182, 0.7140, 0.7176,
0.7188, 0.7174, 0.5184,
           0.7184, 0.7178, 0.7208, 0.7246, 0.7248, 0.7256, 0.7202,
0.7216, 0.7240, 0.7270,
           0.7236, 0.7226, 0.7236, 0.7214, 0.7216, 0.7268, 0.7268,
0.7274, 0.7206, 0.7138,
           0.7254, 0.7246, 0.7260, 0.7288, 0.7254, 0.7238]

val_auc = [0.7759, 0.7780, 0.7791, 0.7812, 0.7830, 0.7812, 0.7817,
0.7832, 0.7858, 0.7588,
           0.7825, 0.7843, 0.7880, 0.7874, 0.7884, 0.7883, 0.7886,
0.7882, 0.7878, 0.7881,
           0.7904, 0.7880, 0.7905, 0.7905, 0.7898, 0.7880, 0.7915,
0.7901, 0.7892, 0.7869,
           0.7925, 0.7909, 0.7910, 0.7922, 0.7915, 0.7901]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, label="Train Loss", marker='o',
linestyle='dashed')
plt.plot(epochs, val_loss, label="Val Loss", marker='s',
linestyle='dashed')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(epochs, val_acc, label="Val Accuracy", marker='o',
linestyle='dashed')
plt.plot(epochs, val_auc, label="Val AUC", marker='s',
linestyle='dashed')
plt.xlabel("Epochs")
```

```
plt.ylabel("Metrics")
plt.title("Validation Accuracy & AUC")
plt.legend()
plt.grid()

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