Group ID: GNN Group 01

Group Members Name with Student ID:

Name	Bits ID	Contribution
Adarsh Sivanandan	2023AA05811	100%
Mano Ranjan Sahu	2023AA05738	100%
Mukesh Kumar Saini	2023aa05880	100%
Sanjay Kumar Agarwal	2023AA05611	100%
Vaibhav Bajpai	2023aa05631	100%

Link of Python code in BITS library needs to be given as we will be running the code in Lab environment.

Python code file path in virtual lab: /home/labuser/Desktop/Persistent_Folder/Group_1/node_classification_solution_Group_01.ipynb

In [ ]:
```
!pip install torch torchvision torchaudio
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torch in /home/labuser/.local/lib/python3.10/site-pac
kages (2.3.1+cpu)
Requirement already satisfied: torchvision in /home/labuser/.local/lib/python3.10/si
te-packages (0.18.1+cpu)
Requirement already satisfied: torchaudio in /home/labuser/.local/lib/python3.10/sit
e-packages (2.3.1+cpu)
Requirement already satisfied: typing-extensions>=4.8.0 in /home/labuser/.local/lib/
python3.10/site-packages (from torch) (4.12.2)
Requirement already satisfied: jinja2 in /usr/lib/python3/dist-packages (from torch)
(3.0.3)
Requirement already satisfied: filelock in /home/labuser/.local/lib/python3.10/site-
packages (from torch) (3.13.1)
Requirement already satisfied: networkx in /home/labuser/.local/lib/python3.10/site-
packages (from torch) (3.3)
Requirement already satisfied: fsspec in /home/labuser/.local/lib/python3.10/site-pa
ckages (from torch) (2024.6.0)
Requirement already satisfied: sympy in /home/labuser/.local/lib/python3.10/site-pac
kages (from torch) (1.12)
Requirement already satisfied: numpy in /home/labuser/.local/lib/python3.10/site-pac
kages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /home/labuser/.local/lib/pyt
hon3.10/site-packages (from torchvision) (10.3.0)
Requirement already satisfied: mpmath>=0.19 in /home/labuser/.local/lib/python3.10/s
ite-packages (from sympy->torch) (1.3.0)
```

In [ ]:
```
!pip install torch-scatter torch-sparse torch-cluster torch-spline-conv -f https://
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: https://data.pyg.org/whl/torch-2.3.1+cpu.html
Requirement already satisfied: torch-scatter in /home/labuser/.local/lib/python3.10/
site-packages (2.1.2+pt23cpu)
Requirement already satisfied: torch-sparse in /home/labuser/.local/lib/python3.10/s
ite-packages (0.6.18+pt23cpu)
Requirement already satisfied: torch-cluster in /home/labuser/.local/lib/python3.10/
site-packages (1.6.3+pt23cpu)
Requirement already satisfied: torch-spline-conv in /home/labuser/.local/lib/python
3.10/site-packages (1.2.2+pt23cpu)
Requirement already satisfied: scipy in /home/labuser/.local/lib/python3.10/site-pac
kages (from torch-sparse) (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in /home/labuser/.local/lib/python
3.10/site-packages (from scipy->torch-sparse) (1.26.4)
```

In [ ]:
```
!pip install torch-geometric
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torch-geometric in /home/labuser/.local/lib/python3.1
0/site-packages (2.5.3)
Requirement already satisfied: tqdm in /home/labuser/.local/lib/python3.10/site-pack
ages (from torch-geometric) (4.66.4)
Requirement already satisfied: aiohttp in /home/labuser/.local/lib/python3.10/site-p
ackages (from torch-geometric) (3.9.5)
Requirement already satisfied: pyparsing in /usr/lib/python3/dist-packages (from tor
ch-geometric) (2.4.7)
Requirement already satisfied: psutil>=5.8.0 in /home/labuser/.local/lib/python3.10/
site-packages (from torch-geometric) (5.9.8)
Requirement already satisfied: fsspec in /home/labuser/.local/lib/python3.10/site-pa
ckages (from torch-geometric) (2024.6.0)
Requirement already satisfied: scipy in /home/labuser/.local/lib/python3.10/site-pac
kages (from torch-geometric) (1.13.1)
Requirement already satisfied: scikit-learn in /home/labuser/.local/lib/python3.10/s
ite-packages (from torch-geometric) (1.5.0)
Requirement already satisfied: jinja2 in /usr/lib/python3/dist-packages (from torch-
geometric) (3.0.3)
Requirement already satisfied: requests in /home/labuser/.local/lib/python3.10/site-
packages (from torch-geometric) (2.32.3)
Requirement already satisfied: numpy in /home/labuser/.local/lib/python3.10/site-pac
kages (from torch-geometric) (1.26.4)
Requirement already satisfied: frozenlist>=1.1.1 in /home/labuser/.local/lib/python
3.10/site-packages (from aiohttp->torch-geometric) (1.4.1)
Requirement already satisfied: yarl<2.0,>=1.0 in /home/labuser/.local/lib/python3.1
0/site-packages (from aiohttp->torch-geometric) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /home/labuser/.local/lib/p
ython3.10/site-packages (from aiohttp->torch-geometric) (4.0.3)
Requirement already satisfied: aiosignal>=1.1.2 in /home/labuser/.local/lib/python3.
10/site-packages (from aiohttp->torch-geometric) (1.3.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /home/labuser/.local/lib/pytho
n3.10/site-packages (from aiohttp->torch-geometric) (6.0.5)
Requirement already satisfied: attrs>=17.3.0 in /home/labuser/.local/lib/python3.10/
site-packages (from aiohttp->torch-geometric) (23.2.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/labuser/.local/lib/
python3.10/site-packages (from requests->torch-geometric) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages (from
requests->torch-geometric) (3.3)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/lib/python3/dist-packages
(from requests->torch-geometric) (1.26.5)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3/dist-packages
(from requests->torch-geometric) (2020.6.20)
Requirement already satisfied: threadpoolctl>=3.1.0 in /home/labuser/.local/lib/pyth
on3.10/site-packages (from scikit-learn->torch-geometric) (3.5.0)
Requirement already satisfied: joblib>=1.2.0 in /home/labuser/.local/lib/python3.10/
site-packages (from scikit-learn->torch-geometric) (1.4.2)
```

In [ ]:
```
!pip install ogb
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: ogb in /home/labuser/.local/lib/python3.10/site-packa
ges (1.3.6)
Requirement already satisfied: urllib3>=1.24.0 in /usr/lib/python3/dist-packages (fr
om ogb) (1.26.5)
Requirement already satisfied: torch>=1.6.0 in /home/labuser/.local/lib/python3.10/s
ite-packages (from ogb) (2.3.1+cpu)
Requirement already satisfied: scikit-learn>=0.20.0 in /home/labuser/.local/lib/pyth
on3.10/site-packages (from ogb) (1.5.0)
Requirement already satisfied: pandas>=0.24.0 in /home/labuser/.local/lib/python3.1
0/site-packages (from ogb) (2.2.2)
Requirement already satisfied: six>=1.12.0 in /usr/lib/python3/dist-packages (from o
gb) (1.16.0)
Requirement already satisfied: tqdm>=4.29.0 in /home/labuser/.local/lib/python3.10/s
ite-packages (from ogb) (4.66.4)
Requirement already satisfied: numpy>=1.16.0 in /home/labuser/.local/lib/python3.10/
site-packages (from ogb) (1.26.4)
Requirement already satisfied: outdated>=0.2.0 in /home/labuser/.local/lib/python3.1
0/site-packages (from ogb) (0.2.2)
Requirement already satisfied: setuptools>=44 in /usr/lib/python3/dist-packages (fro
m outdated>=0.2.0->ogb) (59.6.0)
Requirement already satisfied: littleutils in /home/labuser/.local/lib/python3.10/si
te-packages (from outdated>=0.2.0->ogb) (0.2.2)
Requirement already satisfied: requests in /home/labuser/.local/lib/python3.10/site-
packages (from outdated>=0.2.0->ogb) (2.32.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /home/labuser/.local/lib/py
thon3.10/site-packages (from pandas>=0.24.0->ogb) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /home/labuser/.local/lib/python3.1
0/site-packages (from pandas>=0.24.0->ogb) (2024.1)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages (from
pandas>=0.24.0->ogb) (2022.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /home/labuser/.local/lib/pyth
on3.10/site-packages (from scikit-learn>=0.20.0->ogb) (3.5.0)
Requirement already satisfied: scipy>=1.6.0 in /home/labuser/.local/lib/python3.10/s
ite-packages (from scikit-learn>=0.20.0->ogb) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /home/labuser/.local/lib/python3.10/
site-packages (from scikit-learn>=0.20.0->ogb) (1.4.2)
Requirement already satisfied: jinja2 in /usr/lib/python3/dist-packages (from torch>
=1.6.0->ogb) (3.0.3)
Requirement already satisfied: sympy in /home/labuser/.local/lib/python3.10/site-pac
kages (from torch>=1.6.0->ogb) (1.12)
Requirement already satisfied: fsspec in /home/labuser/.local/lib/python3.10/site-pa
ckages (from torch>=1.6.0->ogb) (2024.6.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /home/labuser/.local/lib/
python3.10/site-packages (from torch>=1.6.0->ogb) (4.12.2)
Requirement already satisfied: networkx in /home/labuser/.local/lib/python3.10/site-
packages (from torch>=1.6.0->ogb) (3.3)
Requirement already satisfied: filelock in /home/labuser/.local/lib/python3.10/site-
packages (from torch>=1.6.0->ogb) (3.13.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3/dist-packages
(from requests->outdated>=0.2.0->ogb) (2020.6.20)
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages (from
requests->outdated>=0.2.0->ogb) (3.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/labuser/.local/lib/
python3.10/site-packages (from requests->outdated>=0.2.0->ogb) (3.3.2)
```

```
Requirement already satisfied: mpmath>=0.19 in /home/labuser/.local/lib/python3.10/s
ite-packages (from sympy->torch>=1.6.0->ogb) (1.3.0)
```

In [ ]:
```
!pip install python-igraph
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting python-igraph
  Downloading python_igraph-0.11.8-py3-none-any.whl (9.1 kB)
Collecting igraph==0.11.8
  Downloading igraph-0.11.8-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(3.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 8.3 MB/s eta 0:00:000m eta
0:00:01[36m0:00:01
Collecting texttable>=1.6.2
  Downloading texttable-1.7.0-py2.py3-none-any.whl (10 kB)
Installing collected packages: texttable, igraph, python-igraph
Successfully installed igraph-0.11.8 python-igraph-0.11.8 texttable-1.7.0
```

Overview of the Project

- We are using ogbn-arxiv dataset after multiple tries for ogbn-products dataset which was giving numerous problems while finding the strongly connected graphe from the parent graph for calculating diameter
- We tried loading the ogbn-product but after that it was killing the kernel multiple times while doing the other operations which restricted us from performing other operations thus we movde to ogbn-arxiv
- We also have given multiple subgraphs with different number of nodes to get a feel of the sub graph with different number of nodes to get some clarity

below are the required dependent libraries used for this project

In [ ]:
```
# Import PyTorch for tensor operations and neural network functionality
import torch
import torch.nn.functional as F  # Provides various loss functions and activation f

# Import GCNConv for Graph Convolutional Network (GCN) layers
from torch_geometric.nn import GCNConv

# Import Data structure from PyTorch Geometric for graph representation
from torch_geometric.data import Data

# Import dataset and evaluator for graph-related tasks
from ogb.nodeproppred import PygNodePropPredDataset, Evaluator  # OGB datasets and

# Import approximation algorithms for graph analysis
from networkx.algorithms import approximation

# Import NetworkX for graph manipulation and visualization
import networkx as nx
```

```
# Utility to convert a PyTorch Geometric graph to a NetworkX graph
from torch_geometric.utils import to_networkx

# Import random for generating random samples or random node selections
import random

# Import Graph from the iGraph library for advanced graph analytics
from igraph import Graph

# Import Matplotlib for graph plotting and visualization
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
Matplotlib is building the font cache; this may take a moment.
```

Part 1 - Use the given dataset to design GNN based model for Node Classification as per details given below:

In [ ]:
```
# Load the dataset
dataset = PygNodePropPredDataset(name="ogbn-arxiv")

# Extract graph and labels
graph_data = dataset[0]

split_idx = dataset.get_idx_split()
evaluator = Evaluator(name="ogbn-arxiv")

# Move data to the GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
graph_data = graph_data.to(device)

#print the graph data details
graph_data
```

```
Downloading http://snap.stanford.edu/ogb/data/nodeproppred/arxiv.zip
Downloaded 0.08 GB: 100%|██████████████████████| 81/81 [00:10<00:00,  7.91it/s]
Extracting dataset/arxiv.zip
Processing...
Loading necessary files...
This might take a while.
Processing graphs...
100%|█████████████████████████████████████████| 1/1 [00:00<00:00, 4148.67it/s]
Converting graphs into PyG objects...
100%|██████████████████████████████████████████| 1/1 [00:00<00:00, 798.92it/s]
Saving...
Done!
```

Out[ ]:
```
Data(num_nodes=169343, edge_index=[2, 1166243], x=[169343, 128], node_year=[16934
3, 1], y=[169343, 1])
```

In [ ]:
```
# Define a Graph Convolutional Network (GCN) model for node classification
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)  # Non-linearity
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)  # Log-softmax for multi-class classificatio
```

In [ ]:
```
# Initialize the GCN model with input, hidden, and output layers; move it to the se
model = GCN(
    in_channels=graph_data.num_features,  # Input features (128-dimensional vectors
    hidden_channels=128,             # Number of hidden units
    out_channels=dataset.num_classes  # Number of classes (40 for ogbn-arxiv)
).to(device)

# Define the optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
```

Part 2 - Use https://ogb.stanford.edu/docs/home/Links to an external site. to learn dataset loading and checking performance method.

In [ ]:
```
# Define the training loop for the GCN model, including forward pass, loss calculat
def train():
    model.train()
    optimizer.zero_grad()
    out = model(graph_data.x, graph_data.edge_index)
    loss = F.nll_loss(out[split_idx['train']], graph_data.y[split_idx['train']].squ
    loss.backward()
    optimizer.step()
    return loss.item()
```

In [ ]:
```
# Define the evaluation function to compute accuracy on train, validation, and test
@torch.no_grad()
def evaluate():
    model.eval()
    out = model(graph_data.x, graph_data.edge_index)
    y_pred = out.argmax(dim=1).unsqueeze(1)

    #print("Shape of y_true:", data.y[split_idx['train']].shape)
    #print("Shape of y_pred:", y_pred[split_idx['train']].shape)

    train_acc = evaluator.eval({
        'y_true': graph_data.y[split_idx['train']],
        'y_pred': y_pred[split_idx['train']],
    })['acc']
    val_acc = evaluator.eval({
        'y_true': graph_data.y[split_idx['valid']],
        'y_pred': y_pred[split_idx['valid']],
    })['acc']

    test_acc = evaluator.eval({
        'y_true': graph_data.y[split_idx['test']],
        'y_pred': y_pred[split_idx['test']],
    })['acc']

    return train_acc, val_acc, test_acc
```

The below part is doing the training with multiple epochs to get the maximum accuracy

In [ ]:

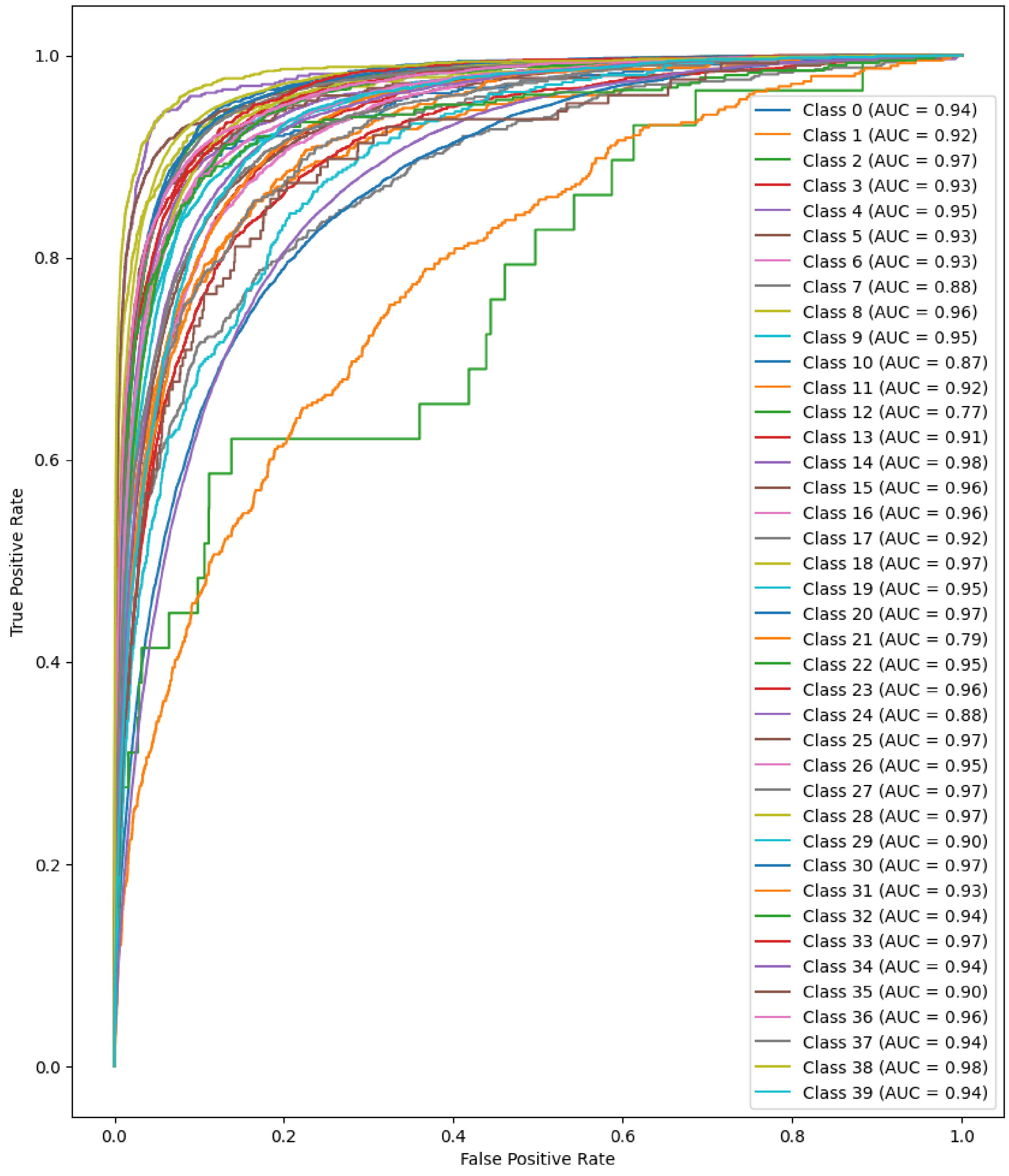
```
# Train the model over multiple epochs and evaluate its performance on training, va
# Track and store the best validation accuracy and the corresponding test accuracy
best_val_acc = 0
for epoch in range(1, 201):  # Train for 100 epochs
    loss = train()
    train_acc, val_acc, test_acc = evaluate()

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        best_test_acc = test_acc

    print(
        f"Epoch {epoch:03d}, Loss: {loss:.4f}, Train Acc: {train_acc:.4f}, "
        f"Val Acc: {val_acc:.4f}, Test Acc: {test_acc:.4f}"
    )

# Print the best validation and corresponding test accuracy achieved by the model.
print(f"Best Validation Accuracy: {best_val_acc:.4f}")
print(f"Corresponding Test Accuracy: {best_test_acc:.4f}")
```

```
Epoch 001, Loss: 3.7527, Train Acc: 0.2079, Val Acc: 0.2684, Test Acc: 0.2352
Epoch 002, Loss: 3.3106, Train Acc: 0.2652, Val Acc: 0.2595, Test Acc: 0.2192
Epoch 003, Loss: 3.0977, Train Acc: 0.2198, Val Acc: 0.2071, Test Acc: 0.2609
Epoch 004, Loss: 3.0551, Train Acc: 0.3011, Val Acc: 0.3189, Test Acc: 0.2936
Epoch 005, Loss: 2.9482, Train Acc: 0.2896, Val Acc: 0.3022, Test Acc: 0.2657
Epoch 006, Loss: 2.8853, Train Acc: 0.2839, Val Acc: 0.2977, Test Acc: 0.2597
Epoch 007, Loss: 2.8389, Train Acc: 0.2939, Val Acc: 0.3066, Test Acc: 0.2735
Epoch 008, Loss: 2.7522, Train Acc: 0.3453, Val Acc: 0.3923, Test Acc: 0.3583
Epoch 009, Loss: 2.6686, Train Acc: 0.3617, Val Acc: 0.4077, Test Acc: 0.4005
Epoch 010, Loss: 2.6176, Train Acc: 0.3837, Val Acc: 0.4281, Test Acc: 0.4139
Epoch 011, Loss: 2.5537, Train Acc: 0.3982, Val Acc: 0.4401, Test Acc: 0.4039
Epoch 012, Loss: 2.4879, Train Acc: 0.3934, Val Acc: 0.4171, Test Acc: 0.3737
Epoch 013, Loss: 2.4385, Train Acc: 0.4102, Val Acc: 0.4276, Test Acc: 0.3734
Epoch 014, Loss: 2.3814, Train Acc: 0.4369, Val Acc: 0.4533, Test Acc: 0.3908
Epoch 015, Loss: 2.3242, Train Acc: 0.4543, Val Acc: 0.4722, Test Acc: 0.4192
Epoch 016, Loss: 2.2796, Train Acc: 0.4681, Val Acc: 0.4844, Test Acc: 0.4374
Epoch 017, Loss: 2.2332, Train Acc: 0.4697, Val Acc: 0.4845, Test Acc: 0.4341
Epoch 018, Loss: 2.1852, Train Acc: 0.4658, Val Acc: 0.4768, Test Acc: 0.4239
Epoch 019, Loss: 2.1448, Train Acc: 0.4695, Val Acc: 0.4798, Test Acc: 0.4229
Epoch 020, Loss: 2.1059, Train Acc: 0.4802, Val Acc: 0.4865, Test Acc: 0.4308
Epoch 021, Loss: 2.0687, Train Acc: 0.4885, Val Acc: 0.4947, Test Acc: 0.4389
Epoch 022, Loss: 2.0367, Train Acc: 0.4961, Val Acc: 0.5014, Test Acc: 0.4448
Epoch 023, Loss: 2.0048, Train Acc: 0.4984, Val Acc: 0.5021, Test Acc: 0.4449
Epoch 024, Loss: 1.9745, Train Acc: 0.5005, Val Acc: 0.5019, Test Acc: 0.4435
Epoch 025, Loss: 1.9469, Train Acc: 0.5047, Val Acc: 0.5019, Test Acc: 0.4435
Epoch 026, Loss: 1.9211, Train Acc: 0.5085, Val Acc: 0.5042, Test Acc: 0.4446
Epoch 027, Loss: 1.8978, Train Acc: 0.5135, Val Acc: 0.5100, Test Acc: 0.4508
Epoch 028, Loss: 1.8728, Train Acc: 0.5184, Val Acc: 0.5161, Test Acc: 0.4570
Epoch 029, Loss: 1.8497, Train Acc: 0.5218, Val Acc: 0.5190, Test Acc: 0.4598
Epoch 030, Loss: 1.8298, Train Acc: 0.5264, Val Acc: 0.5185, Test Acc: 0.4573
Epoch 031, Loss: 1.8087, Train Acc: 0.5287, Val Acc: 0.5172, Test Acc: 0.4544
Epoch 032, Loss: 1.7893, Train Acc: 0.5313, Val Acc: 0.5204, Test Acc: 0.4589
Epoch 033, Loss: 1.7710, Train Acc: 0.5359, Val Acc: 0.5273, Test Acc: 0.4677
Epoch 034, Loss: 1.7537, Train Acc: 0.5388, Val Acc: 0.5294, Test Acc: 0.4698
Epoch 035, Loss: 1.7373, Train Acc: 0.5415, Val Acc: 0.5295, Test Acc: 0.4669
Epoch 036, Loss: 1.7213, Train Acc: 0.5427, Val Acc: 0.5290, Test Acc: 0.4670
Epoch 037, Loss: 1.7071, Train Acc: 0.5455, Val Acc: 0.5320, Test Acc: 0.4697
Epoch 038, Loss: 1.6929, Train Acc: 0.5484, Val Acc: 0.5349, Test Acc: 0.4726
Epoch 039, Loss: 1.6799, Train Acc: 0.5507, Val Acc: 0.5377, Test Acc: 0.4754
Epoch 040, Loss: 1.6679, Train Acc: 0.5518, Val Acc: 0.5383, Test Acc: 0.4761
Epoch 041, Loss: 1.6562, Train Acc: 0.5524, Val Acc: 0.5387, Test Acc: 0.4750
Epoch 042, Loss: 1.6460, Train Acc: 0.5539, Val Acc: 0.5395, Test Acc: 0.4780
Epoch 043, Loss: 1.6359, Train Acc: 0.5563, Val Acc: 0.5424, Test Acc: 0.4801
Epoch 044, Loss: 1.6262, Train Acc: 0.5584, Val Acc: 0.5452, Test Acc: 0.4829
Epoch 045, Loss: 1.6177, Train Acc: 0.5598, Val Acc: 0.5446, Test Acc: 0.4825
Epoch 046, Loss: 1.6090, Train Acc: 0.5611, Val Acc: 0.5448, Test Acc: 0.4815
Epoch 047, Loss: 1.6010, Train Acc: 0.5626, Val Acc: 0.5479, Test Acc: 0.4854
Epoch 048, Loss: 1.5936, Train Acc: 0.5648, Val Acc: 0.5492, Test Acc: 0.4870
Epoch 049, Loss: 1.5863, Train Acc: 0.5663, Val Acc: 0.5504, Test Acc: 0.4896
Epoch 050, Loss: 1.5796, Train Acc: 0.5676, Val Acc: 0.5501, Test Acc: 0.4850
Epoch 051, Loss: 1.5736, Train Acc: 0.5678, Val Acc: 0.5544, Test Acc: 0.4935
Epoch 052, Loss: 1.5681, Train Acc: 0.5699, Val Acc: 0.5514, Test Acc: 0.4868
Epoch 053, Loss: 1.5621, Train Acc: 0.5714, Val Acc: 0.5551, Test Acc: 0.4915
Epoch 054, Loss: 1.5558, Train Acc: 0.5724, Val Acc: 0.5572, Test Acc: 0.4940
Epoch 055, Loss: 1.5508, Train Acc: 0.5734, Val Acc: 0.5567, Test Acc: 0.4903
Epoch 056, Loss: 1.5464, Train Acc: 0.5746, Val Acc: 0.5596, Test Acc: 0.4948
Epoch 057, Loss: 1.5413, Train Acc: 0.5755, Val Acc: 0.5596, Test Acc: 0.4941
Epoch 058, Loss: 1.5365, Train Acc: 0.5759, Val Acc: 0.5603, Test Acc: 0.4933
Epoch 059, Loss: 1.5327, Train Acc: 0.5773, Val Acc: 0.5632, Test Acc: 0.4978
Epoch 060, Loss: 1.5287, Train Acc: 0.5784, Val Acc: 0.5613, Test Acc: 0.4939
Epoch 061, Loss: 1.5244, Train Acc: 0.5791, Val Acc: 0.5621, Test Acc: 0.4946
Epoch 062, Loss: 1.5209, Train Acc: 0.5798, Val Acc: 0.5650, Test Acc: 0.4990
Epoch 063, Loss: 1.5177, Train Acc: 0.5805, Val Acc: 0.5639, Test Acc: 0.4954
Epoch 064, Loss: 1.5139, Train Acc: 0.5812, Val Acc: 0.5636, Test Acc: 0.4959
Epoch 065, Loss: 1.5106, Train Acc: 0.5817, Val Acc: 0.5668, Test Acc: 0.5005
Epoch 066, Loss: 1.5079, Train Acc: 0.5825, Val Acc: 0.5651, Test Acc: 0.4973
Epoch 067, Loss: 1.5049, Train Acc: 0.5831, Val Acc: 0.5678, Test Acc: 0.4996
Epoch 068, Loss: 1.5018, Train Acc: 0.5840, Val Acc: 0.5682, Test Acc: 0.5002
Epoch 069, Loss: 1.4990, Train Acc: 0.5847, Val Acc: 0.5675, Test Acc: 0.4989
Epoch 070, Loss: 1.4966, Train Acc: 0.5851, Val Acc: 0.5691, Test Acc: 0.5017
Epoch 071, Loss: 1.4940, Train Acc: 0.5857, Val Acc: 0.5684, Test Acc: 0.5004
Epoch 072, Loss: 1.4913, Train Acc: 0.5865, Val Acc: 0.5693, Test Acc: 0.5018
Epoch 073, Loss: 1.4888, Train Acc: 0.5869, Val Acc: 0.5709, Test Acc: 0.5036
Epoch 074, Loss: 1.4866, Train Acc: 0.5875, Val Acc: 0.5699, Test Acc: 0.5015
Epoch 075, Loss: 1.4843, Train Acc: 0.5879, Val Acc: 0.5713, Test Acc: 0.5040
Epoch 076, Loss: 1.4820, Train Acc: 0.5884, Val Acc: 0.5712, Test Acc: 0.5030
Epoch 077, Loss: 1.4796, Train Acc: 0.5889, Val Acc: 0.5715, Test Acc: 0.5032
Epoch 078, Loss: 1.4774, Train Acc: 0.5892, Val Acc: 0.5730, Test Acc: 0.5052
Epoch 079, Loss: 1.4753, Train Acc: 0.5900, Val Acc: 0.5724, Test Acc: 0.5042
Epoch 080, Loss: 1.4733, Train Acc: 0.5903, Val Acc: 0.5732, Test Acc: 0.5052
Epoch 081, Loss: 1.4712, Train Acc: 0.5909, Val Acc: 0.5732, Test Acc: 0.5056
Epoch 082, Loss: 1.4691, Train Acc: 0.5910, Val Acc: 0.5734, Test Acc: 0.5054
Epoch 083, Loss: 1.4671, Train Acc: 0.5918, Val Acc: 0.5742, Test Acc: 0.5067
Epoch 084, Loss: 1.4651, Train Acc: 0.5922, Val Acc: 0.5735, Test Acc: 0.5059
Epoch 085, Loss: 1.4631, Train Acc: 0.5927, Val Acc: 0.5745, Test Acc: 0.5076
Epoch 086, Loss: 1.4613, Train Acc: 0.5936, Val Acc: 0.5732, Test Acc: 0.5058
Epoch 087, Loss: 1.4596, Train Acc: 0.5938, Val Acc: 0.5755, Test Acc: 0.5084
Epoch 088, Loss: 1.4579, Train Acc: 0.5942, Val Acc: 0.5738, Test Acc: 0.5062
Epoch 089, Loss: 1.4563, Train Acc: 0.5943, Val Acc: 0.5751, Test Acc: 0.5080
Epoch 090, Loss: 1.4548, Train Acc: 0.5953, Val Acc: 0.5745, Test Acc: 0.5078
Epoch 091, Loss: 1.4532, Train Acc: 0.5949, Val Acc: 0.5742, Test Acc: 0.5084
Epoch 092, Loss: 1.4514, Train Acc: 0.5954, Val Acc: 0.5753, Test Acc: 0.5087
Epoch 093, Loss: 1.4494, Train Acc: 0.5960, Val Acc: 0.5760, Test Acc: 0.5077
Epoch 094, Loss: 1.4478, Train Acc: 0.5961, Val Acc: 0.5763, Test Acc: 0.5103
Epoch 095, Loss: 1.4466, Train Acc: 0.5965, Val Acc: 0.5752, Test Acc: 0.5060
Epoch 096, Loss: 1.4455, Train Acc: 0.5967, Val Acc: 0.5776, Test Acc: 0.5120
Epoch 097, Loss: 1.4442, Train Acc: 0.5972, Val Acc: 0.5749, Test Acc: 0.5061
Epoch 098, Loss: 1.4424, Train Acc: 0.5975, Val Acc: 0.5775, Test Acc: 0.5114
Epoch 099, Loss: 1.4404, Train Acc: 0.5979, Val Acc: 0.5764, Test Acc: 0.5093
Epoch 100, Loss: 1.4388, Train Acc: 0.5982, Val Acc: 0.5769, Test Acc: 0.5089
Epoch 101, Loss: 1.4379, Train Acc: 0.5983, Val Acc: 0.5775, Test Acc: 0.5118
Epoch 102, Loss: 1.4371, Train Acc: 0.5987, Val Acc: 0.5774, Test Acc: 0.5088
Epoch 103, Loss: 1.4358, Train Acc: 0.5991, Val Acc: 0.5773, Test Acc: 0.5105
Epoch 104, Loss: 1.4342, Train Acc: 0.5994, Val Acc: 0.5794, Test Acc: 0.5120
Epoch 105, Loss: 1.4325, Train Acc: 0.5994, Val Acc: 0.5778, Test Acc: 0.5088
Epoch 106, Loss: 1.4314, Train Acc: 0.6000, Val Acc: 0.5796, Test Acc: 0.5137
Epoch 107, Loss: 1.4305, Train Acc: 0.6002, Val Acc: 0.5782, Test Acc: 0.5089
Epoch 108, Loss: 1.4296, Train Acc: 0.6000, Val Acc: 0.5795, Test Acc: 0.5126
Epoch 109, Loss: 1.4282, Train Acc: 0.6009, Val Acc: 0.5788, Test Acc: 0.5108
Epoch 110, Loss: 1.4266, Train Acc: 0.6011, Val Acc: 0.5798, Test Acc: 0.5126
Epoch 111, Loss: 1.4254, Train Acc: 0.6008, Val Acc: 0.5793, Test Acc: 0.5105
Epoch 112, Loss: 1.4248, Train Acc: 0.6022, Val Acc: 0.5818, Test Acc: 0.5129
Epoch 113, Loss: 1.4246, Train Acc: 0.6008, Val Acc: 0.5786, Test Acc: 0.5106
Epoch 114, Loss: 1.4239, Train Acc: 0.6018, Val Acc: 0.5801, Test Acc: 0.5122
Epoch 115, Loss: 1.4226, Train Acc: 0.6026, Val Acc: 0.5812, Test Acc: 0.5148
Epoch 116, Loss: 1.4216, Train Acc: 0.6017, Val Acc: 0.5775, Test Acc: 0.5092
Epoch 117, Loss: 1.4206, Train Acc: 0.6030, Val Acc: 0.5830, Test Acc: 0.5162
Epoch 118, Loss: 1.4192, Train Acc: 0.6028, Val Acc: 0.5805, Test Acc: 0.5127
Epoch 119, Loss: 1.4180, Train Acc: 0.6025, Val Acc: 0.5795, Test Acc: 0.5108
Epoch 120, Loss: 1.4177, Train Acc: 0.6034, Val Acc: 0.5829, Test Acc: 0.5163
Epoch 121, Loss: 1.4170, Train Acc: 0.6032, Val Acc: 0.5797, Test Acc: 0.5118
Epoch 122, Loss: 1.4159, Train Acc: 0.6034, Val Acc: 0.5822, Test Acc: 0.5137
Epoch 123, Loss: 1.4146, Train Acc: 0.6038, Val Acc: 0.5830, Test Acc: 0.5160
Epoch 124, Loss: 1.4139, Train Acc: 0.6034, Val Acc: 0.5793, Test Acc: 0.5112
Epoch 125, Loss: 1.4136, Train Acc: 0.6042, Val Acc: 0.5832, Test Acc: 0.5168
Epoch 126, Loss: 1.4125, Train Acc: 0.6044, Val Acc: 0.5823, Test Acc: 0.5143
Epoch 127, Loss: 1.4115, Train Acc: 0.6040, Val Acc: 0.5802, Test Acc: 0.5130
Epoch 128, Loss: 1.4109, Train Acc: 0.6044, Val Acc: 0.5843, Test Acc: 0.5173
Epoch 129, Loss: 1.4103, Train Acc: 0.6048, Val Acc: 0.5816, Test Acc: 0.5132
Epoch 130, Loss: 1.4093, Train Acc: 0.6046, Val Acc: 0.5815, Test Acc: 0.5147
Epoch 131, Loss: 1.4086, Train Acc: 0.6048, Val Acc: 0.5835, Test Acc: 0.5156
Epoch 132, Loss: 1.4082, Train Acc: 0.6050, Val Acc: 0.5815, Test Acc: 0.5149
Epoch 133, Loss: 1.4076, Train Acc: 0.6048, Val Acc: 0.5818, Test Acc: 0.5134
Epoch 134, Loss: 1.4068, Train Acc: 0.6052, Val Acc: 0.5840, Test Acc: 0.5181
Epoch 135, Loss: 1.4064, Train Acc: 0.6045, Val Acc: 0.5787, Test Acc: 0.5102
Epoch 136, Loss: 1.4069, Train Acc: 0.6048, Val Acc: 0.5856, Test Acc: 0.5228
Epoch 137, Loss: 1.4086, Train Acc: 0.6038, Val Acc: 0.5758, Test Acc: 0.5078
Epoch 138, Loss: 1.4105, Train Acc: 0.6030, Val Acc: 0.5837, Test Acc: 0.5216
Epoch 139, Loss: 1.4110, Train Acc: 0.6054, Val Acc: 0.5818, Test Acc: 0.5126
Epoch 140, Loss: 1.4061, Train Acc: 0.6064, Val Acc: 0.5824, Test Acc: 0.5151
Epoch 141, Loss: 1.4025, Train Acc: 0.6051, Val Acc: 0.5844, Test Acc: 0.5193
Epoch 142, Loss: 1.4041, Train Acc: 0.6048, Val Acc: 0.5802, Test Acc: 0.5113
Epoch 143, Loss: 1.4061, Train Acc: 0.6061, Val Acc: 0.5848, Test Acc: 0.5200
Epoch 144, Loss: 1.4031, Train Acc: 0.6064, Val Acc: 0.5829, Test Acc: 0.5150
Epoch 145, Loss: 1.4002, Train Acc: 0.6057, Val Acc: 0.5820, Test Acc: 0.5139
Epoch 146, Loss: 1.4018, Train Acc: 0.6058, Val Acc: 0.5852, Test Acc: 0.5209
Epoch 147, Loss: 1.4023, Train Acc: 0.6062, Val Acc: 0.5820, Test Acc: 0.5138
Epoch 148, Loss: 1.3995, Train Acc: 0.6071, Val Acc: 0.5855, Test Acc: 0.5183
Epoch 149, Loss: 1.3986, Train Acc: 0.6066, Val Acc: 0.5838, Test Acc: 0.5185
Epoch 150, Loss: 1.3995, Train Acc: 0.6066, Val Acc: 0.5812, Test Acc: 0.5131
Epoch 151, Loss: 1.3986, Train Acc: 0.6073, Val Acc: 0.5858, Test Acc: 0.5214
Epoch 152, Loss: 1.3972, Train Acc: 0.6072, Val Acc: 0.5833, Test Acc: 0.5166
Epoch 153, Loss: 1.3968, Train Acc: 0.6071, Val Acc: 0.5821, Test Acc: 0.5143
Epoch 154, Loss: 1.3967, Train Acc: 0.6076, Val Acc: 0.5861, Test Acc: 0.5219
Epoch 155, Loss: 1.3960, Train Acc: 0.6071, Val Acc: 0.5821, Test Acc: 0.5158
Epoch 156, Loss: 1.3951, Train Acc: 0.6074, Val Acc: 0.5850, Test Acc: 0.5171
Epoch 157, Loss: 1.3946, Train Acc: 0.6078, Val Acc: 0.5853, Test Acc: 0.5204
Epoch 158, Loss: 1.3943, Train Acc: 0.6079, Val Acc: 0.5820, Test Acc: 0.5151
Epoch 159, Loss: 1.3938, Train Acc: 0.6077, Val Acc: 0.5858, Test Acc: 0.5200
Epoch 160, Loss: 1.3931, Train Acc: 0.6079, Val Acc: 0.5850, Test Acc: 0.5185
Epoch 161, Loss: 1.3925, Train Acc: 0.6081, Val Acc: 0.5824, Test Acc: 0.5153
Epoch 162, Loss: 1.3923, Train Acc: 0.6081, Val Acc: 0.5867, Test Acc: 0.5216
Epoch 163, Loss: 1.3920, Train Acc: 0.6081, Val Acc: 0.5834, Test Acc: 0.5171
Epoch 164, Loss: 1.3911, Train Acc: 0.6086, Val Acc: 0.5843, Test Acc: 0.5177
Epoch 165, Loss: 1.3906, Train Acc: 0.6083, Val Acc: 0.5861, Test Acc: 0.5214
Epoch 166, Loss: 1.3905, Train Acc: 0.6082, Val Acc: 0.5830, Test Acc: 0.5166
Epoch 167, Loss: 1.3901, Train Acc: 0.6085, Val Acc: 0.5856, Test Acc: 0.5204
Epoch 168, Loss: 1.3893, Train Acc: 0.6088, Val Acc: 0.5854, Test Acc: 0.5196
Epoch 169, Loss: 1.3888, Train Acc: 0.6084, Val Acc: 0.5836, Test Acc: 0.5174
Epoch 170, Loss: 1.3886, Train Acc: 0.6087, Val Acc: 0.5862, Test Acc: 0.5217
Epoch 171, Loss: 1.3882, Train Acc: 0.6089, Val Acc: 0.5842, Test Acc: 0.5180
Epoch 172, Loss: 1.3877, Train Acc: 0.6087, Val Acc: 0.5849, Test Acc: 0.5200
Epoch 173, Loss: 1.3872, Train Acc: 0.6091, Val Acc: 0.5858, Test Acc: 0.5206
Epoch 174, Loss: 1.3868, Train Acc: 0.6090, Val Acc: 0.5840, Test Acc: 0.5189
Epoch 175, Loss: 1.3864, Train Acc: 0.6092, Val Acc: 0.5858, Test Acc: 0.5202
Epoch 176, Loss: 1.3860, Train Acc: 0.6091, Val Acc: 0.5856, Test Acc: 0.5196
Epoch 177, Loss: 1.3856, Train Acc: 0.6092, Val Acc: 0.5855, Test Acc: 0.5201
Epoch 178, Loss: 1.3853, Train Acc: 0.6094, Val Acc: 0.5859, Test Acc: 0.5203
Epoch 179, Loss: 1.3849, Train Acc: 0.6097, Val Acc: 0.5854, Test Acc: 0.5197
Epoch 180, Loss: 1.3845, Train Acc: 0.6094, Val Acc: 0.5851, Test Acc: 0.5204
Epoch 181, Loss: 1.3842, Train Acc: 0.6094, Val Acc: 0.5861, Test Acc: 0.5195
Epoch 182, Loss: 1.3842, Train Acc: 0.6095, Val Acc: 0.5858, Test Acc: 0.5212
Epoch 183, Loss: 1.3842, Train Acc: 0.6093, Val Acc: 0.5846, Test Acc: 0.5167
Epoch 184, Loss: 1.3843, Train Acc: 0.6094, Val Acc: 0.5877, Test Acc: 0.5242
Epoch 185, Loss: 1.3844, Train Acc: 0.6097, Val Acc: 0.5829, Test Acc: 0.5154
Epoch 186, Loss: 1.3840, Train Acc: 0.6096, Val Acc: 0.5881, Test Acc: 0.5249
Epoch 187, Loss: 1.3832, Train Acc: 0.6099, Val Acc: 0.5834, Test Acc: 0.5172
Epoch 188, Loss: 1.3819, Train Acc: 0.6102, Val Acc: 0.5865, Test Acc: 0.5238
Epoch 189, Loss: 1.3811, Train Acc: 0.6100, Val Acc: 0.5860, Test Acc: 0.5199
Epoch 190, Loss: 1.3809, Train Acc: 0.6103, Val Acc: 0.5859, Test Acc: 0.5209
Epoch 191, Loss: 1.3812, Train Acc: 0.6097, Val Acc: 0.5862, Test Acc: 0.5211
Epoch 192, Loss: 1.3818, Train Acc: 0.6095, Val Acc: 0.5869, Test Acc: 0.5211
Epoch 193, Loss: 1.3817, Train Acc: 0.6099, Val Acc: 0.5843, Test Acc: 0.5195
Epoch 194, Loss: 1.3811, Train Acc: 0.6103, Val Acc: 0.5876, Test Acc: 0.5242
Epoch 195, Loss: 1.3801, Train Acc: 0.6104, Val Acc: 0.5844, Test Acc: 0.5178
Epoch 196, Loss: 1.3792, Train Acc: 0.6105, Val Acc: 0.5879, Test Acc: 0.5264
Epoch 197, Loss: 1.3788, Train Acc: 0.6108, Val Acc: 0.5836, Test Acc: 0.5169
Epoch 198, Loss: 1.3788, Train Acc: 0.6101, Val Acc: 0.5879, Test Acc: 0.5254
Epoch 199, Loss: 1.3786, Train Acc: 0.6109, Val Acc: 0.5851, Test Acc: 0.5186
Epoch 200, Loss: 1.3777, Train Acc: 0.6108, Val Acc: 0.5865, Test Acc: 0.5231
Best Validation Accuracy: 0.5881
Corresponding Test Accuracy: 0.5249
```

In [ ]:

```python
# Define a function to plot the ROC curve for all classes in a multi-class classifi
# Uses the trained model to compute probabilities and evaluates performance using A
@torch.no_grad()
def plot_roc():
    model.eval()
    out = model(graph_data.x, graph_data.edge_index)
    y_prob = F.softmax(out, dim=1).cpu().numpy()
    y_true = graph_data.y.cpu().numpy()

    plt.figure(figsize=(10, 12))

    for class_idx in range(dataset.num_classes):
        fpr, tpr, _ = roc_curve(y_true == class_idx, y_prob[:, class_idx])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"Class {class_idx} (AUC = {roc_auc:.2f})")

    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()
```

```
plot_roc()
```



Part 3.1 Diameter , number of nodes and edges , Global Clustering Coefficient of existing graph

In [ ]:
```
# Extract edge_index and node features
edge_index = graph_data.edge_index  # Edge indices
node_features = graph_data.x        # Node features
labels = graph_data.y               # Node labels
```

```
# Display information
print(f"Number of nodes: {node_features.size(0)}")
print(f"Number of edges: {edge_index.size(1)}")
print(f"Number of features per node: {node_features.size(1)}")
print(f"Number of classes: {labels.max().item() + 1}")
```

```
Number of nodes: 169343
Number of edges: 1166243
Number of features per node: 128
Number of classes: 40
```

In [ ]:
```
# Convert the graph data to a NetworkX graph object
# The resulting graph will be used for further analysis or visualization
G = to_networkx(graph_data)
```

In [ ]:
```
# Step 3: Calculate the number of nodes and edges
num_nodes = G.number_of_nodes()  # Total nodes in the graph
num_edges = G.number_of_edges()  # Total edges in the graph

print(f"Number of nodes: {num_nodes}")
print(f"Number of edges: {num_edges}")
```

```
Number of nodes: 169343
Number of edges: 1166243
```

As the number of nodes and edges are large, 2 different approximation methods have been used to calculate the Diameter of the graph

- The largest strongly connected graph has been calculated from the parent dataset graph and the diameter has been calculated on that sub graph

Approximation method 1

In [ ]:
```
# Diameter is the longest shortest path in the graph. We take the largest strongly
largest_scc = max(nx.strongly_connected_components(G), key=len)  # Largest SCC
print("len of largest_scc: ", len(largest_scc))
G_largest_scc = G.subgraph(largest_scc)  # Subgraph of the largest SCC
print("G_largest_scc: ", G_largest_scc)

approx_diameter = nx.algorithms.approximation.diameter(G_largest_scc)

# Approximate diameter
print(f"Approximate diameter (iGraph): {approx_diameter}")
```

```
len of largest_scc:  23164
G_largest_scc:  DiGraph with 23164 nodes and 223310 edges
Approximate diameter (iGraph): 62
```

Approximation method 2

In [ ]:
```
# Sample a subset of nodes from the largest SCC
sample_nodes = random.sample(list(G_largest_scc.nodes()), k=2000)  # Adjust 'k' bas

# Compute eccentricities for the sampled nodes
eccentricities = nx.eccentricity(G_largest_scc, v=sample_nodes)

# Approximate diameter
approx_diameter = max(eccentricities.values())
print(f"Approximate diameter (sampled nodes): {approx_diameter}")
```

```
Approximate diameter (sampled nodes): 61
```

Global Clustering Coefficient of existing graph

In [ ]:
```
# Step 5: Compute the global clustering coefficient
# Clustering coefficient measures the likelihood of nodes forming triangles (local
clustering_coefficient = nx.average_clustering(G.to_undirected())  # Global cluster
print(f"Global Clustering Coefficient: {clustering_coefficient:.6f}")
```

```
Global Clustering Coefficient: 0.226129
```

Part 3.2 Plot the graph with label

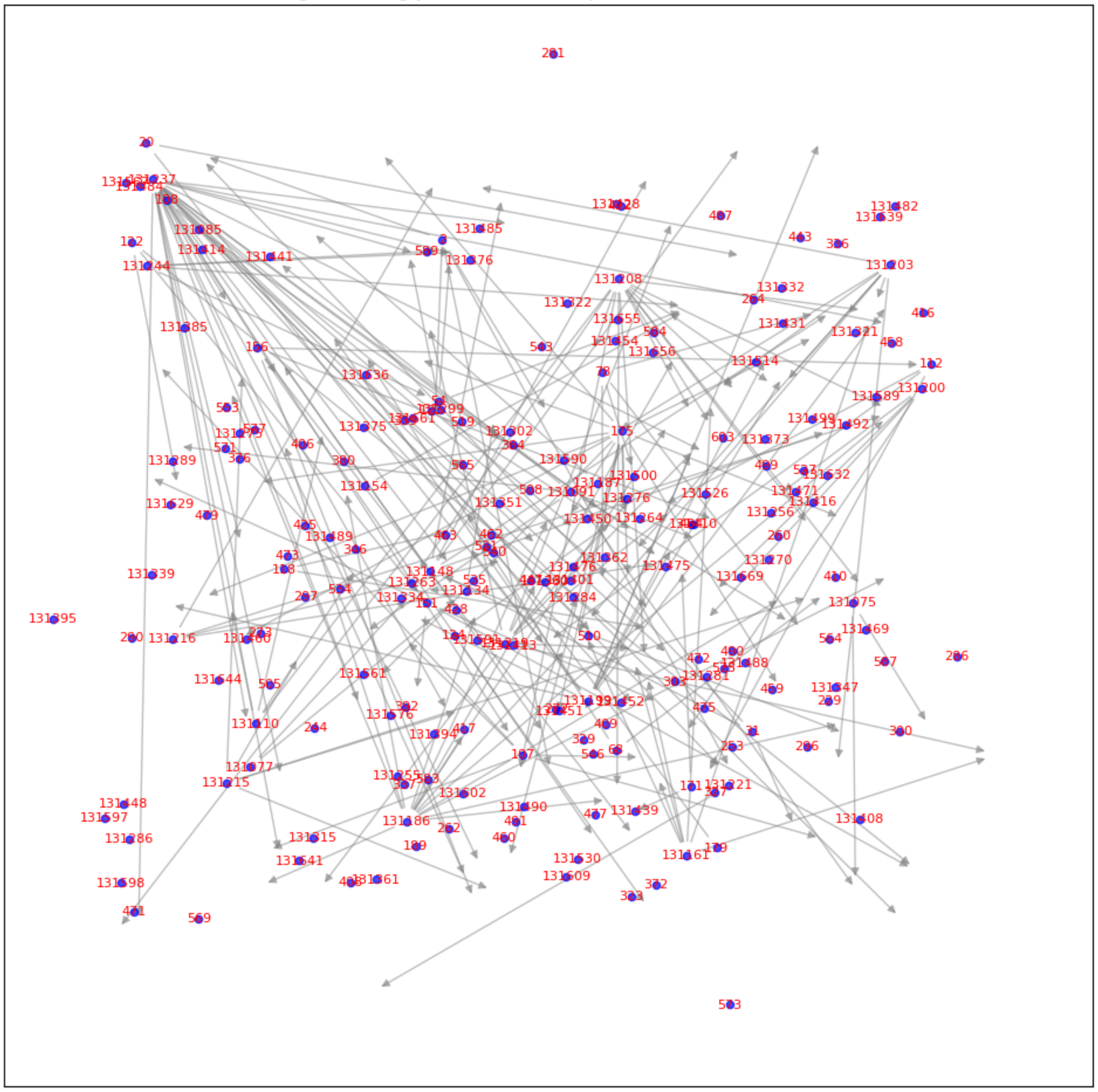
- As the derived sub graphe is having 23164 nodes and 223310 edges, we cant draw the complete graph, we have taken a portion of the graph and drawn
- Taken first 200, 300 and 500 nodes and their edges from the derived largest strongly connected sub graph and drawn to get an idea of the look and feel of the sub graph
- we used this method just to give an look and feel of the main graph, as in section 3.6 below we are using induced node method to draw the graph, we used a random selection method of selecting nodes in this part.

Subgraph with first 200 nodes and edges from the largest subgraph derived from above

In [ ]:
```
# Draw the graph
plt.figure(figsize=(12, 12))  # Set the figure size
pos = nx.spring_layout(G_largest_scc, seed=42, iterations=2)  # Compute positions u
print("G_largest_scc: ", G_largest_scc)

# Draw nodes with labels
nodes_to_draw = list(G_largest_scc.nodes)[:200]
subgraph = G_largest_scc.subgraph(nodes_to_draw)
nx.draw_networkx_nodes(G_largest_scc, pos, nodelist=nodes_to_draw, node_size=20, no

edges_to_draw = list(G_largest_scc.edges)[:200]  # Draw only 1000 edges
nx.draw_networkx_edges(G_largest_scc, pos, edgelist=edges_to_draw, edge_color="gray

nx.draw_networkx_labels(
    G_largest_scc,
    pos,
    labels={node: str(node) for node in list(G_largest_scc.nodes)[:200]},  # Displa
    font_size=8,
    font_color="red"
)
#print("G_largest_scc: ", G_largest_scc)
plt.title("Largest Strongly Connected Component with Node Labels")
plt.show()
```

```
G_largest_scc:  DiGraph with 23164 nodes and 223310 edges
```

Largest Strongly Connected Component with Node Labels



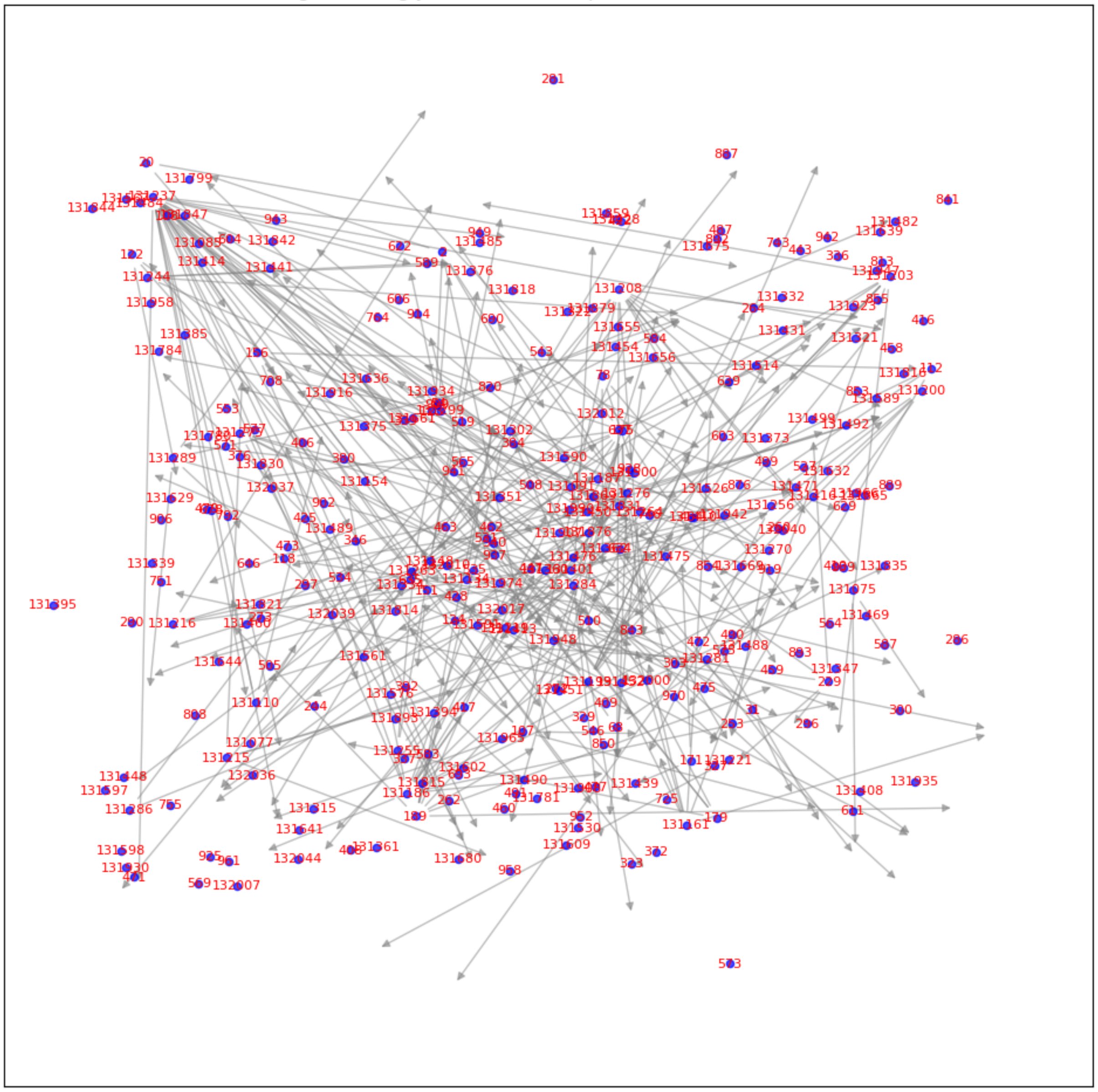
Subgraph with first 300 nodes and edges from the largest subgraph derived from above

In [ ]:
```
# Draw the graph
plt.figure(figsize=(12, 12))  # Set the figure size
#pos = nx.spring_layout(G_largest_scc, seed=42, iterations=2)  # Compute positions
print("G_largest_scc: ", G_largest_scc)
```

```
# Draw nodes with labels
nodes_to_draw = list(G_largest_scc.nodes)[:300]
subgraph = G_largest_scc.subgraph(nodes_to_draw)
nx.draw_networkx_nodes(G_largest_scc, pos, nodelist=nodes_to_draw, node_size=20, no

edges_to_draw = list(G_largest_scc.edges)[:300]  # Draw only 1000 edges
nx.draw_networkx_edges(G_largest_scc, pos, edgelist=edges_to_draw, edge_color="gray

nx.draw_networkx_labels(
    G_largest_scc,
    pos,
    labels={node: str(node) for node in list(G_largest_scc.nodes)[:300]},  # Displa
    font_size=8,
    font_color="red"
)

#print("G_largest_scc: ", G_largest_scc)
plt.title("Largest Strongly Connected Component with Node Labels")
plt.show()
```

```
G_largest_scc:  DiGraph with 23164 nodes and 223310 edges
```

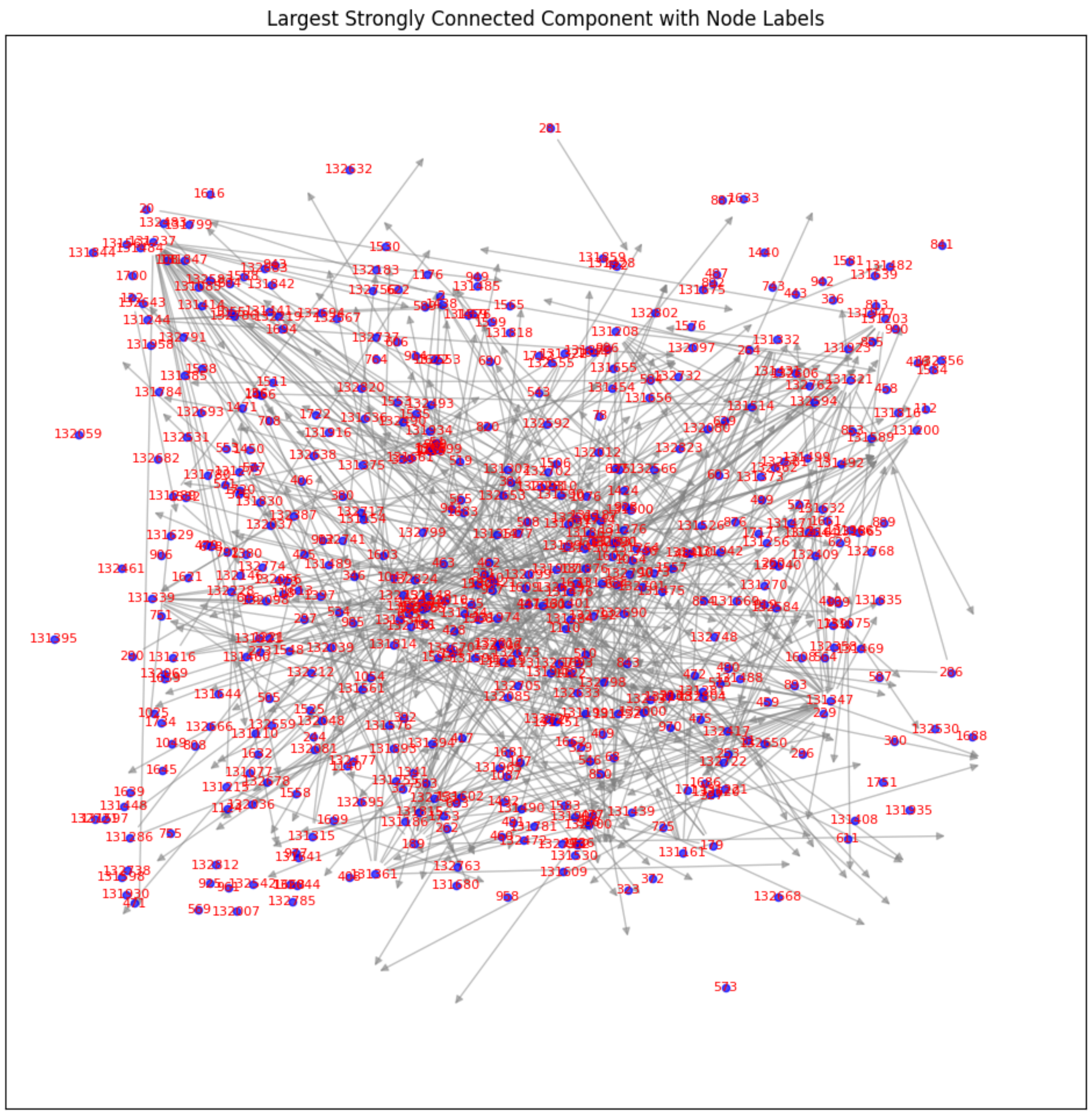
Largest Strongly Connected Component with Node Labels



Subgraph with first 500 nodes and edges from the largest subgraph derived from above

```
In [ ]: # Draw the graph
        plt.figure(figsize=(12, 12))  # Set the figure size
        #pos = nx.spring_layout(G_largest_scc, seed=42, iterations=2)  # Compute positions
        print("G_largest_scc: ", G_largest_scc)
        # Draw nodes with labels
        nodes_to_draw = list(G_largest_scc.nodes)[:500]
        subgraph = G_largest_scc.subgraph(nodes_to_draw)
        nx.draw_networkx_nodes(G_largest_scc, pos, nodelist=nodes_to_draw, node_size=20, no

        edges_to_draw = list(G_largest_scc.edges)[:500]  # Draw only 1000 edges
        nx.draw_networkx_edges(G_largest_scc, pos, edgelist=edges_to_draw, edge_color="gray

        nx.draw_networkx_labels(
            G_largest_scc,
            pos,
            labels={node: str(node) for node in list(G_largest_scc.nodes)[:500]},  # Displa
            font_size=8,
            font_color="red"
        )

        #print("G_largest_scc: ", G_largest_scc)
        plt.title("Largest Strongly Connected Component with Node Labels")
        plt.show()
```

```
G_largest_scc:  DiGraph with 23164 nodes and 223310 edges
```



Part 3.3 Refer Relevant material from Book related to Subgraph generation and provide explanation how you are generating subgraph

We are using the **Node Induced method** to generate the subgraph using **Random walk algorithm**. Please find the details about the same as below.

In the Book and Classes we discusses the below methods by which we can generate the subgraph. Both the below methods are used to generate the subgraph from a larger graph.

- **Node induced subgraph**
  - A node-induced subgraph is created by selecting a subset of nodes from the original graph and including all edges between those nodes that exist in the original graph
- **Edge induced subgraph**
  - An edge-induced subgraph is created by selecting a subset of edges from the original graph and including all nodes connected by those edges.

We can either use the **Random walk** or **Anonymous walk** technique to get the above subgraph.

- **Random walk**

  - A random walk is a process where a "walker" starts at a random node in the graph and moves to a neighboring node based on a specific probability distribution, repeating this process for a predefined number of steps.
- **Anonymous walk**

  - An anonymous walk focuses on the structural pattern of the walk, ignoring the specific identities of nodes. Instead of recording node IDs, it records the structural positions of nodes relative to the walk

Part 3.4 Generate Node Induced Subgraph.

- Used Random walk node induced method and formed the sub graph
- As the Random walk is happending the same node may be traversed multiple times and the the sub graph may contain the less number of nodes than the number of nodes which the induced graph is started with

In [ ]:
```
# Function to visualize a graph using NetworkX with customizable node labels and la
# Supports drawing both induced subgraphs and strongly connected components (SCC) s
# Uses a spring layout for node positioning and allows custom label adjustments
def draw_Graph(G, is_Induced=None):
    # Create a directed graph
    G_example = nx.DiGraph()
    G_example.add_nodes_from(G.nodes)
    G_example.add_edges_from(G.edges)
    pos = nx.spring_layout(G_example, seed=42)

    plt.figure(figsize=(10, 10))  # Set the figure size
    # draw node
    nx.draw(G_example, pos, node_size=20, with_labels=False, node_color="blue", edg

    # Adjust label position
    label_offset = 0.01  # Adjust this value to control how far the labels are from
    adjusted_labels_pos = {node: (x, y + label_offset) for node, (x, y) in pos.item

    # Add custom labels
    nx.draw_networkx_labels(
        G_example,
        adjusted_labels_pos,
        labels={node: str(node) for node in G_example.nodes()},  # Define custom la
        font_size=8,  # Change label size
        font_color="red")  # Change label color

    # check of its for induced graph
    if is_Induced:
      plt.title(f"Induced Subgraph with {len(G.nodes)} nodes and labels")
    else:
      plt.title(f"SCC Subgraph of Induced graph has {len(G.nodes)} nodes and labels
    plt.show()
```

In [ ]:
```
# Function to calculate the diameter of the largest strongly connected subgraph in
# The function optionally prints detailed information about the subgraph and its di
# It visualizes the subgraph by calling the draw_Graph function after computing the
def calculate_Diameter_And_Draw(G, is_Sub=None):
  # Compute the diameter of the subgraph
  try:
      G_largest_scc = G.subgraph(max(nx.strongly_connected_components(G), key=len))
      subgraph_diameter = nx.algorithms.approximation.diameter(G_largest_scc)
      if is_Sub:
        print(f"To calculate the diameter, we need to get the strongly connected su
        print("The strongly connected subgraph of the Induced graph is : ", G_large
        print(f"Diameter of the Induced graph which is strongly connected is: {subg
      else:
        print("The details of strongly connected input graph is: ", G_largest_scc)
        print(f"Diameter of the SCC Input graph is: {subgraph_diameter}")

        print("The scc sub graph referring to the diameter is specified below: ")
      draw_Graph(G_largest_scc)
  except nx.NetworkXError as e:
    print(f"Error computing diameter: {e}")
```

In [ ]:
```
from collections import Counter

def draw_InducedSubgraph(G, no_of_nodes, is_Draw=None, is_Diameter=None, is_Subgrap
  print("Input Graph statistics: ", G)

  # Randomly select a starting node from the graph
  start_node = random.choice(list(G.nodes))
  current_node = start_node
  nodes = [start_node]

  # Randomly walk through the graph and select neighboring nodes until reaching the
  for i in range(no_of_nodes - 1):
    neighbors = list(G.neighbors(current_node))
    if not neighbors:  # Dead end
        print(f"found a dead end at {i} number of node")
        break
    current_node = random.choice(neighbors)
    nodes.append(current_node)

  print(f"No of nodes for making the Induced graph: {len(nodes)} ")

  # Create a subgraph using the selected nodes
  subgraph = G.subgraph(nodes)

  print(f"As the walk is random, the same node might be traversed multiple times, s
  print(f"No of nodes after making the Induced graph: {len(subgraph.nodes)} ")

  # If is_Draw is True, visualize the induced subgraph
  if is_Draw:
    print("Details of the Induced subgraph is: ", subgraph)
    draw_Graph(subgraph, True)

  # If is_Diameter is True, calculate the diameter of the subgraph
  if is_Diameter:
    calculate_Diameter_And_Draw(subgraph, is_Subgraph)

  return subgraph, nodes
```

Below is the Induced graph made with 1000 nodes and same induced graph will be used for generating the node embedding using 2-hop method below

In [ ]:
```
# Generate an induced subgraph from the largest strongly connected component (SCC)
induced_graph, sub_nodes = draw_InducedSubgraph(G_largest_scc, 1000)
print("Induced subgraph Generated for 300 nodes, with graph details: ", induced_gra
print("nodes: ", len(induced_graph.nodes))
print("nodes: ", len(sub_nodes))
```

```
Input Graph statistics:  DiGraph with 23164 nodes and 223310 edges
No of nodes for making the Induced graph: 1000
As the walk is random, the same node might be traversed multiple times, so after mak
ing the subgraph, the number of nodes will be less
No of nodes after making the Induced graph: 72
Induced subgraph Generated for 300 nodes, with graph details:  DiGraph with 72 nodes
and 221 edges
nodes:  72
nodes:  1000
```

Part 3.5 Generate Node embedding using 2-hop method for all nodes in subgraph using MP-GNN library in PyTorch Geometric

In [ ]:
```
from torch_geometric.utils import subgraph
# Map original node indices to a 0-based range
node_mapping = {node: idx for idx, node in enumerate(induced_graph.nodes)}
```

```
edges_mapped = torch.tensor(
    [(node_mapping[src], node_mapping[dst]) for src, dst in induced_graph.edges],
    dtype=torch.long
).t().contiguous()

num_nodes = len(sub_nodes)

x = graph_data.x[sub_nodes]
y = graph_data.y[sub_nodes]

# Create data object
data = Data(x=x, edge_index=edges_mapped, y=y)
data.num_nodes = num_nodes
data
```

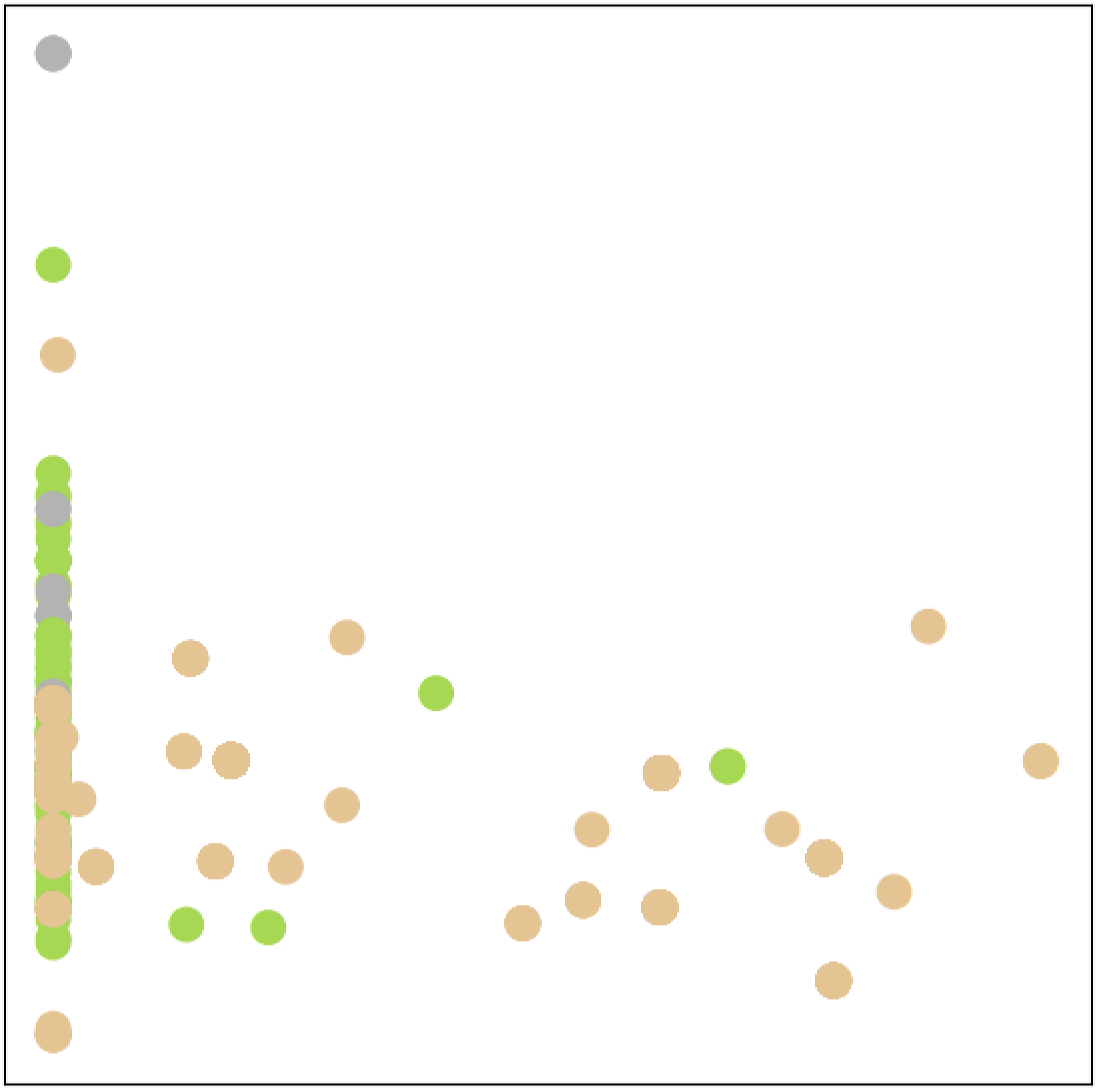
```
Out[ ]:  Data(x=[1000, 128], edge_index=[2, 221], y=[1000, 1], num_nodes=1000)
```

```
In [ ]: # Visualize the 2D embedding of node representations with optional epoch and loss i
        # The embedding is displayed as a scatter plot, with colors indicating node categor
        def visualize_embedding(h, color, epoch=None, loss=None):
            plt.figure(figsize=(7,7))
            plt.xticks([])
            plt.yticks([])
            h = h.detach().cpu().numpy()
            plt.scatter(h[:, 0], h[:, 1], s=140, c=color, cmap="Set2")
            if epoch is not None and loss is not None:
                plt.xlabel(f'Epoch: {epoch}, Loss: {loss.item():.4f}', fontsize=16)
            plt.show()
```

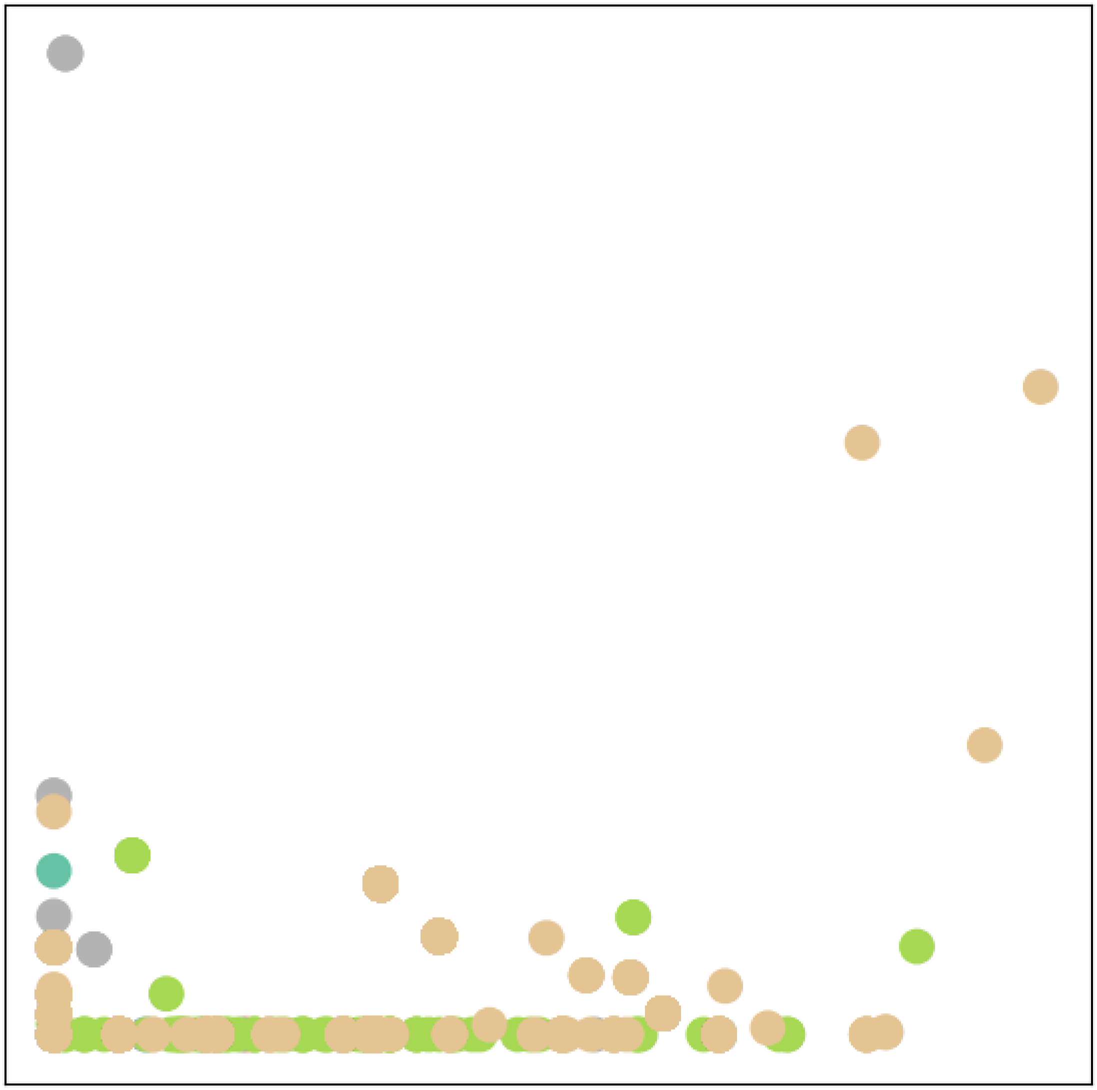
```
In [ ]: from re import X
        from torch.nn import Linear
        import torch.nn.functional as F
        # Define a 2-layer Graph Convolutional Network (GCN)
        class GCN(torch.nn.Module):
            def __init__(self, input_dim, hidden_dim, output_dim):
                super(GCN, self).__init__()
                self.conv1 = GCNConv(input_dim, hidden_dim)
                self.conv2 = GCNConv(hidden_dim, output_dim)
                self.classifier = Linear(hidden_dim, dataset.num_classes)

            def forward(self, x, edge_index):
                x = self.conv1(x, edge_index)
                x = F.relu(x)  # First GCN layer with ReLU activation
                x = self.conv2(x, edge_index)
                x = x.relu()        # Second GCN layer

                out = self.classifier(x)
                return out, x
                #return x

        # Initialize the GCN model
        model = GCN(input_dim=data.x.size(1), hidden_dim=64, output_dim=64)
        optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
        criterion = torch.nn.MSELoss()  # Loss function, can be adjusted based on task
```

```
# Training Loop
model.train()

epochs = 5
for epoch in range(epochs):
    optimizer.zero_grad()
    #print("X: ", data.x)
    #print(f"len1: {len(data.x)}, len2: {len(data.edge_index)} ")
    out, embeddings = model(data.x, data.edge_index)  # Forward pass

    loss = criterion(embeddings, torch.zeros_like(embeddings))  # Dummy loss for de
    loss.backward()
    optimizer.step()
    #if epoch % 10 == 0:
    print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
    visualize_embedding(embeddings, color=data.y)

# Get final node embeddings
#node_embeddings = model(data.x, data.edge_index).detach()
model.eval()
with torch.no_grad():
  print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
  out, node_embeddings = model(data.x, data.edge_index)


#print(f"Node embeddings shape: {node_embeddings.shape}")

#print(f'Embedding shape: {list(embeddings.shape)}')
visualize_embedding(node_embeddings, color=data.y)
```

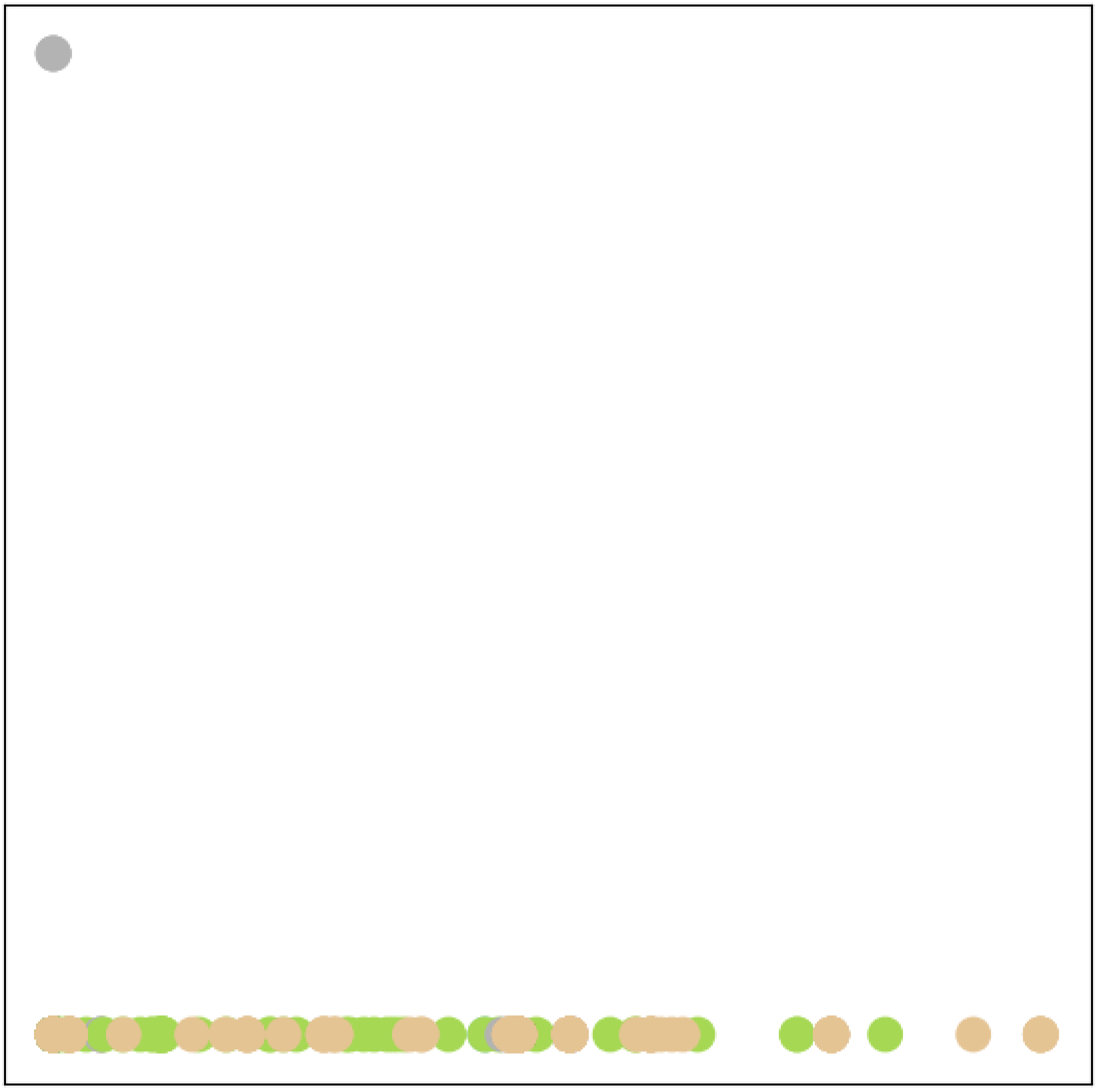
```
Epoch 0, Loss: 0.0176
```



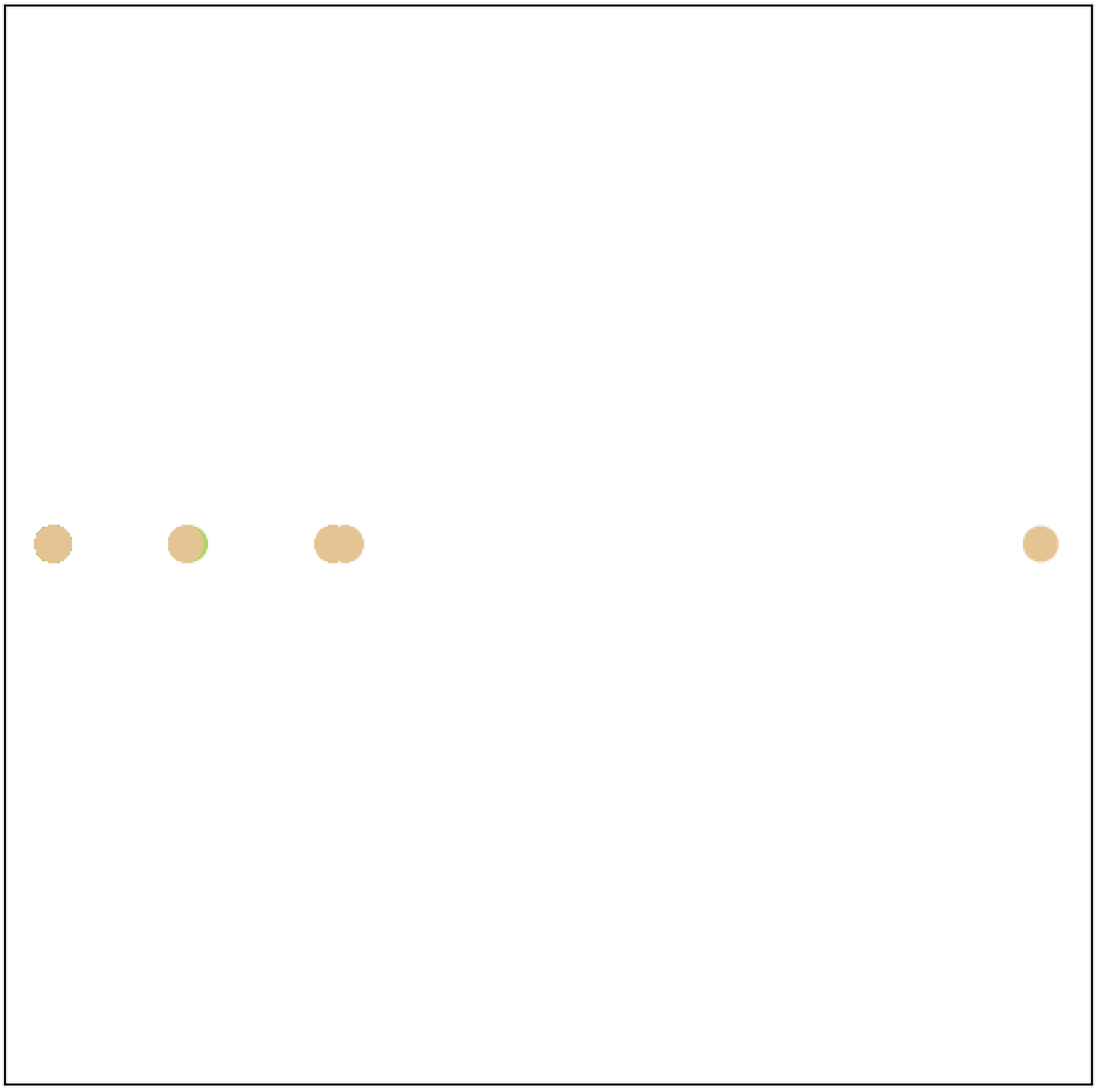
Epoch 1, Loss: 0.0021



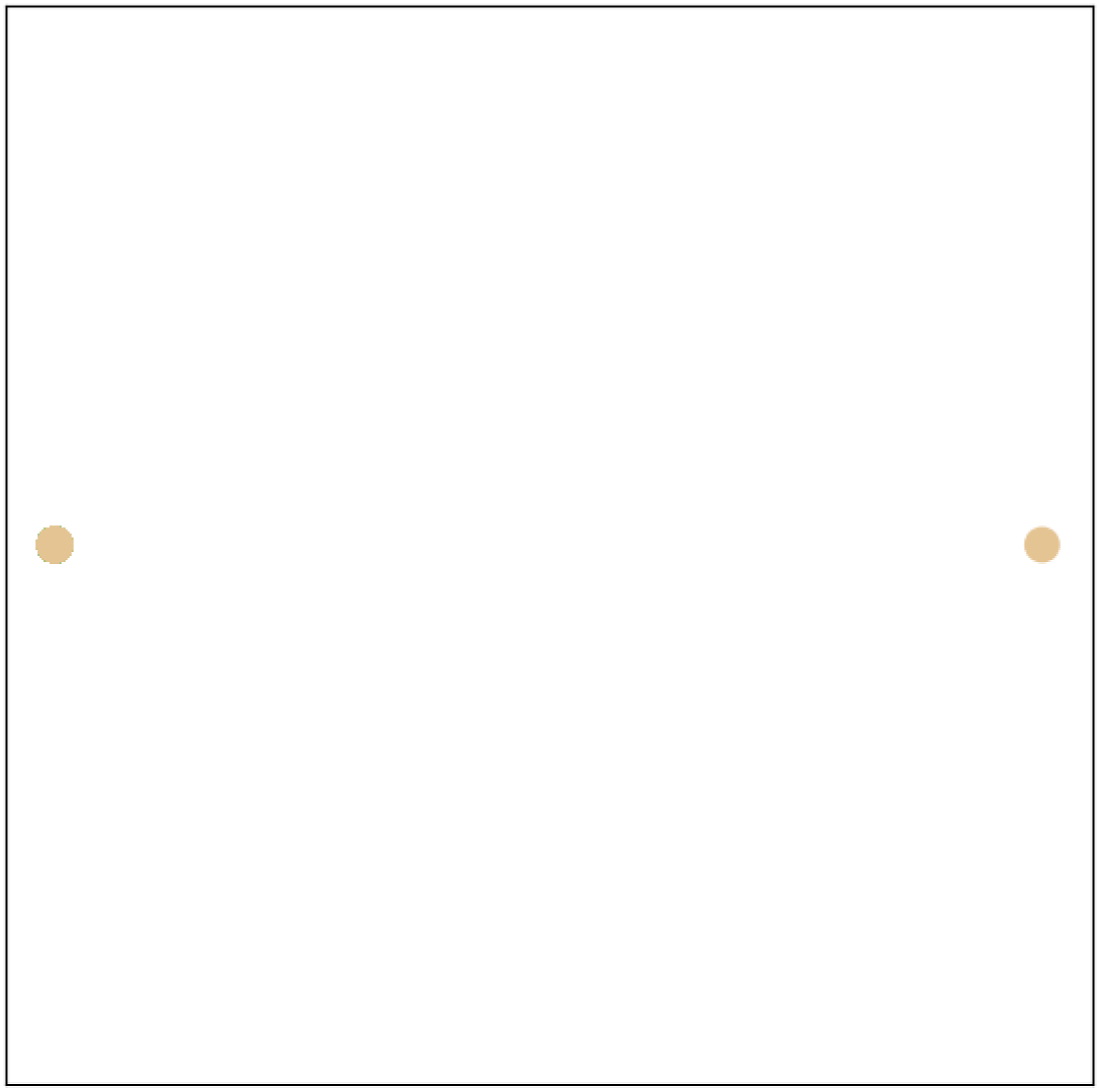
Epoch 2, Loss: 0.0005



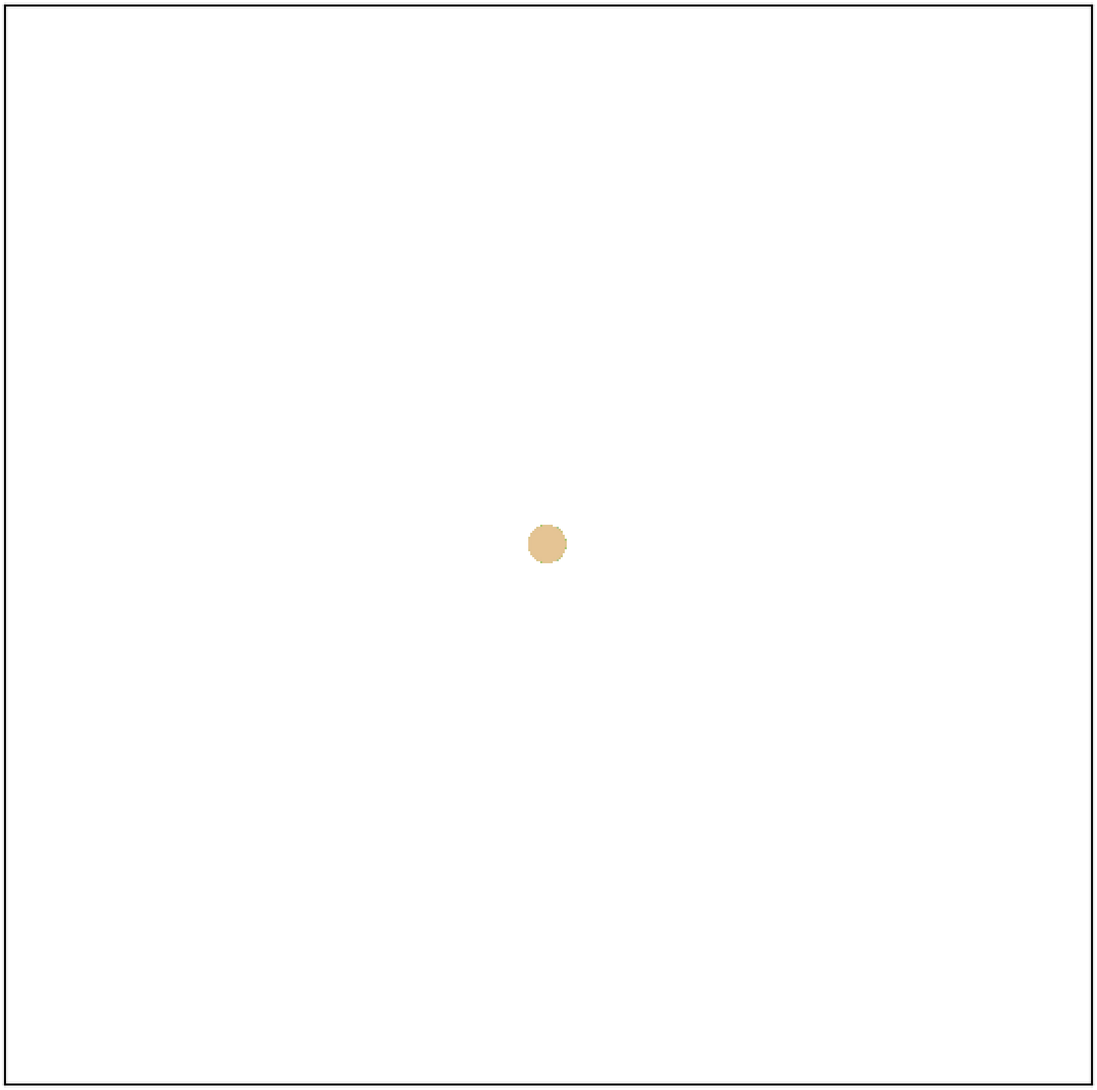
Epoch 3, Loss: 0.0001



Epoch 4, Loss: 0.0000



```
Epoch 4, Loss: 0.0000
```



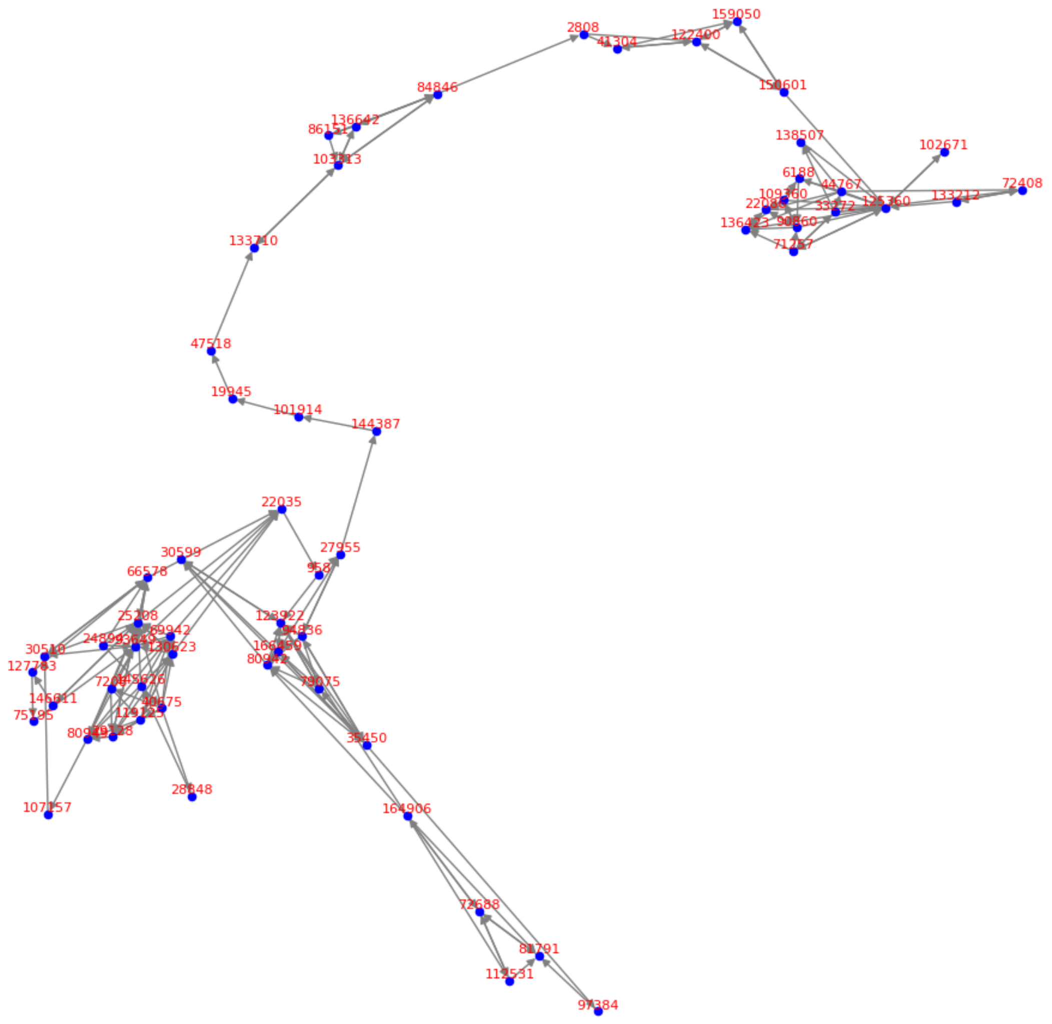
Part 3.6 Plot Subgrpah and compute their Diameter.

- In the below section we have created and drawn multiple Induced graphs and calculated its diameter.
- Have also plot the subgraph consisting of the diameter nodes and edges for multiple induced graphs

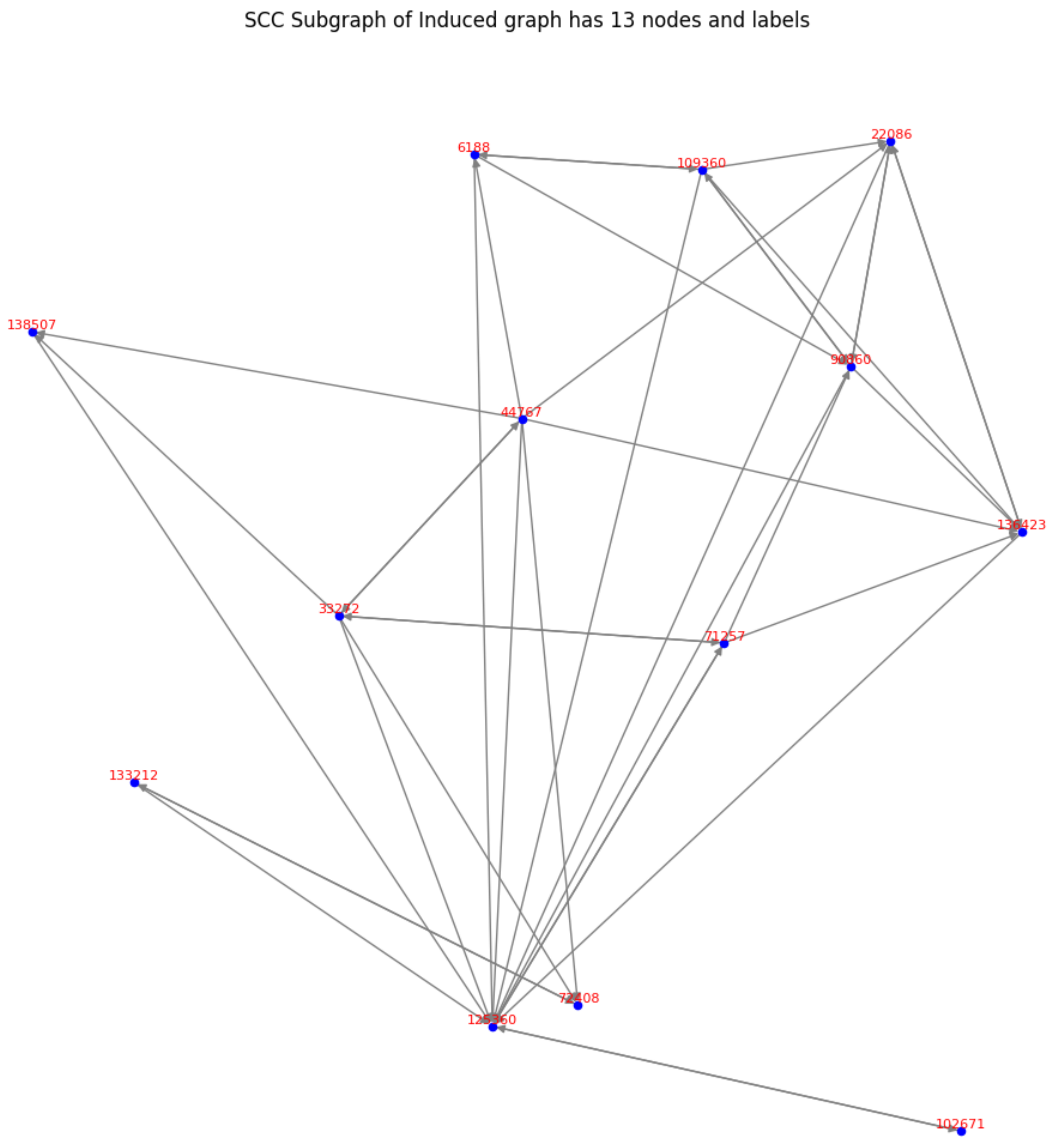
Induced graph for 200 nodes from the largest connected graph derived above and calculating diameter and plottig the sun graph of the same

In [ ]:
```
sub = draw_InducedSubgraph(G_largest_scc, 200, True, True, True)
```

```
Input Graph statistics:  DiGraph with 23164 nodes and 223310 edges
No of nodes for making the Induced graph: 200
As the walk is random, the same node might be traversed multiple times, so after mak
ing the subgraph, the number of nodes will be less
No of nodes after making the Induced graph: 60
Details of the Induced subgraph is:  DiGraph with 60 nodes and 178 edges
```


Induced Subgraph with 60 nodes and labels

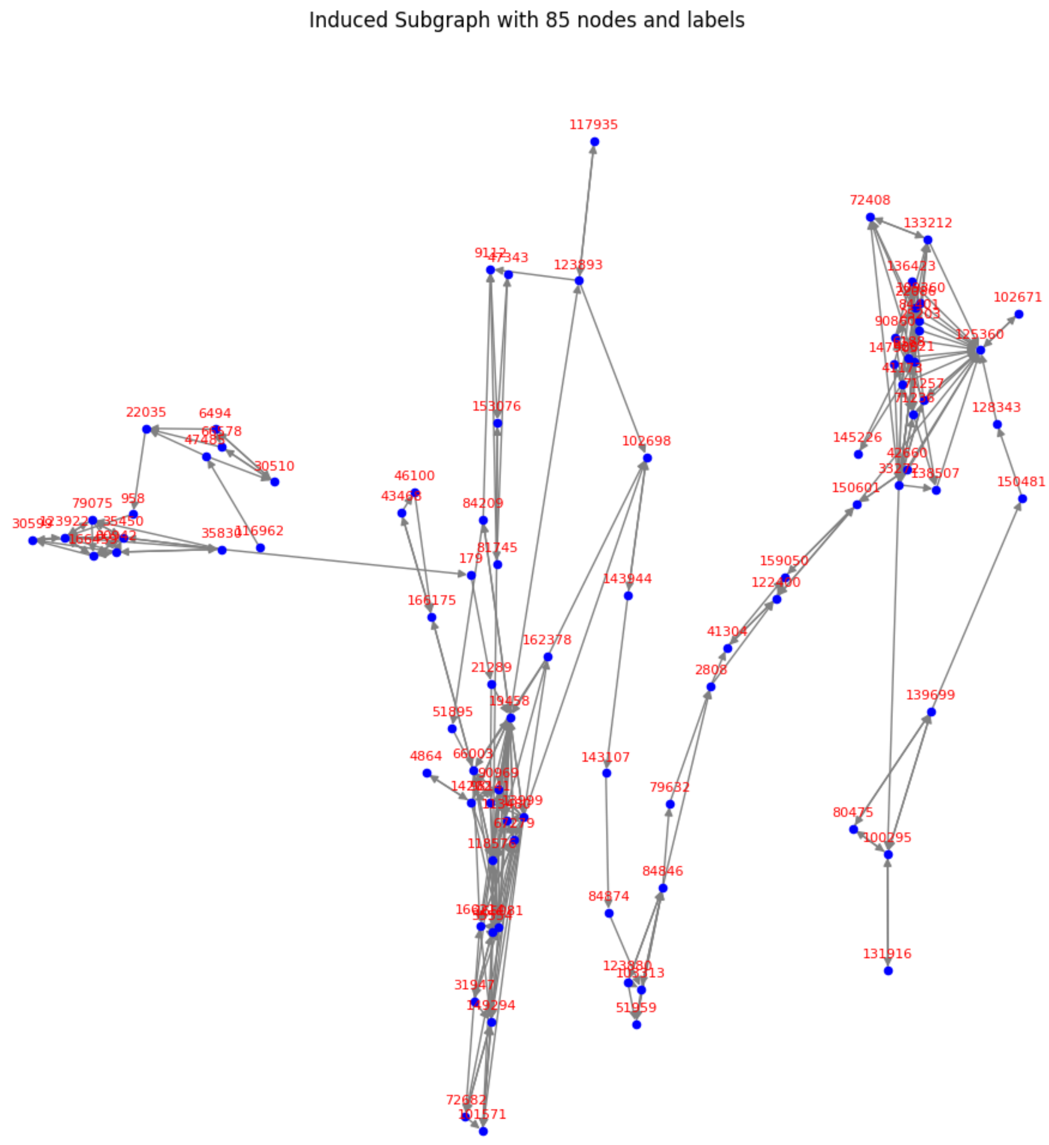
```
To calculate the diameter, we need to get the strongly connected subgraph and while
makging, the nodes that are strongly connected will only be considered
The strongly connected subgraph of the Induced graph is :  DiGraph with 13 nodes and
40 edges
Diameter of the Induced graph which is strongly connected is: 6
```



Induced graph for 1000 nodes from the largest connected graph derived above and calculating diameter and plottig the sun graph of the same

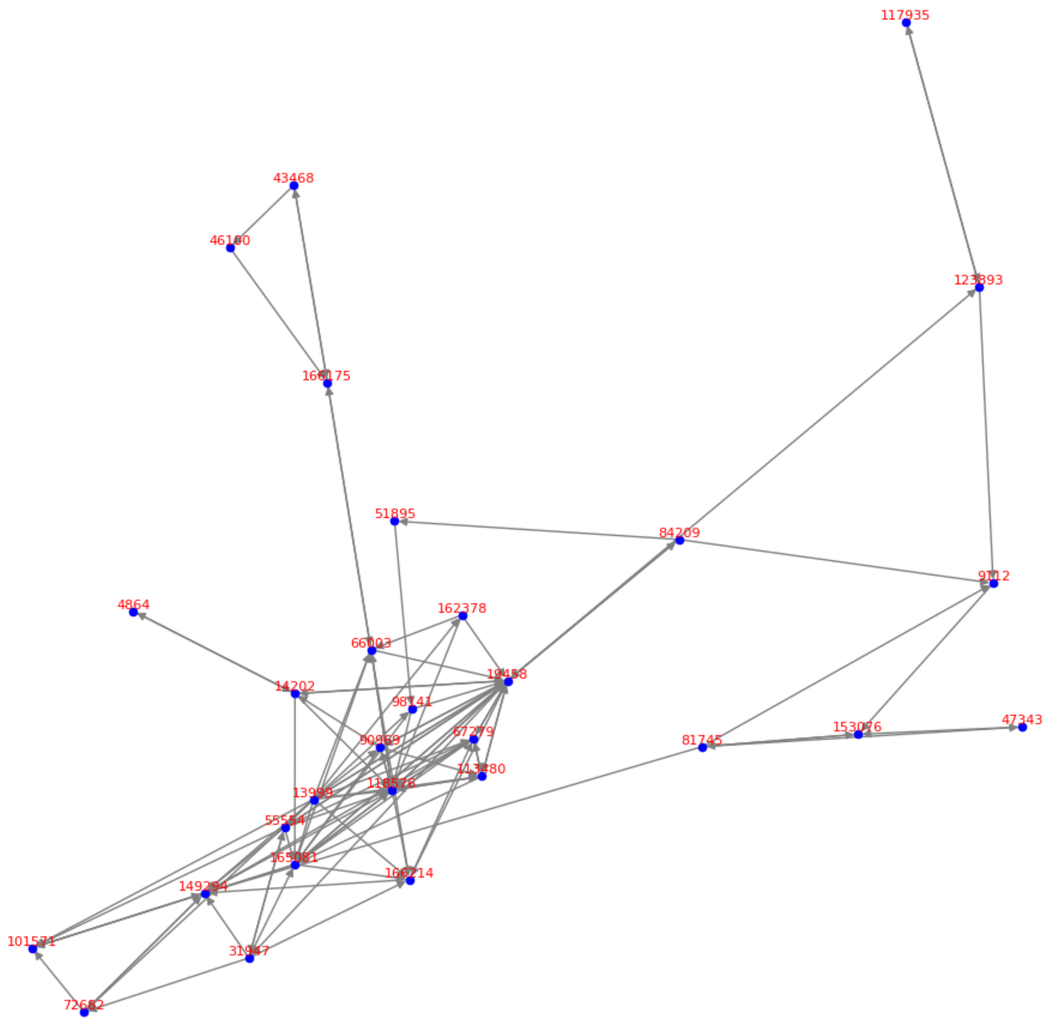
```
In [ ]: sub = draw_InducedSubgraph(G_largest_scc, 1000, True, True, True)
```

```
Input Graph statistics:  DiGraph with 23164 nodes and 223310 edges
No of nodes for making the Induced graph: 1000
As the walk is random, the same node might be traversed multiple times, so after mak
ing the subgraph, the number of nodes will be less
No of nodes after making the Induced graph: 85
Details of the Induced subgraph is:  DiGraph with 85 nodes and 268 edges
```



```
To calculate the diameter, we need to get the strongly connected subgraph and while
makging, the nodes that are strongly connected will only be considered
The strongly connected subgraph of the Induced graph is :  DiGraph with 29 nodes and
107 edges
Diameter of the Induced graph which is strongly connected is: 11
```

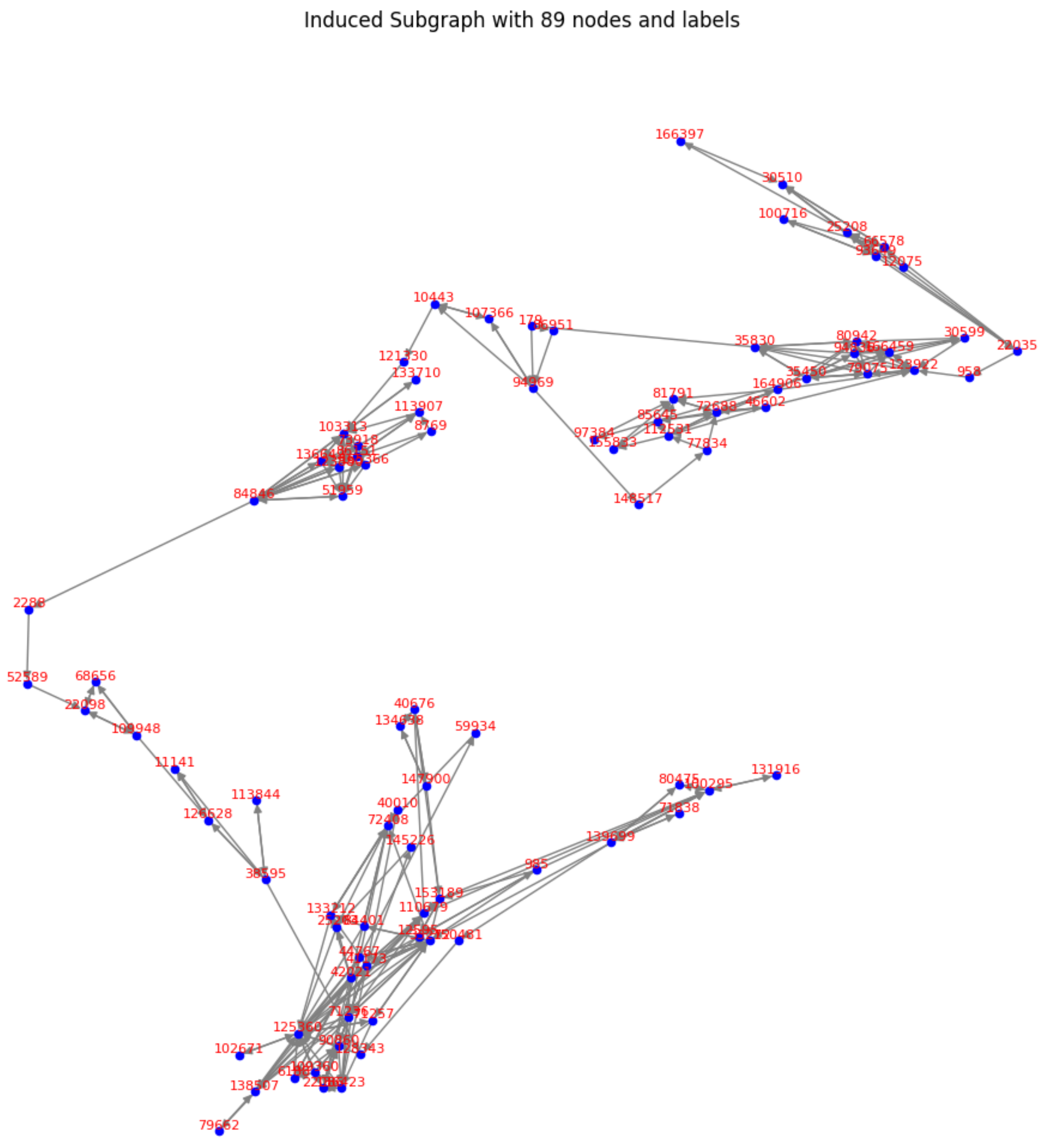
SCC Subgraph of Induced graph has 29 nodes and labels



Induced graph for 1500 nodes from the largest connected graph derived above and calculating diameter and plottig the sun graph of the same

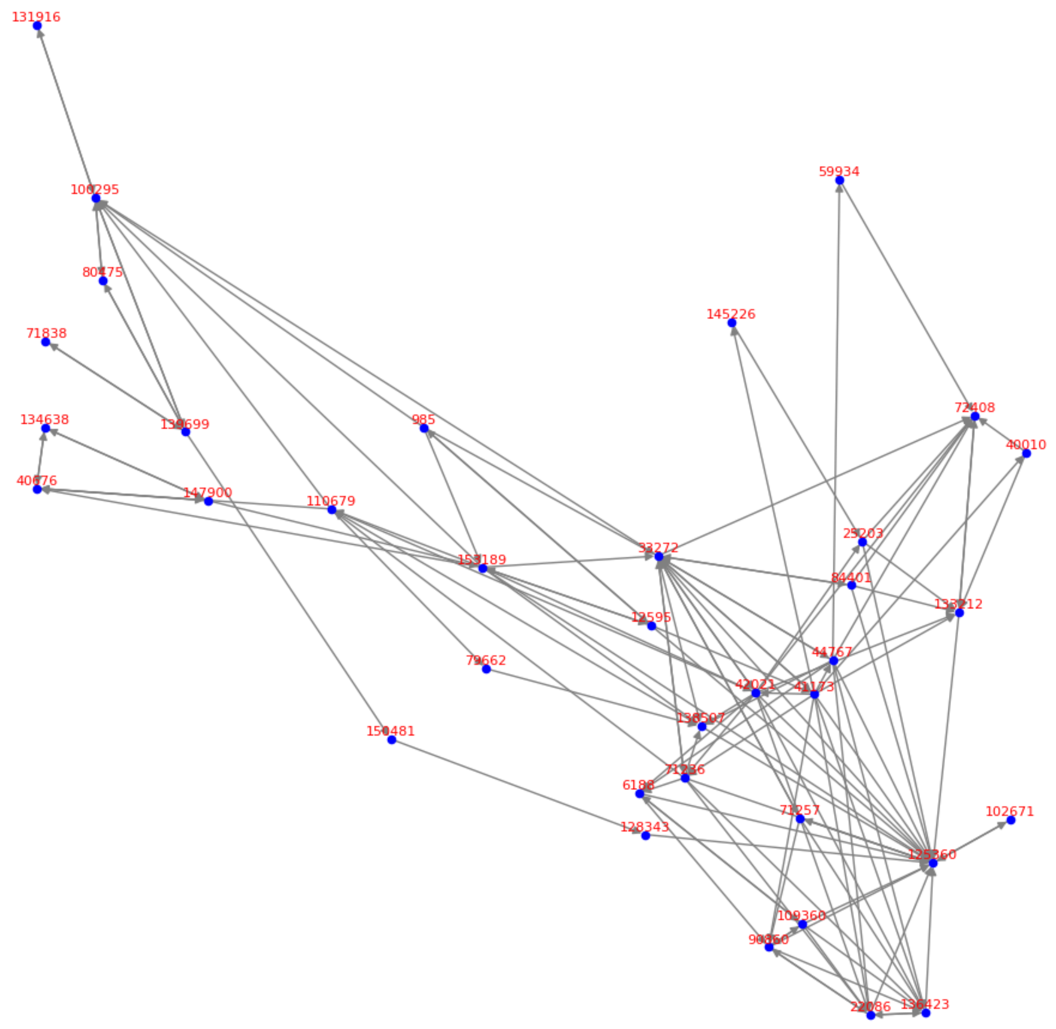
```
In [ ]: sub = draw_InducedSubgraph(G_largest_scc, 1500, True, True, True)
```

```
Input Graph statistics:  DiGraph with 23164 nodes and 223310 edges
No of nodes for making the Induced graph: 1500
As the walk is random, the same node might be traversed multiple times, so after mak
ing the subgraph, the number of nodes will be less
No of nodes after making the Induced graph: 89
Details of the Induced subgraph is:  DiGraph with 89 nodes and 283 edges
```



```
To calculate the diameter, we need to get the strongly connected subgraph and while
makging, the nodes that are strongly connected will only be considered
The strongly connected subgraph of the Induced graph is :  DiGraph with 36 nodes and
127 edges
Diameter of the Induced graph which is strongly connected is: 14
```

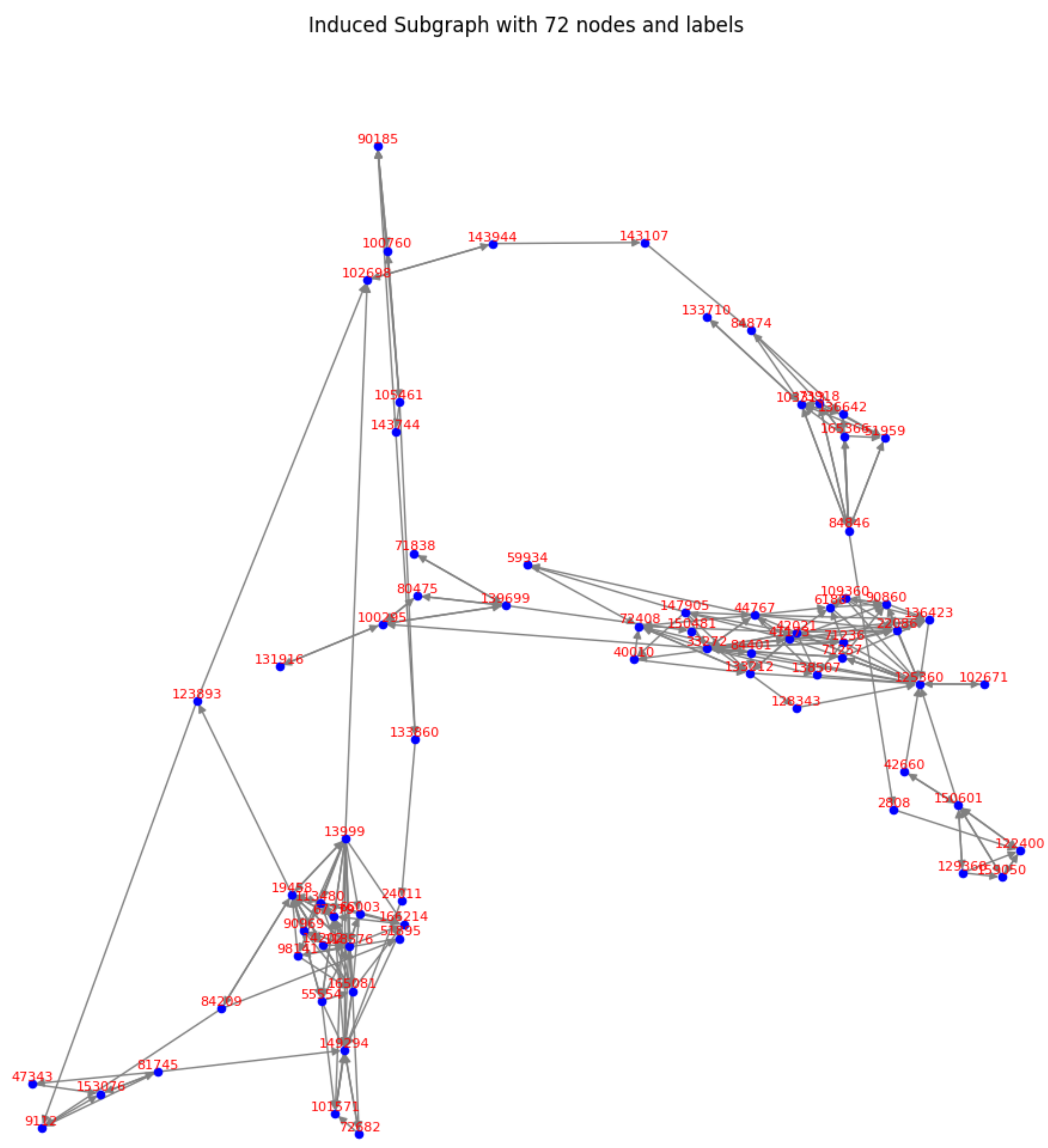
SCC Subgraph of Induced graph has 36 nodes and labels



Induced graph for 2000 nodes from the largest connected graph derived above and calculating diameter and plottig the sun graph of the same

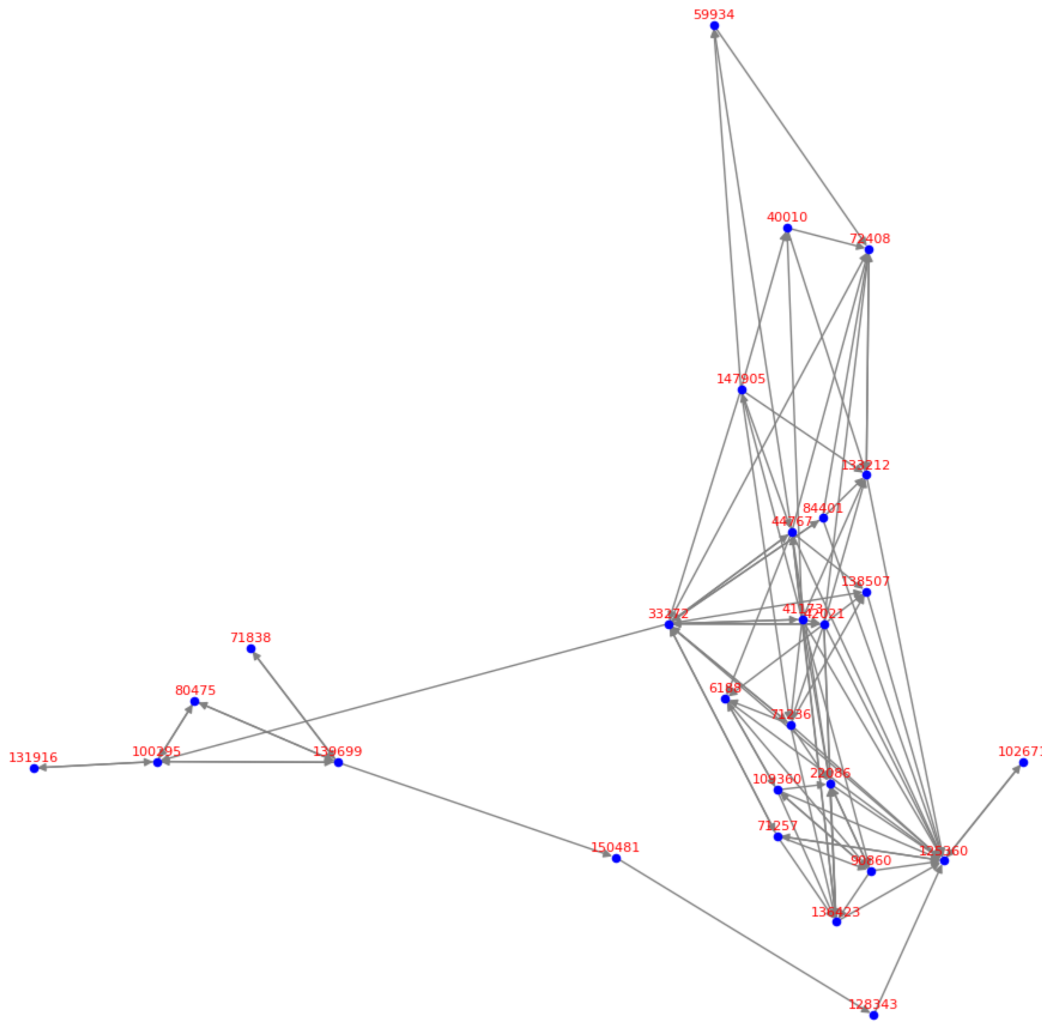
```
In [ ]: sub = draw_InducedSubgraph(G_largest_scc, 2000, True, True, True)
```

```
Input Graph statistics:  DiGraph with 23164 nodes and 223310 edges
No of nodes for making the Induced graph: 2000
As the walk is random, the same node might be traversed multiple times, so after mak
ing the subgraph, the number of nodes will be less
No of nodes after making the Induced graph: 72
Details of the Induced subgraph is:  DiGraph with 72 nodes and 244 edges
```



```
To calculate the diameter, we need to get the strongly connected subgraph and while
makging, the nodes that are strongly connected will only be considered
The strongly connected subgraph of the Induced graph is :  DiGraph with 27 nodes and
98 edges
Diameter of the Induced graph which is strongly connected is: 9
```

SCC Subgraph of Induced graph has 27 nodes and labels


59934
40010
72408
147905
133212
84401
44767
138507
33272
41173
42021
71838
80475
6188
71236
131916
100295
139699
102671
109360
22086
71257
150481
125360
90860
136423
128343