Task 3

```
%pip install torch_geometric

import torch
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
import torch.nn.functional as F

# 0,1,1,1]
```

This cell installs torch_geometric library and imports the required modules. The torch library provides basic operations, Data stores the graph structure, SAGEConv implements GraphSAGE layers, and F provides activation functions.

```
#--- Define a small graph with 6 nodes ---
# Node features (2 features per node).
# Here benign users have [1, 0] and malicious have [0, 1] for illustration.
x = torch.tensor(
    [
        [1.0, 0.0],  # Node 0 (benign)
        [1.0, 0.0],  # Node 1 (benign)
        [1.0, 0.0],  # Node 2 (benign)
        [0.0, 1.0],  # Node 3 (malicious)
        [0.0, 1.0],  # Node 4 (malicious)
        [0.0, 1.0]  # Node 5 (malicious)
    ],
    dtype=torch.float,
)
```

This cell creates features for 6 nodes in the graph. Benign users are represented with features [1.0, 0.0] while malicious users have [0.0, 1.0]. The graph contains 3 benign nodes (0, 1, 2) and 3 malicious nodes (3, 4, 5). Each node has 2 features.

```
# and malicious users (3-4-5 fully connected), plus one cross-edge 2-3.
edge_index = (
    torch.tensor(
        [
            [0, 1],
            [1, 0],
            [1, 2],
            [2, 1],
            [0, 2],
            [2, 0],
            [3, 4],
            [4, 3],
            [4, 5],
            [5, 4],
            [3, 5],
            [5, 3],
            [2, 3],
            [3, 2],  # one connection between a benign (2) and malicious (3)
        ],
        dtype=torch.long,
    )
    .t()
    .contiguous()
)
```

This cell defines the edges between nodes. The benign users (0, 1, 2) are fully connected to each other, and malicious users (3, 4, 5) are also fully connected. There is one connection between node 2 and node 3 linking the two groups. All edges are bidirectional.

```
# Labels: 0 = benign, 1 = malicious
# y contains the true labels of the 6 nodes:
# Nodes 0, 1, 2 are benign → label 0
# Nodes 3, 4, 5 are malicious → label
# data is a torch_geometric.data.Data object containing
# x: node features
# edge_index: graph connections (edges)
# y: labels
y = torch.tensor([0, 0, 0, 1, 1, 1], dtype=torch.long)

data = Data(x=x, edge_index=edge_index, y=y)
```

```python
# --- Define a two-layer GraphSAGE model ---
# his defines a 2-layer GraphSAGE neural network.
# in_channels=2 means each node has 2 features.
# hidden_channels=4 creates a 4-dimensional hidden embedding.
# out_channels=2 means the model outputs scores for 2 classes (benign and malicious).



class GraphSAGENet(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GraphSAGENet, self).__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        # First layer: sample neighbors and aggregate
        x = self.conv1(x, edge_index)
        x = F.relu(x)  # non-linear activation
        # Second layer: produce final embeddings/class scores
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)  # log-probabilities for classes
```

This cell creates the labels where 0 represents benign users and 1 represents malicious users. All components (features, edges, labels) are combined into a Data object. The GraphSAGE model class is defined with two layers: first layer processes 2 input features to 4 hidden features, second layer outputs 2 class scores.

```python
# Instantiate model: input dim=2, hidden=4, output dim=2 (benign vs malicious)
model = GraphSAGENet(in_channels=2, hidden_channels=4, out_channels=2)

# Simple training loop
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
model.train()
for epoch in range(50):
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out, data.y)  # negative log-likelihood
    loss.backward()
    optimizer.step()
```

This cell initializes the model and trains it for 50 epochs. During each epoch, the model processes the graph, calculates predictions, computes loss, and updates weights. Adam optimizer is used with learning rate 0.01.

```
# After training, we can check predictions
model.eval()
pred = model(data.x, data.edge_index).argmax(dim=1)
print("Predicted labels:", pred.tolist())  # e.g. [0,0,
```

```
Predicted labels: [0, 0, 0, 1, 1, 1]
```

This cell evaluates the trained model. The output [0, 0, 0, 1, 1, 1] shows perfect classification - correctly identifying the first three nodes as benign and last three as malicious.