

Ahmed Mahmoud Hassan

2205155

```
--- 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 part creates a tensor representing 6 nodes where each node has two features that label it as normal $[1, 0]$ or malicious $[0, 1]$

```
# Edge list (undirected). Connect benign users (0-1-2 fully connected)
# 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 part defines all the connections between nodes showing which users interact with who in the graph
- It builds two connected groups (normal and malicious) and adds one link between them so the model can learn how information flows across communities

```

# 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 1
# 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)

# --- Define a two-layer GraphSAGE model ---
# This 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 part creates the labels for each node, packages everything into a Data object, and defines a simple 2 layer GraphSAGE model
- It tells the model which nodes are good or bad, then builds a small neural network that learns from connections to classify each node as normal or malicious

```

# 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()

# 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 part trains the GraphSAGE model for 50 steps using Adam, then switches to evaluation mode to predict each node's label
- It teaches the model to tell good from bad nodes then prints out its final guesses after learning