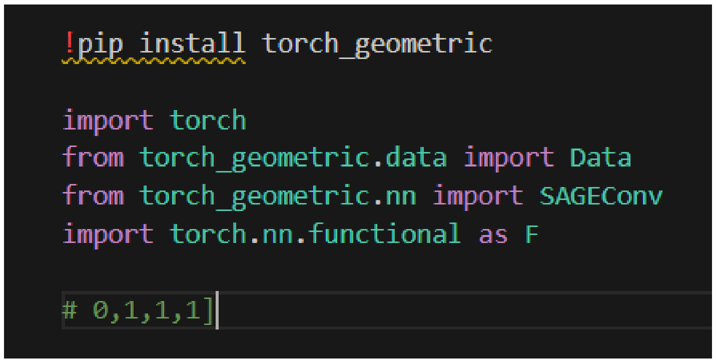
الاسم:محمد أحمد علي مبارك

Id:2205249



```
!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]
```

`!pip install torch_geometric`
Installs the PyTorch Geometric library, which is used for building graph neural networks.

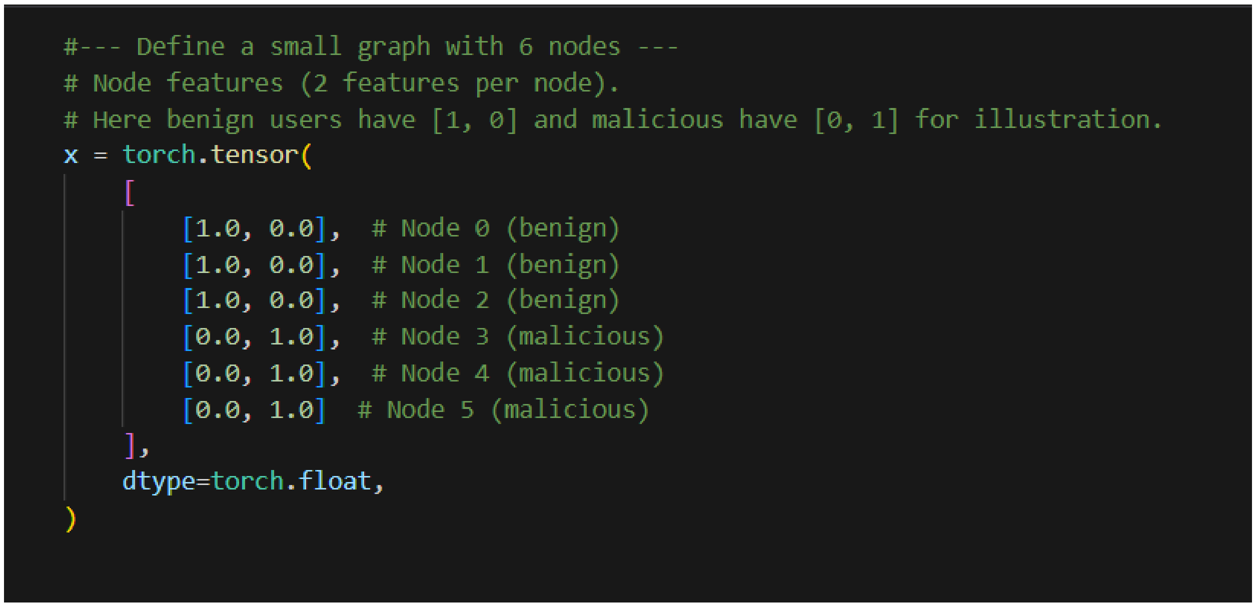
```
import torch
```
Imports the main PyTorch library for tensors and deep-learning operations.

```
from torch_geometric.data import Data
```
Imports the `Data` class, which is used to store graph information (nodes, edges, and features).

```
from torch_geometric.nn import SAGEConv
```
Imports the GraphSAGE convolution layer, used to update node features based on their neighbors.

```
import torch.nn.functional as F
```
Imports functional utilities such as activation functions like ReLU.

```
# 0,1,1,1]
```
This comment likely refers to a node feature vector such as `[0, 1, 1, 1]`, representing the numerical attributes of a node.



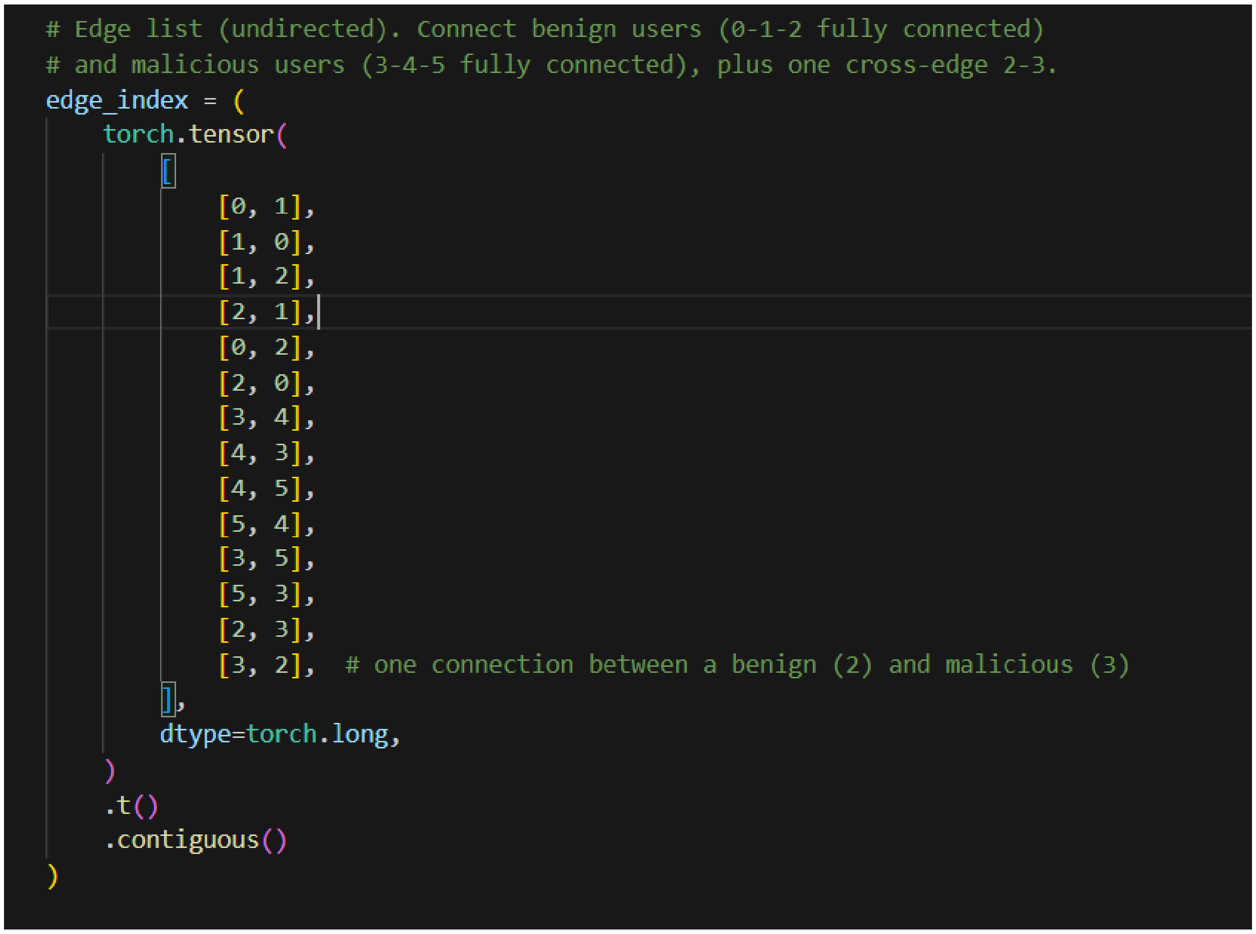
```
#--- Define a small graph with 6 nodes ---
# Node features (2 features per node).
# Here benign users have [1, 0] and malicious have [0, 1] for illustration.
x = torch.tensor(
    [
        [1.0, 0.0],  # Node 0 (benign)
        [1.0, 0.0],  # Node 1 (benign)
        [1.0, 0.0],  # Node 2 (benign)
        [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 of the code defines the **node features** for a small graph with six nodes.
Each node has **two features**:

`[1, 0]` represents a **benign** (normal) user

`[0, 1]` represents a **malicious** user

The tensor assigns one of these feature vectors to each of the six nodes. The first three nodes are labeled as benign, and the last three are labeled as malicious.



```python
# 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()
)
```

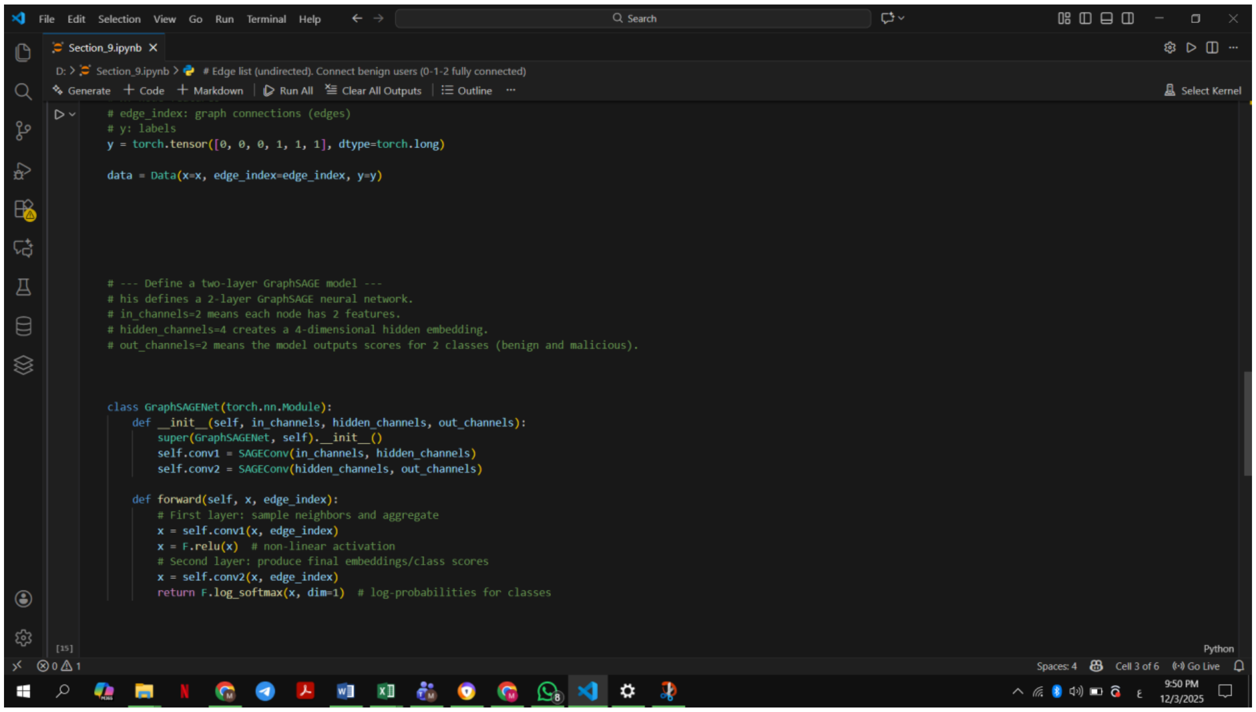
This section defines the **edges** of the graph. The edge list describes which nodes are connected to each other. The graph is **undirected**, so every connection is listed in both directions (e.g., 0→1 and 1→0).

Nodes **0, 1, and 2** (benign users) are all connected to each other (fully connected).

Nodes **3, 4, and 5** (malicious users) are also fully connected.

There is **one cross-group connection** between node **2** (benign) and node **3** (malicious), shown by edges 2→3 and 3→2.

The `.t().contiguous()` part simply transposes the edge list into the format expected by PyTorch Geometric.



```python
    # 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 ---
    # 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
```

Labels (`y`) and Data Object

The `y` tensor stores the **true class labels** of the six nodes.

Nodes **0, 1, and 2** are benign → label **0**

Nodes **3, 4, and 5** are malicious → label **1**

The `Data` object combines everything into one graph structure:

`x` → node features

`edge_index` → graph connections

`y` → node labels

This object is what the model will use for training.

GraphSAGE Model

A two-layer GraphSAGE neural network is defined.

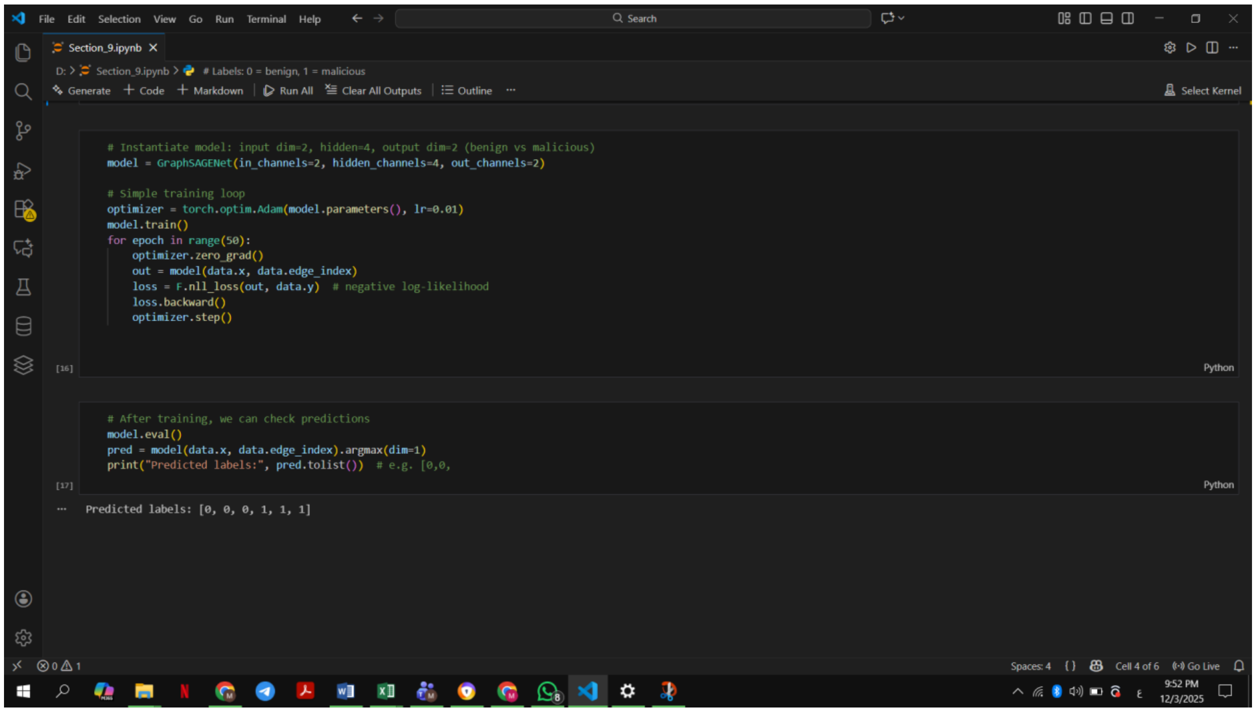
Input size = 2 features per node

Hidden layer size = 4, allowing the model to learn richer node representations

Output size = 2, corresponding to the two classes: benign and malicious

The model works as follows:

1. **First GraphSAGE layer**
   Aggregates information from each node's neighbors and produces a hidden representation.
   A ReLU activation adds non-linearity.
2. **Second GraphSAGE layer**
   Produces the final output for each node, which represents class scores.
3. **Log-softmax output**
   Converts the scores into log-probabilities for the two classe


File Edit Selection View Go Run Terminal Help
Search
Section_9.ipynb
D: > Section_9.ipynb > # Labels: 0 = benign, 1 = malicious
Generate + Code + Markdown Run All Clear All Outputs Outline
Select Kernel
[16]
Python
[17]
Python
Spaces: 4 Cell 4 of 6 Go Live
9:52 PM 12/3/2025

```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()
```

```python
# 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]
```

Model Initialization

The model is created with:

- **2 input features**
- **4 hidden units**
- **2 output classes** (benign vs malicious)

This matches the node feature size and the number of classes.

Training Loop

An Adam optimizer is used with a learning rate of 0.01.
The model is trained for **50 epochs** using the following steps:

1. **Zero gradients**
   Clears previous gradient values.
2. **Forward pass**
   The model computes predictions for all nodes.
3. **Compute loss**
   Negative log-likelihood (`nll_loss`) compares predictions with the true labels.
4. **Backward pass**
   Computes gradients for all model parameters.
5. **Optimizer step**
   Updates the model parameters to reduce the loss.

Prediction

After training, the model is switched to evaluation mode.
It predicts a label for each node by selecting the class with the highest score.
The final line prints the predicted labels as a list (e.g., `[0, 0, 0, 1, 1, 1]`).