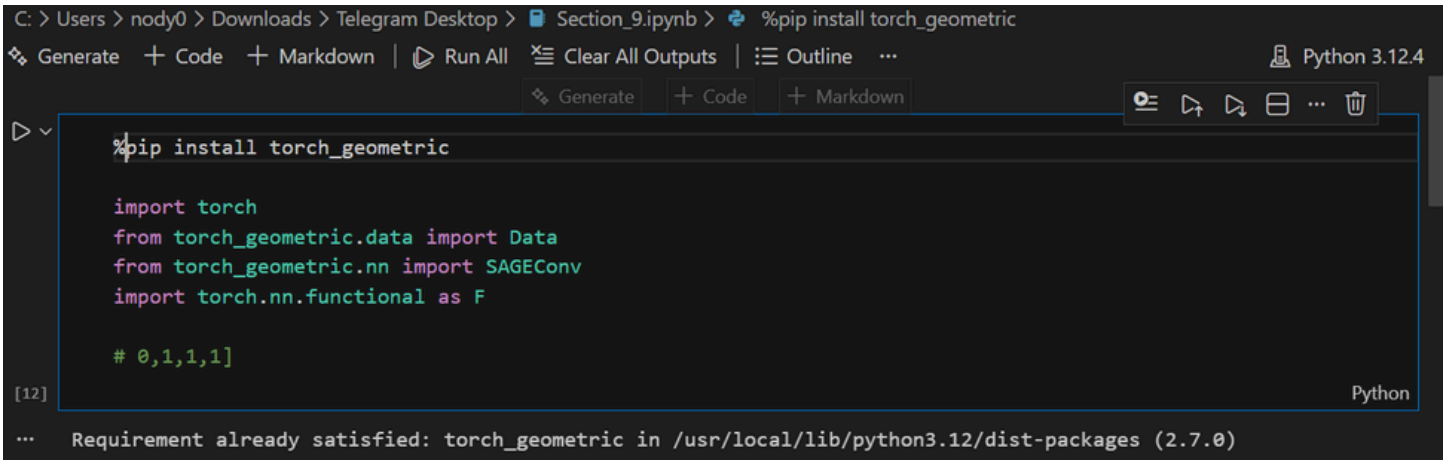
Nada mohamed abdelsatar

2205173



Installs torch_geometric

```python
#--- 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,
)
```

- Creates **6 nodes** in the graph.
- Each node has **2 features**.
- Benign nodes get the feature [1, 0].
- Malicious nodes get the feature [0, 1].
- Stores everything in a PyTorch tensor called **x**.

```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()
)
```

- Creates the edges of the graph.
- Defines which nodes are connected.
- Adds one cross-edge between node 2 (benign) and node 3 (malicious).
- Converts the edge list into edge_index.
- Makes the graph undirected by adding both directions.

```python
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
```

- Creates the graph data.
- Creates labels for each node (0 = benign, 1 = malicious).
- Combines features, edges, and labels.
- Defines a GraphSAGE neural network.
- GraphSAGENet is a GraphSAGE model with 2 layers:
- First layer learns hidden features for each node.
- Second layer predicts if a node is benign or malicious.
- Stores everything inside a Data object used by PyG.

```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()
```

- Creates a simple GraphSAGE model (two layers).
- This part creates the model and trains it.
  It runs 50 steps where it:
- gets predictions,
- computes the loss,
- and updates the model's weights.

```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,
```

- switches the model to evaluation mode, makes predictions for all nodes, and prints the final predicted labels.