# Report for Notebook Section_9.ipynb

**Path:** /mnt/data/Section_9.ipynb

## Notebook Summary

- **Number of cells:** 6
- **Code cells:** 6
- **Markdown cells:** 0

## Headings / Structure

- No markdown headings detected in the notebook.

## Key Imports Found

- No imports detected (static scan)

## Functions and Classes Detected

- **Class:** GraphSAGENet
  **Methods:** __init__, forward

## Outputs Found in Notebook

- stream: 2
- stream_lines: 24

## Notable TODO / FIXME Comments

- No TODO/FIXME/XXX comments found.

## Example Code Snippets (First 5 Code Cells)

### Cell 0

```
!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]
```

**Cell 1**

```
#--- 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)

    ],
```

**Cell 2**

```
# 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],
```

**Cell 3**

```
# 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

y = torch.tensor([0, 0, 0, 1, 1, 1], dtype=torch.long)

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

**Cell 4**

```
# 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)
   loss.backward()
   optimizer.step()
```

---

**Static Analysis — Potential Issues & Recommendations**

This is an automated, best-effort static scan. The code was **not executed**.

**Recommendations**

- **Dependency management:** Add a requirements.txt or environment.yml including all imports.
- **Cell outputs:** Save important results (figures, tables) to external files when needed.
- **Functionization:** Convert repeated logic into reusable functions with docstrings.
- **Error handling:** Use try/except blocks for file I/O or network operations.
- **Type hints & docstrings:** Improve readability and IDE support.
- **Testing:** Add unit tests (e.g., pytest) for important components.

### How to Run This Notebook Reproducibly

1. Create a virtual environment (conda/venv).
2. Install dependencies from requirements.txt.
3. Open the notebook in JupyterLab/Notebook.
4. Run cells in order.
5. For heavy tasks, consider porting logic to Python scripts.

### Suggestions for Further Improvements

- Split logic into smaller modules/scripts.
- Add a README explaining purpose, workflow, and expected outputs.
- Document datasets and add sample data or downloading utilities.
- Add explanations and captions to visualizations.

### Summary

- **6** code cells, **0** markdown cells.
- **0** imports detected (static scan).
- **1** class (GraphSAGENet).
- Recommendations provided for clarity, reproducibility, and maintainability.