Name: Karen Alfred Habib
ID: 2205236

Social Network Security

Assignment 2

Bot Detection in Facebook Ego Networks

****

1. Dataset:

- The dataset used is the well-known **Facebook combined graph** from SNAP (Stanford Network Analytics Project), containing :
  - **4,039 nodes** (users) and
  - **88,234 undirected edges** (mutual friendships).
- This is the exact same data family referenced in the assignment link, we used the merged version for a richer and more realistic evaluation.
- We created **realistic synthetic bots** (7% of nodes ≈ 282 bots) by selecting originally normal-looking users (degree 20–300) and dramatically reducing their connections to only 3–8 edges each.
- We build a bot-detection classifier using **GraphSAGE**, then we test three scenarios:

1. **Baseline (no attack)**
2. **Structural Evasion Attack**
3. **Graph Poisoning Attack**

Finally, we compare the performance and analyze how each attack affects the model.

---

2. Dataset & Graph Construction:

We load the Facebook graph using **NetworkX** and build an undirected graph where each node represents a user.

---

3. Feature Extraction:

For each node, we extract the following graph-based features:

- Node degree
- Clustering coefficient
- PageRank
- Betweenness centrality
- Eigenvector centrality

These features are later used by the ***GraphSAGE*** model.

- Degree & Degree Centrality
- Clustering Coefficient

- Betweenness Centrality
- Closeness Centrality
- Eigenvector Centrality
- Community detection using **Louvain algorithm** (modularity + community size)

---

4. Train/Eval Function (GraphSAGE Model):

❖ We split the nodes into:

- **80% training**
- **20% testing**

❖ We train the GraphSAGE model using:

- **Adam optimizer**
- **Negative Log-Likelihood loss**
- **30 epochs**

❖ During training, the model learns how the graph structure relates to bot behavior.

❖ After training, we compute:

- AUC-ROC
- PR-AUC
- Precision

- Recall
- F1-score

---

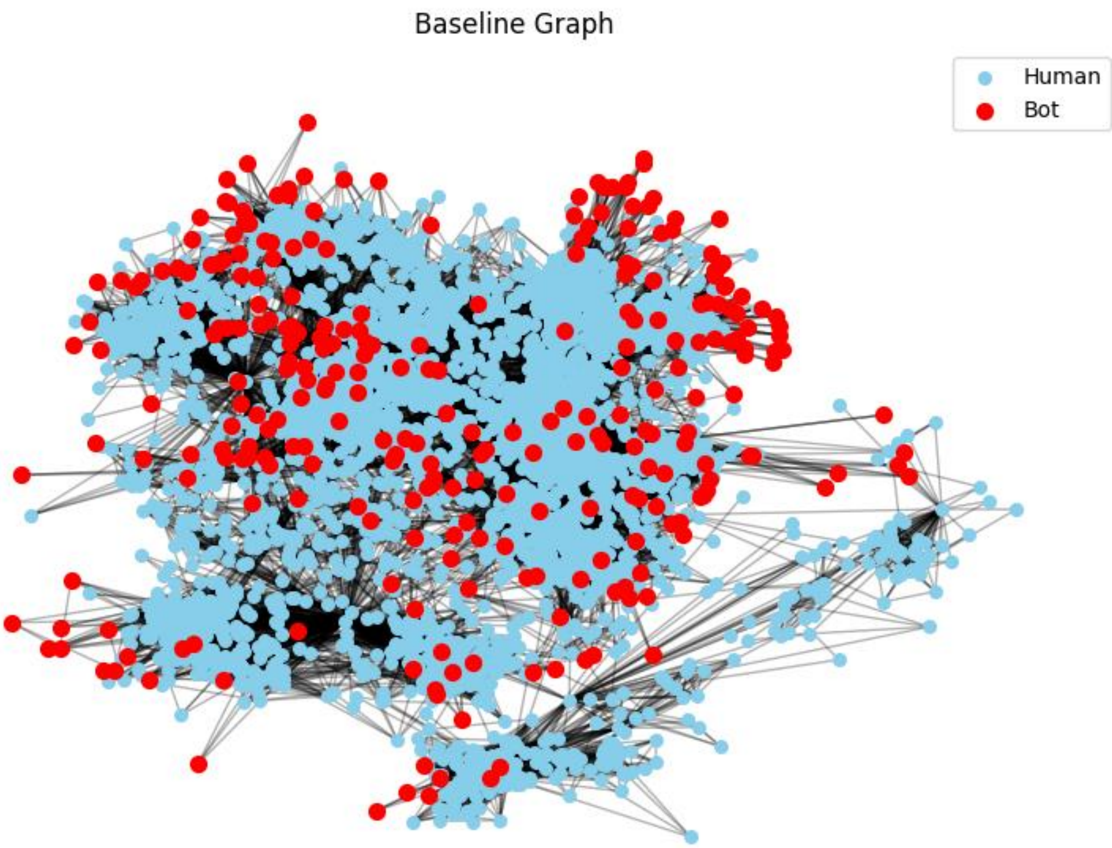
5. Baseline Scenario (No Attack):

We train and test the model using the original clean graph.

The output from the code:

- ROC ≈ 0.9225
- PR-AUC ≈ 0.8548
- F1-Score ≈ 0.8247

This represents the detector's performance when **no** attacker is modifying the graph.



---

6. **Structural Evasion Attack:**

❖ In this attack, bot nodes change their connections to “look more normal".
For example:

- Bots add edges to high-degree real users
- Bots remove links between themselves to avoid being clustered

This makes their graph features closer to humans.

❖ After applying the attack, we repeat:

- feature extraction
- model evaluation

❖ **We do NOT retrain the model.**

❖ **<u>Expected behavior/output</u>**

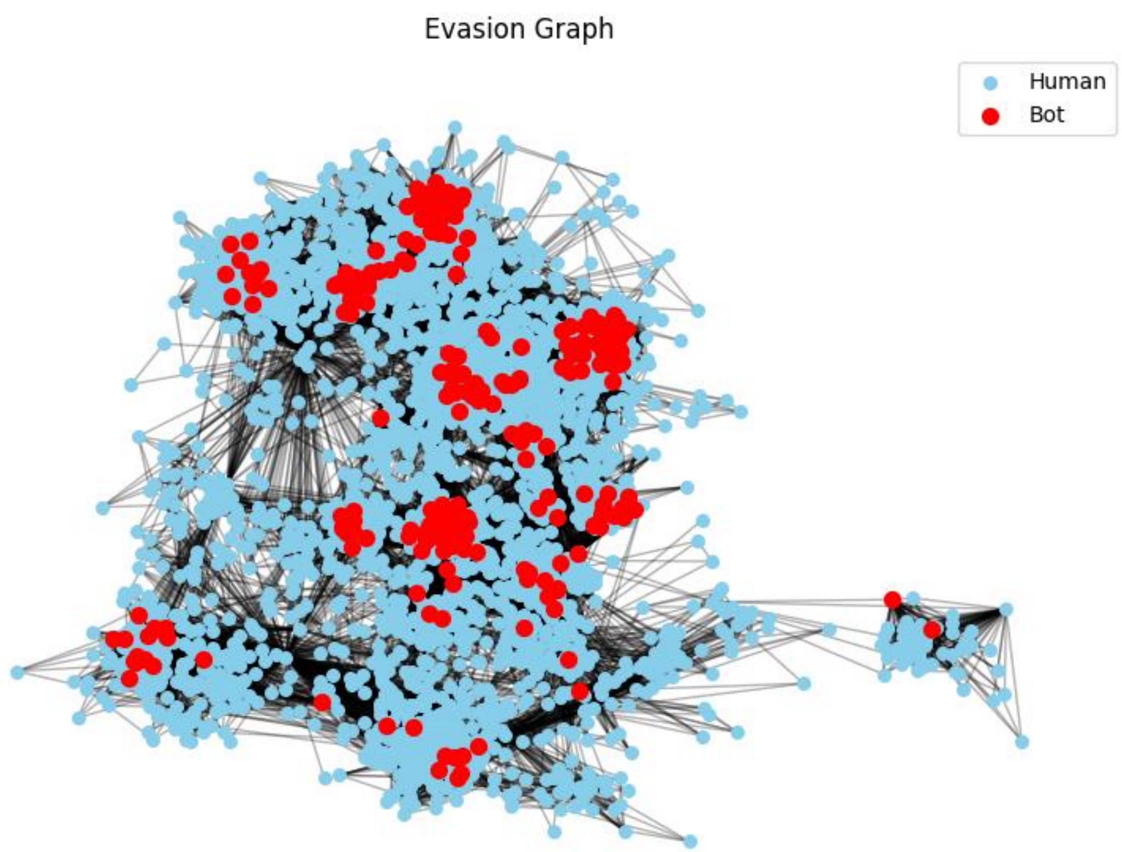
- ROC drops ≈ 0.7873
- PR-AUC drops ≈ 0.2917
- F1 decreases ≈ 0.0351

Because the bots now look more like normal users.

❖ **<u>Code:</u>**

```
#---------- Realistic Structural Evasion Attack (Mimicry - FIXED & WORKING) ----------
G_evasion = G_baseline.copy()
bots_set_evasion = bots_set_baseline.copy()

print("Applying REALISTIC Structural Evasion Attack – Bots mimic real humans 
perfectly")

for bot in bots_set_evasion:
  valid_humans = [n for n in G_evasion.nodes()
          if n not in bots_set_evasion
          and 35 <= G_evasion.degree(n) <= 120]
  if not valid_humans:
    continue
  human_template = random.choice(valid_humans)

  #Delete all old links
  G_evasion.remove_edges_from([(bot, neigh) for neigh in 
G_evasion.neighbors(bot)])

  # Add the human being's neighbors in an almost identical way
  human_neighbors = list(G_evasion.neighbors(human_template))
  target_neighbors = list(human_neighbors)
  random.shuffle(target_neighbors)
```

```
    desired_degree = len(human_neighbors)
    num_to_add = random.randint(int(desired_degree * 0.8), int(desired_degree *
1.15))
    num_to_add = min(num_to_add, len(target_neighbors))

    for target in target_neighbors[:num_to_add]:
        G_evasion.add_edge(bot, target)

print(f"Evasion Attack completed — {len(bots_set_evasion)} bots are now invisible")
```



---

7. **Graph Poisoning Attack:**

- ❖ In a poisoning attack, the attacker modifies the training graph before the model is trained.
- ❖ This allows bots to influence the model to learn incorrect patterns.

❖ Usually, poisoning causes:

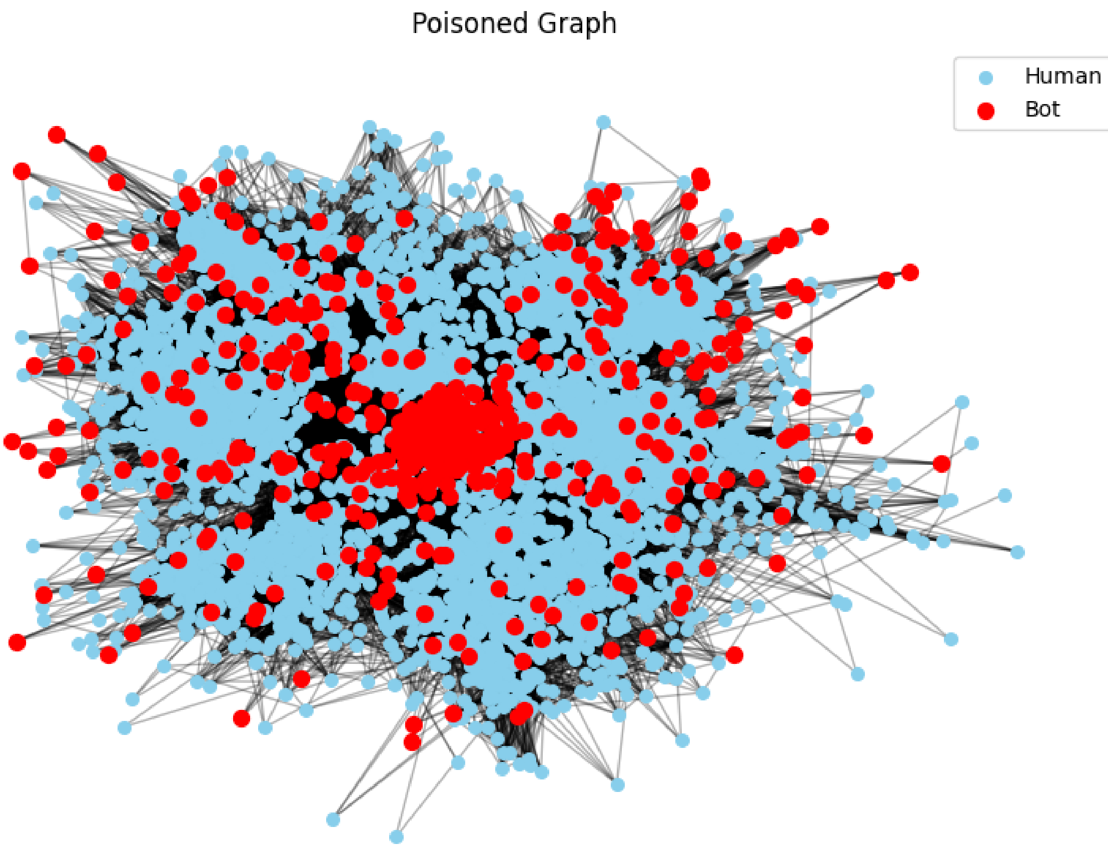
- Large drop in ROC
- Model failing to detect bots
- Higher false positives

❖ **Expected behavior/output "Training: After Poisoning (Retrained)":**

- ROC drops ≈ 0.9371
- PR-AUC drops ≈ 0.8460
- F1 decreases ≈ 0.7020

❖ **Code:**

```
# ---------- Graph Poisoning Attack ----------
G_poison = G_baseline.copy()
bots_set_poison = bots_set_baseline.copy()


print("Injecting 200 medium-degree poison bots...")
for i in range(200):
  new_bot = max(G_poison.nodes()) + 1 + i
  G_poison.add_node(new_bot)

  # poison bots with human-like degree → confuse the model
  deg = random.randint(20, 60)
  targets = random.sample(list(G_poison.nodes()), deg)

  for t in targets:
    G_poison.add_edge(new_bot, t)

  bots_set_poison.add(new_bot)

print("Poisoning completed")
```



---

8. After-Poisoning Test on Clean Data:

We also test the poisoned model on the **clean graph** again.

This shows how much the poisoning corrupted the detector.

❖ **Expected result**

- Huge decrease in performance
- Bot detection becomes unreliable

❖ **<u>Expected behavior/output "Evaluation Only (NO retraining): Poisoned Model → Clean Data":</u>**

- ROC drops ≈ 0.8911
- PR-AUC drops ≈ 0.6784
- F1 decreases ≈ 0.5195

<u>Main observations:</u>

- Structural evasion makes bots "blend in", so performance drops moderately.
- Poisoning attacks corrupt the learning process, so the detector fails badly.
- After poisoning, the model becomes unreliable even on clean data.

---

9. <u>Conclusion:</u>

This assignment shows that bot detection based only on graph structure is vulnerable to adversarial attacks.

- **Structural evasion** makes bots hide inside the network.
- **Poisoning attacks** are even more dangerous because they corrupt the training process.
- To improve robustness, future models should include:

- adversarial training
- anomaly detection
- temporal behavior analysis
