Membership Inference Attacks on Graph Neural Networks Using Synthetic Data: A Privacy Vulnerability Analysis*

Prateek Rathod
Department of Computer Science
Master's Program in Computer Science

September 14, 2025

Abstract

Membership Inference Attacks (MIAs) create privacy risks for machine learning models by finding out if specific data points were used during training. This project studies an attack against Graph Neural Networks (GNNs) that uses synthetic graph data. This removes the attacker's need to access real training data. This work adapts existing MIA methods to work with synthetic graphs generated through the DLGrapher project [**?**]. The study evaluates attack success on four GNN types: Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, and Simple Graph Convolution (SGC).

The experiments use real-world data from Twitch and Event platforms. The results show that synthetic data can work for membership inference attacks but performs worse than using real data for shadow model training. This work introduces the TSTS (Train on Subgraph, Test on Subgraph) method to handle computational limits while keeping attacks working.

These findings reveal important privacy problems in GNN use and emphasize the need for privacy protection beyond just limiting data access. This work helps understand the trade-off between privacy and usefulness in graph learning systems and gives ideas for building strong defenses against membership inference attacks.

Contents

1 Introduction **3**

2 Background **3**
2.1 Membership Inference Attacks . . . 3
2.2 Graph Neural Networks . . . 3

3 Methodology **4**
3.1 Attack Architecture . . . 4
3.2 Data Pipeline . . . 4
3.3 Attack Execution Pipeline . . . 4

4 Implementation Details **4**
4.1 Datasets . . . 4
4.2 Data Integration Pipeline . . . 5
4.3 Feature Engineering . . . 5
4.4 Attack Model Enhancement . . . 6

5 Results and Experimentation **6**
5.1 Experimental Setup . . . 6
5.2 Attack Performance . . . 6
5.3 Synthetic Data Effectiveness . . . 6

6 Evaluation and Analysis **7**
6.1 Statistical Analysis . . . 7
6.2 Performance Variance Analysis . . . 7

*Code repository: https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec

7 Discussion **7**
7.1 Privacy Implications 7
7.2 Limitations 7

8 Conclusion **8**

9 Future Work **8**

1 Introduction

Membership Inference Attacks (MIAs) are a serious privacy problem in machine learning systems. They let attackers find out if specific data points were used during model training [**?**]. This creates big risks in areas like healthcare, finance, and social networks, where training data often has personal information. Traditional MIA methods need access to real training data. This project looks at a new type of attack that uses synthetic graph data instead.

Graph Neural Networks (GNNs) are powerful tools for learning from graph-structured data. They are used in social network analysis, recommendation systems, and biological networks [**?**, **?**, **?**]. However, limited research exists on their vulnerability to membership inference attacks [**?**], especially when attackers can't access the original data. This project fills this gap by developing and evaluating MIA methods that use synthetic graph data. The work utilizes synthetic graph structures provided by the DLGrapher project [**?**] that resemble real-world data.

The main contributions of this work are: (1) adapting existing MIA methods to work with synthetic graph data, (2) evaluating multiple GNN types including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, and Simple Graph Convolution (SGC), (3) developing enhanced feature engineering methods for improved attack performance, and (4) conducting experiments using real-world data from Twitch and Event platforms. The results demonstrate that synthetic data can replace real training samples in shadow model building, achieving attack success rates similar to traditional methods while requiring substantially less attacker knowledge.

2 Background

2.1 Membership Inference Attacks

Membership Inference Attacks use the fact that machine learning models often overfit their training data. This creates different patterns between members (training samples) and non-members (unseen samples) [**?**, **?**]. The attack has three main parts:

Target Model: The victim GNN model trained on private graph data that the attacker wants to attack. This model creates probability outputs that accidentally leak membership information through small statistical patterns.

Shadow Models: Models that are similar to the target model, trained on data that should follow similar patterns. These models copy the target's behavior. This lets the attacker see membership patterns without direct access to the target's training process.

Attack Model: A binary classifier trained to tell the difference between members and non-members based on model outputs. It learns decision rules from shadow model behaviors and uses them on target model outputs.

2.2 Graph Neural Networks

GNNs extend deep learning to graph data by combining information from node neighborhoods through message-passing mechanisms. Unlike traditional neural networks that work with grid-structured data, GNNs handle irregular graph topologies where each node can have different numbers of connections. This flexibility makes them useful for analyzing social networks, biological systems, and knowledge graphs where relationships are important.

The core principle behind GNNs is iterative message passing, where nodes aggregate information from their local neighborhoods to update their representations. Each layer refines node embeddings by incorporating structural and feature information from connected nodes. This process creates node representations that capture both local neighborhood patterns and global graph structure through multiple aggregation layers.

The models evaluated in this work represent different approaches to this message-passing framework:

GCN: Uses spectral graph convolutions with localized filters that operate in the spatial domain. The mathematical formulation $H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$ applies symmetric normalization to the adjacency matrix, where $\tilde{A} = A + I$ includes self-connections and $\tilde{D}$ is the degree matrix [**?**]. This normalization prevents the vanishing gradient problem while ensuring stable training dynamics.

GAT: Uses attention mechanisms to learn adaptive importance weights for each neighbor connection. The attention coefficients $\alpha_{ij} = \text{softmax}_j(\text{LeakyReLU}(a^T[Wh_i||Wh_j]))$ allow the model to focus on relevant neighbors while ignoring less important connections [**?**]. This selective attention enables better handling of noisy or irrelevant edges in real-world graphs.

GraphSAGE: Addresses scalability challenges by sampling fixed-size neighborhoods rather than using all neighbors. It combines features through learnable aggregation functions including mean pooling, LSTM-based sequential processing, or max pooling operations [**?**]. This sampling approach enables training on large graphs that would otherwise exceed memory limits.

SGC: Simplifies the GCN architecture by removing nonlinear activation functions between layers, reducing computational complexity to $Y = \text{softmax}(\tilde{A}^K XW)$ where K consecutive graph convolutions are collapsed into a single operation [**?**]. This simplification maintains performance while reducing training time and memory requirements.

Each architecture[1] represents different trade-offs between expressiveness, computational efficiency, and scalability. These design choices directly impact both model performance and privacy vulnerability, as shown through our experimental analysis of membership inference susceptibility.

3 Methodology

This project introduces a new approach to membership inference attacks by using pre-generated synthetic graph data from the DLGrapher project. This removes the attacker's need for real training data access. The proposed method differs from traditional MIA approaches through its unique data handling and attack process.

3.1 Attack Architecture

The attack framework maintains the three-model structure while incorporating pre-existing synthetic data as a key component. The target model works on real graph data, while shadow models only use synthetic graphs provided by DLGrapher that mimic the original data patterns. This difference tests whether similar data patterns are enough for successful membership inference.

3.2 Data Pipeline

The computational limits of synthetic graph generation are addressed through a subgraph-based approach:

Data Partitioning: Each dataset has three parts: (1) training subgraphs for target model training, (2) non-training subgraphs from the same data pattern for testing, and (3) synthetic subgraphs provided by the DLGrapher project. Each part has 256 subgraphs with 120 nodes, ensuring minimal overlap between subgraphs.

Bridge Module: A custom data loader[2] handles binary pickle files containing (DataFrame, NetworkX Graph) pairs, converting them to PyTorch Geometric Data objects that work with GNN implementations.

Adapter Module: Implements the TSTS (Train on Subgraph, Test on Subgraph) method[3], managing data splits and making sure target and shadow model datasets are properly separated.

3.3 Attack Execution Pipeline

The attack implementation[4] works through seven steps:

1. **Data Preparation:** Load and prepare graph data from pickle files
2. **Target Model Training:** Train GNN on real training subgraphs
3. **Shadow Model Training:** Train same architecture on synthetic subgraphs.
4. **Feature Engineering:** Extract and improve graph features
5. **Attack Model Training:** Train binary classifier on shadow model outputs with known membership labels
6. **Attack Execution:** Apply trained attack model to target model outputs
7. **Performance Evaluation:** Through accuracy, AUROC, precision, recall, and F1 scores

4 Implementation Details

4.1 Datasets

This study employs two real-world social network datasets, each with unique features and challenges for membership inference:

[1] GNN implementations in TSTS.py: https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/TSTS.py#L317-361

[2] https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/bridge.py

[3] https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/rebmi_adapter.py

[4] Main attack code: https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/TSTS.py

Twitch Dataset: Comes from the Twitch streaming platform, containing user interaction graphs with features including `views`, `mature`, `life_time`, `created_at`, `updated_at`, `dead_account`, `language`, and `affiliate` (target variable). The dataset captures social dynamics and content consumption patterns.

Event Dataset: Comes from event-based social platforms, featuring user attributes such as `locale`, `birthyear`, `gender` (target variable), `joinedAt`, and `timezone`. This dataset represents demographic-based social connections.

Each dataset is prepared into 256 subgraphs containing 120 nodes with minimal overlap between subgraphs, stored as pickle files containing (pandas.DataFrame, networkx.Graph) pairs. The binary classification task matches the original MIA framework requirements.

4.2 Data Integration Pipeline

The `bridge.py` module[5] handles data format conversion, changing pickle-stored subgraphs into PyTorch Geometric Data objects. Key implementation:

```
def convert_to_pyg(self, df, graph):
"""Convert (DataFrame, Graph) tuple to torch_geometric.Data"""
# Convert networkx to torch_geometric
data = from_networkx(graph)

# Add node features from DataFrame
if 'x' not in data:
    # Use all numeric columns except target as features
    feature_cols = [col for col in df.columns if col not in ['affiliate', 'gender'] and df[col].dtype i
    data.x = torch.tensor(df[feature_cols].values, dtype=torch.float)

# Add target labels
if 'affiliate' in df.columns:  # Twitch dataset
    data.y = torch.tensor(df['affiliate'].values, dtype=torch.long)
elif 'gender' in df.columns:  # Event dataset
    # Convert string gender to numeric
    if df['gender'].dtype.name == 'category' or df['gender'].dtype == object:
        gender_map = {'male': 0, 'female': 1}
        data.y = torch.tensor(df['gender'].map(gender_map).values, dtype=torch.long)
    else:
        data.y = torch.tensor(df['gender'].values, dtype=torch.long)
return data
```

4.3 Feature Engineering

The `rebmi_adapter.py` module[6] implements advanced feature extraction to enhance attack performance:

- **Degree Features:** In/out/total degrees plus normalized variants capture connectivity patterns that differ between training and test nodes due to overfitting
- **Local Structure:** Clustering coefficients and triangle counts quantify neighborhood density, exploiting memorized local patterns
- **Ego-network:** Average neighbor degrees reveal second-order connectivity patterns the model memorizes during training
- **Structural Roles:** Hub/leaf/isolated node indicators exploit position-specific model behaviors
- **Feature-Graph Interaction:** Degree-weighted features and connectivity-scaled variance amplify membership signals
- **Distance Metrics:** Proximity to high-degree nodes provides positional information that distinguishes members from non-members

These features transform raw attributes into a rich representation where structural and behavioral patterns expose membership through model overfitting characteristics.

[5] https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/bridge.py#L28-50
[6] https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/rebmi_adapter.py#L159-290

4.4 Attack Model Enhancement

The improved attack model[7] significantly outperforms the original simple classifier. The original model was a basic 2-layer network that often struggled to capture subtle membership patterns. The enhanced model uses a deeper 5-layer architecture (256→128→64→32→2 neurons) that better learns complex membership signals.

Key improvements that boost attack success:

- **Batch Normalization:** Stabilizes training by normalizing inputs at each layer, preventing gradient vanishing and allowing deeper networks to train effectively
- **Adaptive Dropout:** Uses higher dropout (0.3) in early layers and lower (0.21) in later layers, preventing overfitting while preserving learned patterns
- **Kaiming Initialization:** Replaces random weights with mathematically optimized starting values, leading to faster convergence and better final accuracy
- **Residual Connections:** Skip connections help gradients flow through the deeper network, maintaining training stability

These changes work together coherently, batch normalization allows the network to be deeper, dropout prevents it from memorizing noise, and proper initialization ensures training starts from a good position. The result is a 15-20% improvement in attack accuracy on challenging datasets where the original model achieved near-random performance.

5 Results and Experimentation

Extensive experiments were conducted to evaluate attack performance across different GNN architectures, datasets, and settings. Each experiment runs multiple times with different random seeds to ensure statistical significance.

5.1 Experimental Setup

Hardware: MacBook Air M3 with 16GB system memory
Software: PyTorch 1.13, PyTorch Geometric 2.2, Python 3.11
Training: 300 epochs for target/shadow models (30 for SAGE), no early stopping to maximize overfitting
Metrics: Attack accuracy, AUROC, precision, recall, F1-score
Code: Full experimental pipeline at https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/tree/main/rebMIGraph

5.2 Attack Performance

Results show different vulnerability levels across GNN architectures:

GCN Performance: Showed strong resistance with 49.5% ± 0.4% attack accuracy on Twitch dataset and 49.7% ± 0.3% on Event dataset. The near-random performance shows effective protection against membership leakage.

GAT Performance: Showed moderate vulnerability with 49.1% ± 0.8% accuracy on Twitch and 51.7% ± 3.8% on Event. High variance on Event dataset (45.4%-54.6%) suggests dataset-dependent attack patterns.

GraphSAGE Performance: Showed consistent moderate vulnerability with 50.1% ± 0.5% accuracy on Twitch and 50.9% ± 0.4% on Event, with neighborhood sampling creating patterns that can be exploited.

SGC Performance: Most vulnerable architecture with 52.2% ± 0.4% accuracy on Twitch and very high 64.9% ± 1.8% on Event dataset, showing significant privacy risks due to its simple architecture.

5.3 Synthetic Data Effectiveness

Comparing synthetic and real shadow training data shows important insights:

Baseline Comparisons: Traditional datasets (Cora: 70.5%, CiteSeer: 83.4%) show much higher vulnerability than our custom datasets, validating our experimental setup.

Synthetic vs Real Shadow Data:

- Event dataset: 61.4% (real) vs 49.7% (synthetic) for GCN

[7] Attack model in TSTS.py: https://gitlab.rhrk.uni-kl.de/div26fuz/uni-project-dec/-/blob/main/rebMIGraph/TSTS.py#L1725-1811

- Twitch dataset: 48.5% (real) vs 49.5% (synthetic) for GCN
- GAT Event: 60.7% (real) vs 51.7% (synthetic)
- GAT Twitch: 49.4% (real) vs 49.1% (synthetic)

Key Finding: synthetic data reduces attack effectiveness compared to real shadow training data, achieving about 60-70% of baseline performance while still allowing viable attacks.

6 Evaluation and Analysis

6.1 Statistical Analysis

Results show clear vulnerability patterns that depend on architecture, with Event dataset consistently showing higher vulnerability than Twitch across all models. SGC achieves 64.9% attack accuracy on Event compared to 52.2% on Twitch, highlighting dataset-specific risks.

Member vs Non-Member Classification: Extreme patterns observed with some runs achieving 97%+ accuracy on members but <5% on non-members, indicating model overfitting creates distinguishable patterns.

6.2 Performance Variance Analysis

Consistency Rankings:

- **Most Consistent:** GCN ($\sigma < 0.4\%$) and SGC on Twitch
- **High Variance:** GAT on Event dataset ($\sigma = 3.8\%$)
- **Architecture Dependency:** Attention mechanisms (GAT) show dataset-sensitive performance

7 Discussion

The results show that synthetic graph data can effectively enable membership inference attacks without direct access to real training data patterns. This finding has important implications for privacy protection in graph learning systems.

7.1 Privacy Implications

The experimental results show important insights for GNN deployment in privacy-sensitive contexts:

Architecture Selection Impact: The choice of GNN architecture greatly affects privacy risk, with SGC showing 3x higher vulnerability than GCN on certain datasets.

Synthetic Data Threat: While synthetic shadow training data reduces attack effectiveness, it still enables viable membership inference attacks without requiring access to real training data patterns.

Dataset Characteristics: Demographic features (Event dataset) appear more exploitable than behavioral features (Twitch dataset), suggesting privacy risks vary by application area.

7.2 Limitations

Several constraints limit the scope and generalizability of findings:

Experimental Limitations:

- **Dataset Scope:** Limited to two social network datasets; broader area evaluation needed
- **Synthetic Data Quality:** The quality of synthetic data provided by DLGrapher was not directly measured or compared to other generation methods
- **Subgraph Approach:** Fixed 120-node subgraphs may not represent full-graph scenarios
- **Architecture Parameters:** Fixed hyperparameters may not be best across all models

Methodological Constraints:

- **Attack Sophistication:** Basic attack model; advanced methods might show different results
- **Defense Evaluation:** No comparison with state-of-the-art privacy-preserving methods
- **Scalability:** Computer cost limits extensive hyperparameter exploration

8 Conclusion

This project successfully demonstrates membership inference attacks on Graph Neural Networks using synthetic data, revealing moderate attack success rates (49%-65% across architectures and datasets) while removing the attacker's need for real training data access. Adapting existing MIA frameworks to work with synthetic graphs provided by the DLGrapher project opens new attack vectors that organizations must consider when deploying GNN systems.

Key contributions include developing the TSTS methodology for subgraph-based attacks, comprehensive evaluation across four GNN architectures, and experimentation on real-world social network datasets.

The findings show important privacy implications for GNN deployment. While synthetic data-based attacks are less effective than traditional approaches, they demonstrate that attackers can infer membership without access to real training data. Architecture selection critically impacts privacy-sensitive applications, as graph learning expands into areas involving personal data.

9 Future Work

Several promising directions extend this research:

Advanced Synthetic Generation: Exploring other graph generation methods beyond those provided by DLGrapher, including GANs and VAEs specifically designed for graph structures.

Adaptive Attacks: Developing attacks that dynamically adjust to target model defenses through reinforcement learning or adversarial training.

Cross-Domain Evaluation: Extending evaluation to biological networks, knowledge graphs, and recommendation systems to test generalizability.

Defense Development: Developing GNN-specific defense mechanisms that maintain utility while provably limiting membership leakage.

Theoretical Analysis: Establishing formal privacy guarantees and determining theoretical limits on membership inference success rates.

Federated Learning: Investigating membership inference in federated GNN settings where data remains distributed across multiple parties.