HỌC VIỆN KỸ THUẬT QUÂN SỰ

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

CHUYÊN NGÀNH: CÔNG NGHỆ THÔNG TIN

NGHIÊN CỨU CÁC GIẢI PHÁP MẠNG NƠ RON ĐỒ THỊ TRONG BÀI TOÁN PHÂN LOẠI TÀI KHOẢN MẠNG XÃ HỘI

NĂM 2025.

MỤC LỤC

BẢNG CHỮ VIẾT TẮT

TỪ VIẾT TẮT	TỪ ĐẦY ĐỦ
Adam	Adaptive Moment Estimation
API	Application Programming Interface
AUC	Area Under the Curve
CNN	Convolutional Neural Network
ELU	Exponential Linear Unit
GAE	Graph Autoencoder
GAN	Generative Adversarial Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
Graph RNN	Graph Recurrent Neural Network
Graph RL	Graph Reinforcement Learning
GraphSAGE	Graph Sample and Aggregate
GRU	Gated Recurrent Unit
ICML	International Conference on Machine Learning
IoT	Internet of Things
Isomap	Isometric Mapping
KNN	K Nearest Neighbors
L2-norm	Euclidean norm
LE	Laplacian Eigenmaps
LeakyReLU	Leaky Rectified Linear Unit
LLE	Locally Linear Embedding
LPP	Locality Preserving Projection
LSTM	Long Short-Term Memory
MDS	Multidimensional Scaling
MLP	Multilayer Perceptron
NeurIPS	Conference on Neural Information Processing Systems
NLP	Natural Language Processing

TỪ VIẾT TẮT	TỪ ĐẦY ĐỦ
PyG	PyTorch Geometric
ReLU	Rectified Linear Unit
R-GCN	Relational Graph Convolutional Network
RMSProp	Root Mean Square Propagation
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VAE	Variational Autoencoder

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

MỞ ĐẦU

1. Tính cấp thiết, lý do lựa chọn đồ án

Trong thời đại kỷ nguyên số phát triển mạnh mẽ, mạng xã hội đã trở thành không gian không chỉ để giao tiếp và chia sẻ thông tin mà còn là môi trường diễn ra các hoạt động tuyên truyền, truyền thông thậm chí là các hình thức đấu tranh bảo vệ an ninh chính trị quốc gia, trật tự an toàn xã hội. Sự bùng nổ về số lượng người dùng và dữ liệu khiến mạng xã hội trở thành mục tiêu, đồng thời là công cụ cho nhiều loại hình hoạt động nguy hại. Một trong số đó là việc tạo ra vô số tài khoản giả mạo nhằm mục đích lừa đảo, đánh cắp thông tin cá nhân, phát tán thông tin sai lệch, xuyên tạc, kích động, chia rẽ hay các tài khoản được tổ chức bài bản nhằm phục vụ chiến tranh thông tin và tác chiến không gian mạng.

Đối với các cơ quan nhà nước, quân đội và lực lượng an ninh, khả năng nhận diện nhanh chóng và chính xác các tài khoản độc hại là yếu tố then chốt để kịp thời ngăn chặn các chiến dịch tấn công thông tin, phát hiện mạng lưới hoạt động bất hợp pháp, bảo vệ uy tín của tổ chức cũng như chủ quyền số của quốc gia. Tuy nhiên, các phương pháp phân loại tài khoản truyền thống chủ yếu dựa trên thông tin hồ sơ hoặc nội dung đăng tải chưa khai thác sâu mối quan hệ kết nối giữa các tài khoản, trong khi đây là dấu hiệu quan trọng để phát hiện các cụm tài khoản có liên kết chặt chẽ và phối hợp hành động.

Mạng nơ-ron đồ thị (Graph Neural Network - GNN) xuất hiện như một hướng tiếp cận hiện đại, cho phép mô hình học được không chỉ đặc trưng của từng tài khoản, mà còn cả mối quan hệ và cấu trúc mạng xã hội mà tài khoản đó tham gia. Việc ứng dụng GNN vào bài toán phân loại tài khoản mạng xã hội không chỉ mang ý nghĩa khoa học, đóng góp vào lĩnh vực trí tuệ nhân tạo và học máy, mà còn có ý nghĩa chiến lược đối với an ninh - quốc phòng.

Lựa chọn nghiên cứu đồ án ***"Nghiên cứu các giải pháp mạng nơ ron đồ thị trong bài toán phân loại tài khoản mạng xã hội"*** xuất phát từ nhu cầu cấp thiết bảo vệ an ninh thông tin trong không gian mạng, đặc biệt là trong bối cảnh các thế lực thù địch ngày càng gia tăng hoạt động tấn công và thao túng thông tin trên mạng xã hội. Đồ án đồng thời phù hợp với định hướng chuyển đổi số quốc gia, góp phần nâng cao năng lực phân tích và xử lý dữ liệu mạng xã hội của các

cơ quan chức năng. Bên cạnh đó, đây là một lĩnh vực nghiên cứu còn khá mới mẻ ở Việt Nam, chưa có nhiều nghiên cứu chuyên sâu ở quy mô học thuật. Nội dung nghiên cứu này đề xuất một cách tiếp cận hiện đại, tận dụng sức mạnh của mạng nơ-ron đồ thị để phát hiện và phân loại tài khoản độc hại một cách hiệu quả hơn, qua đó đóng góp thiết thực vào công tác bảo vệ an ninh thông tin và củng cố chủ quyền số quốc gia.

2. Mục tiêu đồ án

a) Mục tiêu tổng quát

Nghiên cứu các giải pháp và đánh giá hiệu quả một số mô hình phân loại tài khoản mạng xã hội sử dụng GNN nhằm xác định tài khoản thật - giả. Đề xuất hướng cải tiến và ứng dụng thực tiễn, đảm bảo khả năng mở rộng và triển khai trong môi trường thực tế của các cơ quan chức năng.

b) Mục tiêu cụ thể

- Nghiên cứu đặc điểm dữ liệu mạng xã hội và các phương pháp phân loại tài khoản truyền thống.

- Tìm hiểu và phân tích một số kiến trúc phổ biến của GNN như GCN, GraphSAGE, GAT… đánh giá ưu, nhược điểm của từng kiến trúc trong bối cảnh bài toán.

- Thu thập bộ dữ liệu tài khoản mạng xã hội có gắn nhãn (Real - Fake) bao gồm thông tin thuộc tính tài khoản và mối quan hệ kết nối giữa chúng.

- Cài đặt, huấn luyện và tối ưu một số mô hình GNN cho bài toán phân loại tài khoản mạng xã hội, khai thác cả đặc trưng cá nhân và đặc trưng quan hệ trong mạng xã hội.

- Đánh giá hiệu quả thông qua các chỉ số đo lường như Accuracy, Precision, Recall, F1-score và so sánh với các phương pháp phân loại truyền thống.

3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đồ án là các kiến trúc mô hình GNN phổ biến cùng với dữ liệu tài khoản mạng xã hội bao gồm thông tin thuộc tính của tài khoản và mối quan hệ liên kết giữa các tài khoản.

Phạm vi nghiên cứu tập trung vào việc áp dụng và đánh giá hiệu quả các kiến trúc mô hình GNN phổ biến như GCN, GAT và GraphSAGE trong bài toán

phân loại tài khoản thật - giả trên mạng xã hội. Việc triển khai được thực hiện trong môi trường mô phỏng và đánh giá, chưa áp dụng ở quy mô giám sát thực tế. Dữ liệu phục vụ nghiên cứu được thu thập từ bộ dữ liệu Cresci-2015 đã được công bố rộng rãi và sử dụng phổ biến trong các nghiên cứu quốc tế, bảo đảm tính minh bạch và khả năng tái lập.

4. Phương pháp nghiên cứu

Đồ án kết hợp phương pháp nghiên cứu lý thuyết và thực nghiệm. Tiến hành tìm hiểu cơ sở lý thuyết về GNN và các kiến trúc tiêu biểu như GCN, GAT, GraphSAGE, đồng thời khảo sát các phương pháp phân loại tài khoản truyền thống để làm cơ sở so sánh. Sử dụng bộ dữ liệu Cresci-2015, thực hiện tiền xử lý, xây dựng đồ thị mạng xã hội và trích xuất đặc trưng. Trên dữ liệu này, tiến hành huấn luyện và đánh giá các mô hình GCN, GAT, GraphSAGE theo các chỉ số Accuracy, Precision, Recall, F1-score. So sánh kết quả và rút ra kết luận khả năng ứng dụng các giải pháp mạng nơ-ron đồ thị đối với bài toán phân loại tài khoản mạng xã hội.

5. Ý nghĩa khoa học và thực tiễn

Đồ án góp phần bổ sung và làm rõ khả năng ứng dụng mạng nơ-ron đồ thị trong bài toán phân loại tài khoản mạng xã hội, một hướng nghiên cứu còn tương đối mới ở Việt Nam. Kết quả nghiên cứu cung cấp cơ sở lý thuyết và thực nghiệm cho việc khai thác đồng thời đặc trưng cá nhân và đặc trưng quan hệ của tài khoản thay vì chỉ dựa vào thông tin cá nhân như các phương pháp truyền thống. Ngoài ra, việc so sánh giữa các kiến trúc GCN, GAT và GraphSAGE cũng như các mô hình học máy truyền thống khác giúp xác định mô hình phù hợp nhất cho từng loại dữ liệu.

Kết quả của đồ án có thể làm nền tảng cho việc phát triển các hệ thống giám sát và phát hiện tự động tài khoản giả mạo trên mạng xã hội, hỗ trợ các cơ quan quản lý, tổ chức và doanh nghiệp trong công tác bảo vệ an toàn thông tin, phòng chống lừa đảo, ngăn chặn các chiến dịch tấn công và thao túng thông tin.

6. Cấu trúc đồ án

Ngoài phần Mở đầu, Kết luận, Tài liệu tham khảo, nội dung đồ án được tổ chức thành 3 chương chính như sau:

Chương 1: Tổng quan về mạng nơ-ron đồ thị và ứng dụng

Giới thiệu các khái niệm cơ bản về lý thuyết đồ thị, các phương pháp biểu diễn dữ liệu dạng đồ thị và tổng quan những nghiên cứu, ứng dụng GNN trong nhiều lĩnh vực.

Chương 2: Phân tích các mô hình Graph Neural Network phù hợp với bài toán phân loại tài khoản mạng xã hội

Đặt vấn đề về sự cần thiết và ưu điểm nổi bật của ứng dụng GNN trong nhận diện tài khoản mạng xã hội. Trình bày lịch sử ra đời và phát triển của GNN. Phân tích một số kiến trúc tiêu biểu như GCN, GAT, GraphSAGE. Tổng hợp và đánh giá hiệu năng của các kiến trúc này dựa trên các nghiên cứu thực nghiệm đã công bố trong lĩnh vực phân loại tài khoản mạng xã hội.

Chương 3: Cài đặt, huấn luyện mô hình, đánh giá hiệu quả

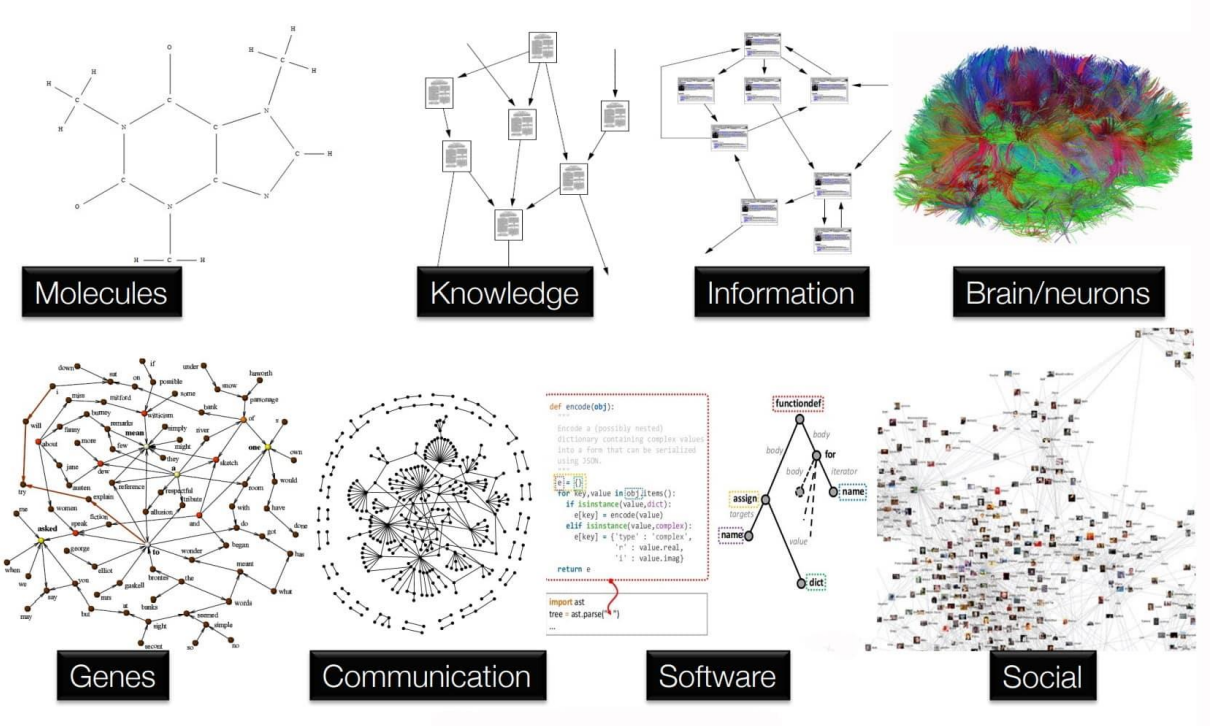
Mô tả quy trình chuẩn bị và tiền xử lý dữ liệu Cresci-2015, xây dựng đồ thị mạng xã hội, thiết lập tham số, huấn luyện các mô hình GNN, đồng thời trình bày kết quả thực nghiệm, so sánh hiệu suất giữa các mô hình cũng như các mô hình học máy truyền thống khác từ đó rút ra nhận xét, đề xuất hướng cải thiện.

Do thời gian và kiến thức về lĩnh vực nghiên cứu cũng như kinh nghiệm bản thân còn nhiều hạn chế nên đồ án chắc chắn không tránh khỏi những thiếu sót. Vì vậy, tôi rất mong nhận được sự bổ sung, góp ý của các thầy, cô giáo để giúp tôi hoàn thiện hơn những thiếu sót mà đồ án chưa đạt được. Tôi xin chân thành cảm ơn ThS. Nguyễn Chí Công, ThS. Vi Bảo Ngọc cùng các thầy, cô tại Viện CNTT&TT đã tạo điều kiện giúp đỡ và chỉ bảo tôi trong suốt quá trình thực hiện đồ án tốt nghiệp này.

Chương 1:
TỔNG QUAN VỀ MẠNG NƠ-RON ĐỒ THỊ VÀ ỨNG DỤNG

1.1. Khái quát về mạng nơ-ron đồ thị

Mạng nơ-ron đồ thị (GNN) là một lớp mô hình học sâu (Deep Learning) đặc biệt được thiết kế để xử lý dữ liệu có cấu trúc đồ thị, trong đó các đỉnh (node) biểu diễn các thực thể và các cạnh (edge) biểu diễn mối quan hệ giữa chúng. Khác với mạng nơ-ron truyền thống chỉ xử lý dữ liệu dạng lưới (hình ảnh, video…) hoặc chuỗi (văn bản, email…), GNN khai thác cấu trúc quan hệ phức tạp giữa các đỉnh thông qua cơ chế lan truyền thông tin (message passing). Ở mỗi lớp của GNN, mỗi đỉnh sẽ thu thập thuộc tính (feature) từ chính nó và các đỉnh lân cận, sau đó thực hiện phép kết hợp (aggregation) theo các hàm như tổng, trung bình hoặc hàm chú ý (attention) để tạo ra biểu diễn mới cho đỉnh đó. Quá trình này lặp lại qua nhiều lớp, giúp mỗi đỉnh tích hợp thông tin từ một vùng lân cận ngày càng rộng hơn, từ đó nâng cao khả năng học hiểu cấu trúc và tính chất toàn cục của đồ thị.



Hình 1.1: Các miền ứng dụng của GNN

GNN thường áp dụng phương pháp học có giám sát (Supervised Learning) với hàm mất mát dựa trên nhiệm vụ như phân loại đỉnh, phân loại cạnh hoặc dự đoán nhãn đồ thị... Ngoài ra, học bán giám sát (Semi-Supervised Learning) cũng

là xu hướng phổ biến, tận dụng cả các đỉnh đã gán nhãn và chưa gán nhãn thông qua kỹ thuật propagation label. Đối với các ứng dụng đòi hỏi suy luận cấu trúc, GNN có thể kết hợp với mô hình sinh như VAE hoặc GAN để mô phỏng và tạo ra đồ thị mới.

GNN đã chứng minh hiệu quả vượt trội trong nhiều lĩnh vực như phân tích mạng xã hội (dự đoán liên kết, phát hiện cộng đồng…), sinh học (dự đoán tương tác protein, xác định cấu trúc phân tử…), hóa học tính toán (dự đoán tính chất vật liệu, thuốc…), hệ thống khuyến nghị (recommendation systems) và trong xử lý ngôn ngữ tự nhiên (cấu trúc phụ thuộc, semantic graph…). Những ứng dụng này cho thấy khả năng mô hình hóa và tổng quát hóa mạnh mẽ của GNN khi phải xử lý dữ liệu phi cấu trúc thuần túy và có quan hệ phức tạp.

1.1.1. Cơ sở lý thuyết về đồ thị

Đồ thị (graph) là một đối tượng cơ bản trong toán học và khoa học máy tính, được định nghĩa dưới dạng $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, trong đó $\boldsymbol{V}$ là tập hợp các đỉnh và $\boldsymbol{E}$ là tập hợp các cạnh nối giữa các đỉnh. Với đồ thị vô hướng (undirected), mỗi cạnh là một cặp đỉnh không thứ tự, với đồ thị có hướng (directed), mỗi cạnh được coi là một cặp đỉnh có thứ tự, có một điểm đầu và một điểm cuối. Mỗi đỉnh và cạnh đều là những thành phần cơ bản của đồ thị, và ta có thể gán thêm trọng số (weight) cho mỗi cạnh trong các bài toán tối ưu hóa đi.

**** Thành phần cơ bản của đồ thị***

Đỉnh (Node): Mỗi đỉnh là một điểm cơ bản trong đồ thị. Tập hợp các đỉnh của đồ thị được ký hiệu là $\boldsymbol{V}$.

Cạnh (Edge): Mỗi cạnh nối hai đỉnh và là phần tử của tập $\boldsymbol{E}$. Trong đồ thị có hướng, cạnh còn được gọi là cung (arc) và có chiều cụ thể giữa một cặp đỉnh. Trong đồ thị vô hướng, cạnh được biểu diễn bằng một cặp đỉnh không có chiều. Với đồ thị khuyên (loop) thì cạnh có thể nối một đỉnh với chính nó; Với đồ thị có cạnh song song có thể có nhiều cạnh cùng nối một cặp đỉnh.

Hướng (Directed/Undirected): Đồ thị vô hướng là đồ thị mà các cạnh không phân biệt chiều; đồ thị có hướng thì mỗi cạnh có chiều cụ thể. Đồ thị hỗn hợp có thể chứa cả hai loại cạnh.

Trọng số (Weight): Trong đồ thị có trọng số, mỗi cạnh được gán một giá trị (trọng số, độ dài, chi phí,...) thể hiện tiêu chí quan tâm (như chi phí di chuyển, độ mạnh kết nối…). Đồ thị không trọng số thì có thể coi mỗi cạnh có trọng số mặc định bằng 1 hoặc chỉ quan tâm đến sự tồn tại của cạnh.

Chu trình (Cycle) và đỉnh liền kề: Một chu trình là một đường đi khép kín đi qua các đỉnh (bắt đầu và kết thúc tại cùng một đỉnh) mà không lặp lại cạnh. Đồ thị có chu trình gọi là đồ thị có chu trình (cyclic), ngược lại nếu không có chu trình thì đồ thị đó là vô chu trình (acyclic). Hai đỉnh được gọi là kề nhau nếu giữa chúng có một cạnh; tương tự, hai cạnh được gọi là kề nhau nếu chúng cùng gặp tại một đỉnh.

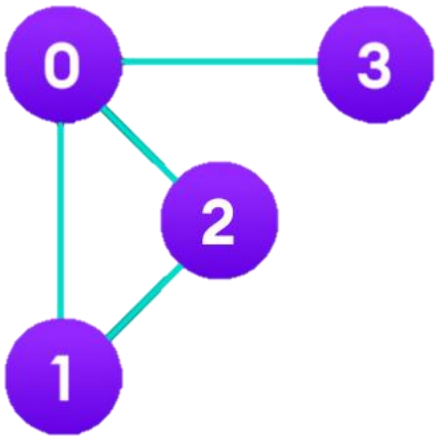
Tính liên thông: Đồ thị liên thông (connected) là đồ thị mà tồn tại đường đi giữa mọi cặp đỉnh bất kỳ. Nếu đồ thị không thỏa mãn điều kiện này thì gọi là rời rạc (disconnected).

**** Phân loại các loại đồ thị***

Đơn đồ thị (Simple graph): Là đồ thị không chứa khuyên và không có cạnh song song giữa cùng một cặp đỉnh. Mỗi cặp đỉnh chỉ được nối bởi tối đa một cạnh đơn.

Đa đồ thị (Multigraph): Là đồ thị không thỏa mãn điều kiện đơn đồ thị, tức có thể có nhiều cạnh song song giữa một cặp đỉnh hoặc có khuyên.

Đồ thị hai phía (Bipartite graph): Là đồ thị mà tập đỉnh $\mathcal{V}$ có thể phân thành hai tập con không giao nhau sao cho không có cạnh nào nối hai đỉnh trong cùng một tập. Đồ thị hai phía thường dùng để biểu diễn các mối quan hệ giữa hai nhóm đối tượng khác nhau (ví dụ: nam - nữ trong hôn nhân, khách hàng - đơn hàng trong mạng lưới giao dịch).



Hình 1.2: Đồ thị vô hướng với $\boldsymbol{\mathcal{V}} = \{0,1,2,3\}$, $\boldsymbol{\mathcal{E}} = \{\{0,1\},\{0,2\},\{0,3\},\{1,2\}\}$

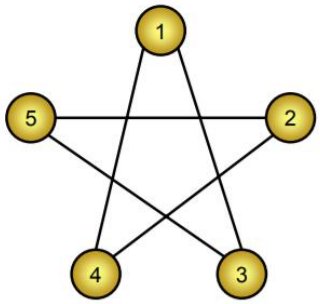
1.1.2. Mô hình hoá dữ liệu đồ thị trong học máy

a. Biểu diễn ma trận kề (Adjacency Matrix)

Ma trận kề là một ma trận vuông kích thước $\boldsymbol{n} \times \boldsymbol{n}$ (với n là số lượng đỉnh của đồ thị) dùng để biểu diễn quan hệ kề giữa các đỉnh. Mỗi phần tử $\boldsymbol{A_{ij}}$ trong ma trận cho biết sự kết nối giữa đỉnh $\boldsymbol{i}$ và đỉnh $\boldsymbol{j}$. Theo định nghĩa, $\boldsymbol{A_{ij}} = 1$ nếu có cạnh nối từ đỉnh $\boldsymbol{i}$ tới đỉnh $\boldsymbol{j}$, và $\boldsymbol{A_{ij}} = 0$ nếu giữa $\boldsymbol{i}$ và $\boldsymbol{j}$ không có cạnh nối. Trong trường hợp đồ thị vô hướng, ma trận kề sẽ đối xứng quanh đường chéo chính ($\boldsymbol{A_{ij}} = \boldsymbol{A_{ji}}$). Nếu đồ thị có trọng số, giá trị 1 có thể được thay bằng trọng số của cạnh tương ứng. Ngược lại, với đồ thị có hướng, ma trận kề có thể không đối xứng và $\boldsymbol{A_{ij}} = 1$ chỉ khi có cung từ $\boldsymbol{i}$ đến $\boldsymbol{j}$.

Để xây dựng ma trận kề từ dữ liệu đồ thị, trước tiên ta đánh số các đỉnh từ 1 đến n. Khởi tạo một ma trận $\boldsymbol{n} \times \boldsymbol{n}$ với tất cả phần tử bằng 0. Sau đó, duyệt qua danh sách các cạnh của đồ thị. Với mỗi cạnh $(\boldsymbol{u}, \boldsymbol{v})$ trong tập cạnh $\boldsymbol{\mathcal{E}}$, $\boldsymbol{A_{uv}} = 1$ (đồng thời $\boldsymbol{A_{vu}} = 1$ nếu đồ thị vô hướng). Quá trình này đảm bảo rằng những cặp đỉnh có kết nối sẽ được đánh dấu bằng 1 trong ma trận. Nếu đồ thị cho phép cạnh tự nối (self-loop) hoặc nhiều cạnh song song giữa hai đỉnh, các giá trị trên đường chéo hoặc giá trị $\boldsymbol{A_{ij}}$ có thể được điều chỉnh tương ứng (ví dụ: đếm số cạnh giữa

i và j thay vì chỉ đánh dấu 1). Thông thường, đối với đồ thị đơn không có cạnh tự nối, ta giữ các phần tử đường chéo $\boldsymbol{A_{ii} = 0}$ cho mọi đỉnh $\boldsymbol{i}$.



$$A=\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Hình 1.3: Biểu diễn đồ thị bằng ma trận kề

b. Biểu diễn ma trận đặc trưng đỉnh (Feature Matrix)

Bên cạnh ma trận kề mô tả cấu trúc đồ thị, mỗi đỉnh thường đi kèm một vector đặc trưng mô tả các thuộc tính hoặc thông tin liên quan đến đỉnh đó. Tập hợp các vector đặc trưng của toàn bộ $\boldsymbol{n}$ đỉnh có thể được tổ chức thành một ma trận đặc trưng đỉnh $\boldsymbol{X}$ kích thước $\boldsymbol{n} \times \boldsymbol{f}$, trong đó $\boldsymbol{f}$ là số lượng đặc trưng mô tả mỗi đỉnh. Nói cách khác, ma trận đặc trưng đỉnh (thường ký hiệu là $\boldsymbol{X}$) có mỗi hàng tương ứng với một đỉnh và mỗi cột tương ứng với một thuộc tính đặc trưng. Ví dụ, trong một đồ thị với $\boldsymbol{N = 6}$ đỉnh, nếu mỗi đỉnh được mô tả bởi $\boldsymbol{F = 4}$ đặc trưng, thì $\boldsymbol{X}$ sẽ là ma trận 6×4 chứa toàn bộ thuộc tính của các đỉnh đó. Các đặc trưng này có thể là bất kỳ thông tin gì gắn với đỉnh, tùy thuộc vào bài toán thực tế đang mô hình hóa. Việc xác định đặc trưng đỉnh phụ thuộc vào loại dữ liệu đồ thị và bài toán cần giải. Một số ví dụ điển hình gồm:

Mạng xã hội: Mỗi đỉnh đại diện cho một người dùng, có thể gắn các thuộc tính như tuổi, giới tính, số lượng bài đăng, số bạn bè, sở thích,... Các thuộc tính này được mã hóa thành dạng số và xếp thành vector đặc trưng cho từng người dùng. Chẳng hạn, ta có thể dùng một vector 3 chiều để biểu diễn [tuổi, số bài viết, số lượt thích] của mỗi người dùng.

Đồ thị văn bản - tài liệu: Xem mỗi đỉnh là một tài liệu hoặc một từ trong ngữ cảnh đồ thị ngôn ngữ. Khi đó, đặc trưng đỉnh có thể là biểu diễn nội dung của tài liệu (ví dụ: một vector độ dài cố định biểu thị tần suất các từ khóa hoặc vector word embedding của tài liệu đó).

Đồ thị khoa học (hóa học, sinh học): Đỉnh có thể đại diện cho phân tử hoặc nguyên tử trong đồ thị phân tử. Đặc trưng của một nguyên tử có thể bao gồm các thuộc tính hóa học như số hiệu nguyên tử, khối lượng nguyên tử, điện tích, loại nguyên tố,… Những thuộc tính này được mã hóa (ví dụ dưới dạng số thực hoặc nhị phân) và tạo thành vector đặc trưng cho mỗi nguyên tử.

Ma trận $\boldsymbol{X}$ cung cấp đầu vào ban đầu về nội dung của các đỉnh cho mô hình học. Trong mạng nơ-ron đồ thị, ta không chỉ quan tâm đến cấu trúc kết nối mà còn cần tận dụng thông tin nội tại của từng đỉnh. Các thuật toán GNN sẽ học cách kết hợp đặc trưng của một đỉnh với đặc trưng của các đỉnh láng giềng (thông qua ma trận kề) để tạo ra biểu diễn mới cho đỉnh đó ở các lớp sau. Có thể hiểu, $\boldsymbol{X}$ giống như dữ liệu đầu vào ban đầu cho mỗi node, và qua mỗi lớp GNN, ta thu được một ma trận đặc trưng mới (thường gọi là ma trận embedding $\boldsymbol{H}$) với cùng số hàng nhưng số cột (chiều không gian đặc trưng) có thể thay đổi theo số lượng ẩn của lớp. Mục tiêu của quá trình này là học được biểu diễn ẩn cho mỗi đỉnh sao cho biểu diễn đó phản ánh cả thuộc tính ban đầu lẫn bối cảnh hàng xóm của đỉnh trong đồ thị. Nói một cách khác, ma trận đặc trưng đỉnh chứa thông tin định danh và thuộc tính của các thực thể (đỉnh) trong đồ thị, giúp mô hình GNN phân biệt giữa các đỉnh và cung cấp cơ sở để tổng hợp thông tin.

Ma trận kề và ma trận đặc trưng đỉnh có vai trò quan trọng trong các bài toán, các mô hình học máy liên quan đến đồ thị vì nó mã hoá cấu trúc liên kết của đồ thị, quyết định cách thức lan truyền thông tin giữa các đỉnh láng giềng. Cụ thể, ma trận kề thường được sử dụng để tổng hợp thông tin từ các đỉnh lân cận. Ví dụ, nếu $\boldsymbol{A}$ là ma trận kề và $\boldsymbol{X}$ là ma trận đặc trưng, thì phép nhân ma trận $\boldsymbol{A} \times \boldsymbol{X}$ sẽ tạo ra một ma trận đặc trưng mới mà mỗi hàng $\boldsymbol{i}$ là tổng các vector đặc trưng của những đỉnh kề với đỉnh $\boldsymbol{i}$ (bao gồm cả $\boldsymbol{i}$ nếu có thêm cạnh tự nối). Nói cách khác, phần tử $\boldsymbol{A_{ij}}$ (0 hoặc 1) khi nhân với $\boldsymbol{X}$ sẽ quyết định việc đưa đặc trưng của đỉnh $\boldsymbol{j}$ đóng góp vào đỉnh $\boldsymbol{i}$ trong quá trình lan truyền thông tin. Nhờ đó, mô hình GNN có thể tổng hợp thông tin, mỗi đỉnh cập nhật biểu diễn của mình dựa trên đặc trưng của các đỉnh liên kết trực tiếp với nó.

Trong thực tế, để mô hình GNN học hiệu quả hơn, ma trận kề thường được điều chỉnh trước khi đưa vào thuật toán. Một kỹ thuật phổ biến là thêm self-loop cho mỗi đỉnh - tương đương với việc đặt $\boldsymbol{A_{ii}} = 1$ cho mọi đỉnh $\boldsymbol{i}$ (thêm ma trận đơn vị $\boldsymbol{I}$ vào $\boldsymbol{A}$) nhằm bảo đảm mỗi đỉnh cũng truyền thông tin của chính nó vào quá trình tổng hợp đặc trưng. Ma trận kề cũng có thể được chuẩn hóa để tránh trường hợp các đỉnh có nhiều láng giềng đóng góp quá lớn. Tóm lại, ma trận kề là thành phần không thể thiếu trong các thuật toán GNN, giúp mạng nơ-ron đồ thị biết được cấu trúc nào của đồ thị cần được khai thác khi cập nhật trạng thái của các đỉnh.

1.2. Học biểu diễn đồ thị (Graph Representation Learning)

Hiện nay nhiều hệ thống phức tạp có dạng đồ thị, chẳng hạn như mạng xã hội, mạng sinh học và mạng thông tin… Các nhà khoa học đều thừa nhận rằng dữ liệu đồ thị thường rất phức tạp do đó khó xử lý. Để xử lý dữ liệu đồ thị một cách hiệu quả, thách thức quan trọng đầu tiên là tìm được phương pháp biểu diễn dữ liệu đồ thị hiệu quả, tức là cách biểu diễn đồ thị một cách ngắn gọn để các tác vụ phân tích nâng cao, chẳng hạn như khám phá mẫu, phân tích và dự đoán, có thể được thực hiện một cách hiệu quả cả về thời gian và không gian.

Một đồ thị thường được biểu diễn dưới dạng $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, trong đó $\boldsymbol{V}$ là tập các đỉnh và $\boldsymbol{E}$ là tập các cạnh. Đối với các đồ thị lớn, chẳng hạn như đồ thị có hàng tỷ đỉnh, phương pháp biểu diễn truyền thống đặt ra nhiều thách thức cho việc xử lý và phân tích đồ thị.

- Độ phức tạp tính toán cao: Các mối quan hệ được mã hoá bởi tập cạnh $\boldsymbol{E}$ đòi hỏi hầu hết các thuật toán xử lý hay phân tích đồ thị phải thực hiện các bước tính toán lặp hoặc tổ hợp. Ví dụ, một cách phổ biến là sử dụng độ dài đường đi ngắn nhất hoặc trung bình giữa hai đỉnh để biểu diễn khoảng cách của chúng. Để tính khoảng cách này theo biểu diễn truyền thống, ta phải liệt kê rất nhiều đường đi khả dĩ giữa hai đỉnh, vốn về bản chất là một bài toán tổ hợp. Các phương pháp như vậy dẫn đến độ phức tạp tính toán cao, khiến chúng không thể áp dụng cho các đồ thị quy mô lớn trong thực tế.

- **Khả năng song song hóa thấp:** Tính toán song song và phân tán đã trở thành phương pháp mặc định để xử lý và phân tích dữ liệu quy mô lớn. Tuy nhiên, dữ liệu đồ thị khi được biểu diễn theo cách truyền thống lại gặp khó khăn lớn trong việc thiết kế và triển khai các thuật toán song song và phân tán. Đỉnh trong đồ thị bị ràng buộc với nhau thông qua $\mathcal{E}$, do đó khi phân tán các tập con đỉnh sang các phân vùng hoặc máy chủ khác nhau, chi phí giao tiếp cao giữa các máy chủ trở nên rất nặng nề và làm hạn chế tỉ lệ tăng tốc.

- **Không khả dụng cho các phương pháp học máy:** Gần đây, các phương pháp học máy, đặc biệt là học sâu, tỏ ra rất mạnh mẽ ở nhiều lĩnh vực. Tuy nhiên, khi dữ liệu đồ thị được biểu diễn theo cách truyền thống, hầu hết các phương pháp học máy có sẵn không thể áp dụng được. Những phương pháp này thường giả định rằng các mẫu dữ liệu có thể được biểu diễn như các vector độc lập trong một không gian vector, trong khi các mẫu trong dữ liệu đồ thị (tức là các đỉnh) lại phụ thuộc lẫn nhau ở mức độ nhất định do $\mathcal{E}$ xác định. Mặc dù ta có thể đơn giản biểu diễn một đỉnh bằng vector hàng tương ứng trong ma trận kề của đồ thị, song kích thước cực lớn của vector đó trong đồ thị có nhiều đỉnh khiến việc xử lý và phân tích đồ thị sau đó trở nên khó khăn.

Để giải quyết những thách thức này, cần phát triển phương pháp học biểu diễn đồ thị mới, tức là học các biểu diễn vectơ chiều thấp liên tục và dày đặc cho các đỉnh, để có thể giảm nhiễu hoặc thông tin dư thừa và bảo toàn thông tin cấu trúc nội tại. Trong không gian biểu diễn đã học, các mối quan hệ giữa các đỉnh, ban đầu được biểu diễn bằng các cạnh hoặc các phép đo topo bậc cao khác trong đồ thị, được nắm bắt bằng khoảng cách giữa các đỉnh trong không gian vectơ và các đặc điểm cấu trúc của một đỉnh được mã hóa vào vectơ biểu diễn của nó. Về cơ bản, để không gian biểu diễn có thể hỗ trợ tốt các tác vụ phân tích đồ thị, học biểu diễn đồ thị cần đạt hai mục tiêu:

- **Phục hồi lại đồ thị gốc:** Nếu có một cạnh hoặc quan hệ giữa hai đỉnh, thì khoảng cách giữa hai vector biểu diễn tương ứng phải tương đối nhỏ, sao cho đồ thị ban đầu có thể được khôi phục từ không gian biểu diễn.

- Hỗ trợ hiệu quả suy luận trên đồ thị: Không gian biểu diễn phải giúp dự đoán các cạnh chưa biết, xác định các đỉnh quan trọng, và suy luận nhãn đỉnh. Chỉ đạt mục tiêu khôi phục đồ thị thôi chưa đủ để hỗ trợ suy luận. Sau khi có biểu diễn, các tác vụ mức thấp như phân loại đỉnh, phân cụm đỉnh, trực quan hóa đồ thị và dự đoán liên kết có thể được giải quyết dựa trên các biểu diễn này.

1.2.1. Biểu diễn đồ thị truyền thống

Các phương pháp biểu diễn đồ thị truyền thống ban đầu được nghiên cứu như các kỹ thuật giảm chiều. Một đồ thị thường được xây dựng từ một tập dữ liệu được biểu diễn bởi các đặc trưng. Như đã đề cập trước đó, biểu diễn đồ thị thường có hai mục tiêu, đó là tái tạo cấu trúc đồ thị gốc và hỗ trợ suy luận trên đồ thị. Các hàm mục tiêu của phương pháp biểu diễn đồ thị truyền thống chủ yếu hướng tới mục tiêu tái tạo đồ thị.

Cụ thể, đầu tiên xây dựng một đồ thị lân cận $\boldsymbol{\mathcal{G}}$ bằng các thuật toán kết nối như KNN. Tiếp theo, dựa trên $\boldsymbol{\mathcal{G}}$, có thể tính toán đường đi ngắn nhất giữa các điểm dữ liệu khác nhau. Kết quả là, đối với tất cả N điểm dữ liệu trong tập, ta có ma trận khoảng cách trên đồ thị. Cuối cùng, phương pháp đa chiều cổ điển MDS được áp dụng lên ma trận này để thu được các vector tọa độ. Các biểu diễn mà Isomap học được gần đúng bảo toàn khoảng cách giữa các cặp điểm trong không gian có chiều thấp. Vấn đề then chốt của Isomap là độ phức tạp cao do việc tính toán đường đi ngắn nhất cho từng cặp điểm.

Phương pháp LLE được đề xuất nhằm loại bỏ việc phải ước tính khoảng cách giữa các cặp điểm cách xa nhau. LLE giả định rằng mỗi điểm và các điểm lân cận của nó nằm trên hoặc gần một vùng tuyến tính cục bộ của đa tạp. Để đặc trưng hóa hình học cục bộ, mỗi điểm có thể được tái tạo từ các điểm lân cận. Cuối cùng, trong không gian có chiều thấp, LLE xây dựng một ánh xạ bảo toàn lân cận dựa trên tái tạo tuyến tính cục bộ.

Trong khi đó, phương pháp LE cũng bắt đầu với việc xây dựng đồ thị sử dụng vùng lân cận $\boldsymbol{\varepsilon}$ hoặc KNN. Tiếp theo, hàm Heat Kernel được sử dụng xác định trọng số giữa hai đỉnh trong đồ thị. Cuối cùng, các biểu diễn đỉnh có thể thu

được bằng cách dựa trên quy tắc hóa ma trận Laplacian. Hơn nữa, phép chiếu bảo toàn tính cục bộ LPP một xấp xỉ tuyến tính của LE phi tuyến, cũng được đề xuất.

1.2.2. Mạng nơ-ron trên đồ thị

Trong thập kỷ qua, học sâu đã trở thành "viên ngọc quý" của trí tuệ nhân tạo và học máy, cho thấy hiệu suất vượt trội trong âm thanh, hình ảnh và xử lý ngôn ngữ tự nhiên... Mặc dù rõ ràng là đồ thị tồn tại khắp nơi trong thế giới thực, song việc ứng dụng các phương pháp học sâu để phân tích dữ liệu đồ thị là rất khó. Vấn đề này không hề đơn giản vì các thách thức sau đây:

- **Cấu trúc không đều của đồ thị:** Không giống như hình ảnh, âm thanh và văn bản có cấu trúc lưới rõ ràng, đồ thị có cấu trúc không đều, làm cho việc tổng quát hóa một số phép toán cơ bản lên đồ thị trở nên khó khăn. Ví dụ, định nghĩa các phép tích chập và gộp, là các phép toán nền tảng trong mạng CNN đối với dữ liệu đồ thị là không đơn giản.

- **Tính dị hướng và đa dạng của đồ thị**: Một đồ thị có thể rất phức tạp, chứa nhiêu loại và tính chất khác nhau. Những dạng, tính chất và nhiệm vụ đa dạng này đòi hỏi các kiến trúc mô hình khác nhau để giải quyết các bài toán cụ thể trong thực tiễn.

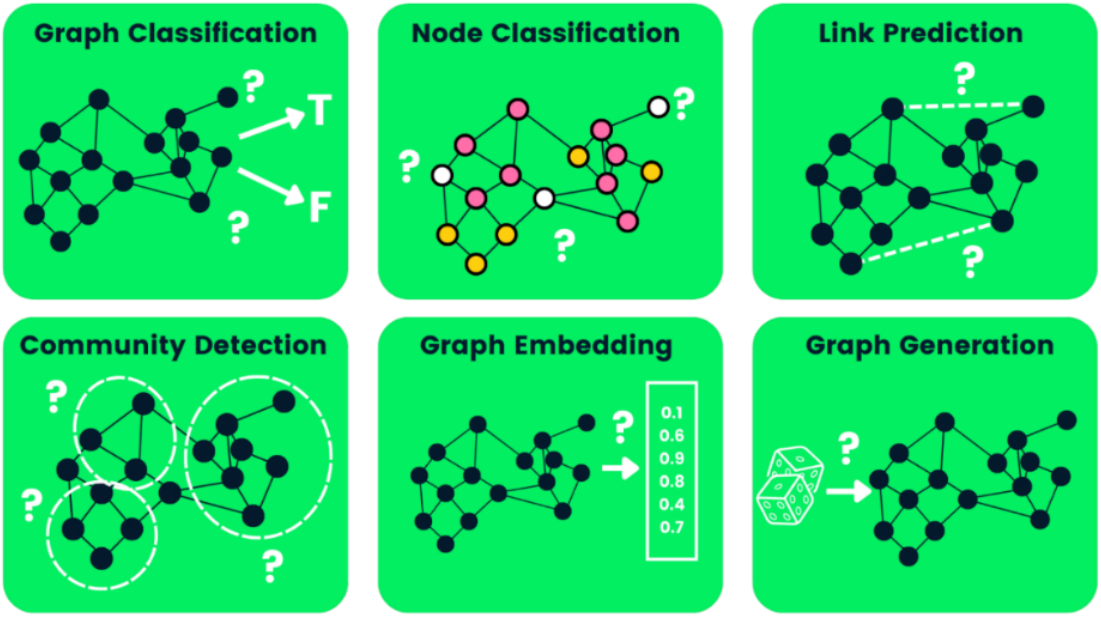
- **Đồ thị có quy mô lớn**: Trong kỷ nguyên dữ liệu lớn, đồ thị thực có thể có hàng triệu hoặc hàng tỷ đỉnh và cạnh. Thiết kế các mô hình có khả năng mở rộng, ưu tiên các mô hình có độ phức tạp thời gian tuyến tính theo kích thước đồ thị, là vấn đề then chốt.

- **Kết hợp kiến thức liên ngành:** Đồ thị thường liên quan đến các ngành khác nhau như sinh học, hóa học và khoa học xã hội… Bản chất liên ngành này đồng thời mang lại cơ hội và thách thức. Kiến thức chuyên môn có thể được tận dụng để giải quyết các bài toán cụ thể nhưng việc tích hợp kiến thức chuyên ngành làm cho thiết kế mô hình trở nên phức tạp.

Hiện nay, mạng nơ-ron trên đồ thị đã thu hút được sự quan tâm đáng kể của giới nghiên cứu trong vài năm gần đây. Các kiến trúc được áp dụng và chiến lược huấn luyện rất đa dạng, từ giám sát đến không giám sát và từ mô hình tích chập

đến mô hình hồi quy, bao gồm các mạng nơ-ron hồi quy trên đồ thị (Graph RNN), mạng nơ-ron tích chập trên đồ thị (GCN), bộ mã hóa tự động trên đồ thị (GAE), học tăng cường trên đồ thị (Graph RL) và các phương pháp đối kháng trên đồ thị. Có nhiều hướng nghiên cứu đang diễn ra hoặc dự kiến trong tương lai cũng rất đáng được tiếp tục nghiên cứu, bao gồm các mô hình mới cho các cấu trúc đồ thị chưa được khảo sát, đồ thị động, khả năng giải thích và tính bền vững... Nhìn chung, học sâu trên đồ thị là một lĩnh vực nghiên cứu đầy hứa hẹn và đang phát triển nhanh chóng, vừa mang đến những cơ hội vừa đặt ra nhiều thách thức. Việc nghiên cứu học sâu trên đồ thị là một bước xây dựng then chốt trong mô hình hoá dữ liệu quan hệ và là một bước quan trọng hướng tới tương lai với các kỹ thuật học máy và trí tuệ nhân tạo ngày càng tốt hơn.

1.3. Ứng dụng của Graph Neural Network trong các lĩnh vực



Hình 1.4: Các bài toán điển hình trong học sâu trên GNN

1.3.1. Mạng xã hội và phân tích người dùng

a. Phân tích tài khoản thật - giả (Real vs. Fake Account Detection)

Phân tích tài khoản thật - giả trên mạng xã hội nhằm xác định và phân loại các tài khoản thật (real) và tài khoản ảo (fake) do con người hoặc bot tạo ra. Trong đồ thị người dùng, mỗi đỉnh đại diện cho một tài khoản, các cạnh biểu diễn mối quan hệ như tương tác, theo dõi hoặc bình luận. Bài toán phát hiện tài khoản mạng

xã hội thật - giả tìm cách học một hàm ánh xạ các đỉnh vào không gian sao cho tài khoản thật và tài khoản giả được phân tách rõ ràng về vị trí. GNN tận dụng cơ chế lan truyền truyền tin giữa các đỉnh láng giềng để thu thập thông tin về hành vi kết nối và đặc trưng nội tại (profile metadata, lịch sử đăng bài, tần suất tương tác…). Qua mỗi lớp truyền tin, GNN không chỉ học được biểu diễn cục bộ dựa trên đặc trưng của đỉnh mà còn khái quát hóa cấu trúc xung quanh, từ đó khuếch đại khác biệt giữa các nhóm tài khoản thật và tài khoản giả. Nhiều nghiên cứu đã áp dụng GNN cho bài toán này, bao gồm cả phương pháp học có giám sát và không giám sát. Kết quả cho thấy GNN hiệu quả cao trong việc giảm tỉ lệ báo cáo sai (false positives) và bỏ sót (false negatives) so với các phương pháp truyền thống chỉ dựa vào đặc trưng đơn luồng (feature-based).

b. Phân tích cộng đồng (Community Detection)

Phân tích cộng đồng trên mạng xã hội nhằm xác định các nhóm người dùng có mức độ kết nối mật thiết với nhau. Ví dụ, các nhóm bạn bè hay nhóm người dùng có chung sở thích thường tạo thành những cộng đồng. Bài toán cộng đồng tìm cách phân loại các đỉnh trong đồ thị thành các nhóm sao cho bên trong nhóm có nhiều cạnh hơn so với giữa các nhóm. GNN có khả năng học biểu diễn cho từng đỉnh dựa trên cấu trúc lân cận, giúp các đỉnh cùng cộng đồng có biểu diễn tiệm cận nhau. Qua các lớp truyền tin, GNN thu thập thông tin về kết nối và tầng lớp xung quanh, từ đó khuếch đại sự tương đồng của các đỉnh trong cùng một cộng đồng. Các nghiên cứu ghi nhận GNN đã được áp dụng thành công cho bài toán phát hiện cộng đồng, bao gồm cả trường hợp học có giám sát (đã biết một số cấu trúc cộng đồng) và không giám sát.

c. Dự đoán liên kết (Link Prediction)

Bài toán dự đoán liên kết đặt mục tiêu tìm các cạnh tiềm năng giữa hai đỉnh chưa có kết nối hiện tại. Ví dụ, tính năng “gợi ý kết bạn” trên Facebook, Twitter, Zalo… dự đoán người dùng nào có khả năng kết bạn với nhau dựa trên mạng lưới hiện tại. Tương tự, trong hệ thống đề xuất nội dung, ta có thể xem bài toán gợi ý bạn bè là một dạng đề bài dự đoán liên kết trong đồ thị người dùng. GNN học

được biểu diễn ngầm của người dùng từ cấu trúc mạng xã hội, sau đó sử dụng biểu diễn này để đánh giá khả năng tồn tại của một liên kết mới. Thông qua quá trình truyền tin, GNN tự động tổng hợp thông tin của các đỉnh lân cận, nên các đỉnh có kết nối gián tiếp với nhau nhiều khả năng sẽ có biểu diễn gần nhau và được dự đoán thành liên kết. Điều này cho phép GNN nắm bắt ảnh hưởng xã hội và thông tin ngữ cảnh rộng hơn thay vì chỉ dựa trên số lượng bạn chung hay tiêu chí cục bộ. Các nghiên cứu cho thấy GNN có tiềm năng lớn trong học biểu diễn người dùng trên mạng xã hội nhờ cơ chế truyền tin theo lân cận, từ đó hỗ trợ tốt bài toán gợi ý bạn bè.

1.3.2. Phân loại và phát hiện bất thường

a. Phân loại đồ thị sinh học

Trong sinh học và hóa học, nhiều đối tượng được biểu diễn dưới dạng đồ thị (ví dụ phân tử hóa học, mạng tương tác protein…). Bài toán phân loại đồ thị (graph classification) là gán nhãn cho cả đồ thị, chẳng hạn phân loại phân tử theo tính chất vật lý hoặc hoạt tính sinh học (độc tính, khả năng tương tác thuốc…). Mỗi phân tử có thể coi là một đồ thị với các đỉnh là nguyên tử và các cạnh là liên kết hóa học. Ví dụ, bộ dữ liệu MUTAG gồm 188 phân tử nitroaromatic được sử dụng để dự đoán tính đột biến của phân tử. GNN được thiết kế để tận dụng cấu trúc đồ thị và truyền tin giữa các đỉnh kề nhau, do đó có thể học được đặc trưng sâu sắc từ cấu trúc topology của phân tử. Các nghiên cứu chỉ ra GNN thể hiện hiệu quả cao trong dự đoán tính chất phân tử nhờ khai thác thông tin cấu trúc đồ thị. So với phương pháp truyền thống bỏ qua cấu trúc liên kết, GNN có ưu thế trong việc kết hợp đồng thời thông tin thuộc tính đỉnh và liên kết.

b. Phát hiện gian lận tài chính

Trong lĩnh vực tài chính, bài toán phát hiện gian lận (fraud detection) thường liên quan đến mạng lưới giao dịch giữa các tài khoản. Ví dụ, phát hiện giao dịch thẻ tín dụng gian lận, rửa tiền, hoặc các bất thường khác trong mạng lưới giao dịch ngân hàng… Mỗi tài khoản và giao dịch có thể được mô hình hóa như một đỉnh và một cạnh trong đồ thị giao dịch, và gian lận biểu hiện thành các

mô hình kết nối phức tạp trong đồ thị. GNN rất phù hợp vì nó xử lý trực tiếp dữ liệu dạng đồ thị. Thay vì phân tích từng giao dịch riêng lẻ, GNN có thể xem xét mối quan hệ giữa các giao dịch thông qua đồ thị chung. Cụ thể, GNN coi các tài khoản, giao dịch, thiết bị như các đỉnh liên kết với nhau, nhờ đó có thể phát hiện các mô hình gian lận tiềm ẩn trong toàn bộ mạng lưới. Các phương pháp truyền thống chỉ phân tích giao dịch đơn lẻ, trong khi GNN tận dụng ngữ cảnh đồ thị để giảm báo động sai và tăng độ chính xác phát hiện.

1.3.3. Một số ứng dụng khác

Xử lý tài liệu và ngôn ngữ: tài liệu và ngôn ngữ cũng có thể được mô hình hóa dưới dạng đồ thị, trong đó các đỉnh biểu diễn các thực thể như từ, cụm từ, câu hoặc đoạn văn, còn các cạnh thể hiện mối quan hệ ngữ nghĩa, cú pháp hay tham chiếu giữa chúng. GNN đã được áp dụng rộng rãi trong các hệ thống xử lý ngôn ngữ tự nhiên NLP tiên tiến để giải quyết những bài toán như phân loại văn bản, trích xuất quan hệ, sinh tóm tắt tự động, hỏi - đáp và hoàn thiện ngữ cảnh. Nhờ khả năng xử lý hiệu quả cấu trúc đồ thị phức tạp, các mô hình GNN thường đạt hiệu quả hàng đầu trong việc khai thác mối liên kết ngữ nghĩa giữa các thành tố của văn bản. Các mô hình GNN trong NLP tận dụng thông tin lịch sử (ngữ cảnh tuần tự trong câu) và mối liên hệ không gian (đường dẫn ngữ nghĩa giữa các thực thể) để nâng cao độ chính xác cho bài toán. Khi phân loại tài liệu, GNN có thể lan truyền thông tin giữa các đỉnh từ khóa để nhận diện chủ đề. Khi trích xuất quan hệ, GNN sử dụng đường đi ngắn nhất trên đồ thị phụ thuộc cú pháp để xác định chính xác mối quan hệ giữa hai thực thể. Khi tóm tắt, GNN kết hợp cấu trúc câu với trọng số quan trọng để sinh đoạn văn ngắn gọn mà vẫn đầy đủ ý chính. Nhờ đó, các ứng dụng NLP có tích hợp GNN góp phần nâng cao chất lượng hiểu và sinh ngôn ngữ, đáp ứng tốt các yêu cầu của các hệ thống xử lý ngôn ngữ tự nhiên hiện đại.

Giao thông: Mạng lưới giao thông (đường bộ, đường sắt, phương tiện giao thông…) cũng có thể biểu diễn thành đồ thị không gian - thời gian. GNN được áp dụng rộng rãi trong các hệ thống giao thông thông minh để dự báo lưu lượng, điều

khiển tín hiệu và quản lý giao thông. Nhờ xử lý được thông tin không gian phức tạp, GNN đã đạt hiệu quả hàng đầu trong dự báo tình trạng giao thông, ví dụ dự báo lưu lượng và tốc độ dòng xe, dự báo lưu lượng hành khách trên mạng lưới đường sắt đô thị, cũng như nhu cầu gọi xe công cộng. Các mô hình GNN khai thác lịch sử và mối liên hệ giữa các nút giao thông để cung cấp dự báo chính xác, giúp giảm ùn tắc và tối ưu hóa quản lý giao thông.

Năng lượng: Trong ngành năng lượng, lưới điện phân phối và truyền tải cũng tạo thành một đồ thị (các trạm biến áp, máy phát, đường dây). GNN được dùng để dự báo công suất tiêu thụ, cân bằng tải và phát hiện sự cố mạng lưới. Ví dụ, một nghiên cứu trên dữ liệu lưới điện Bắc Âu ứng dụng GNN (GCN kết hợp GRU và biến thể Fourier GNN) để dự báo vị trí công suất, cho hiệu năng cạnh tranh với các mô hình dự báo truyền thống. Điều này cho thấy GNN tận dụng tốt cấu trúc đồ thị vốn có của lưới điện để nâng cao độ chính xác dự báo và giám sát hệ thống năng lượng.

Ngoài ra, GNN còn được ứng dụng trong nhiều lĩnh vực khác như IoT, mạng lưới cảm biến, khuyến nghị sản phẩm và các vấn đề khoa học khác liên quan mạng phức tạp… GNN là một phương pháp hiện đại được phát triển để xử lý hiệu quả các loại dữ liệu có dạng mạng lưới hoặc có quan hệ kết nối phức tạp giữa các phần tử. Trong nhiều loại dữ liệu thực tế, thông tin không chỉ nằm ở bản thân từng đối tượng riêng lẻ mà còn ở cách các đối tượng đó liên kết và tác động qua lại với nhau. GNN nổi bật ở khả năng kết hợp hai loại thông tin này, vừa xem xét đặc điểm riêng của từng điểm trong mạng, vừa phân tích cấu trúc kết nối giữa chúng. Bằng cách "truyền" và "tổng hợp" thông tin qua các mối liên kết, GNN dần hình thành một cái nhìn toàn diện hơn về dữ liệu, từ những chi tiết nhỏ nhất đến các mối quan hệ tổng thể. Điều này cho phép mô hình nhận ra được các quy luật tiềm ẩn, các nhóm dữ liệu có liên quan mật thiết hoặc các tín hiệu bất thường mà nếu chỉ nhìn vào từng phần tử riêng lẻ thì khó nhận ra.

Chương 2:
PHÂN TÍCH CÁC MÔ HÌNH GRAPH NEURAL NETWORK PHÙ HỢP VỚI BÀI TOÁN PHÂN LOẠI TÀI KHOẢN MẠNG XÃ HỘI

2.1. Đặt vấn đề

Trong kỷ nguyên số hiện nay, mạng xã hội đã trở thành một phần không thể thiếu trong đời sống hàng ngày của hàng tỷ người trên toàn thế giới. Đây là môi trường trung tâm cho việc kết nối, giao lưu, chia sẻ thông tin, hình thành các cộng đồng trực tuyến bất kể khoảng cách địa lý hay rào cản ngôn ngữ. Người dùng có thể cập nhật tin tức, bày tỏ quan điểm cá nhân, tham gia thảo luận và tiếp cận nguồn tri thức phong phú chỉ với vài thao tác đơn giản. Tuy nhiên, chính đặc điểm mở, phi tập trung và tính lan truyền nhanh chóng của mạng xã hội lại tạo ra "mảnh đất màu mỡ" cho sự tồn tại và phát triển của các tài khoản ảo. Những tài khoản này được tạo ra không nhằm mục đích kết nối chân chính, mà thường phục vụ cho nhiều hoạt động tiêu cực như phát tán tin giả, quảng bá thông tin sai lệch, kích động chia rẽ, gây nhiễu loạn dư luận, thao túng nhận thức cộng đồng hoặc thậm chí thực hiện các hành vi lừa đảo, chiếm đoạt tài sản. Sự gia tăng và biến hóa ngày càng tinh vi của các tài khoản giả đã đặt ra thách thức lớn đối với cả nền tảng mạng xã hội lẫn người dùng. Việc phân biệt giữa tài khoản thật và tài khoản giả không chỉ là bài toán kỹ thuật đòi hỏi các giải pháp công nghệ hiện đại, mà còn là yêu cầu cấp bách để bảo đảm tính an toàn, sự tin cậy và lành mạnh của môi trường trực tuyến.

Trong những năm qua, nhiều phương pháp học máy truyền thống đã được áp dụng để giải quyết bài toán phân loại tài khoản mạng xã hội, điển hình như: Logistic Regression, Decision Tree, Random Forest, SVM, và các mô hình học sâu dựa trên mạng nơ-ron truyền thống (MLP, CNN)… Các phương pháp này chủ yếu dựa vào các đặc trưng của từng tài khoản riêng lẻ như số lượng bài đăng, số lượng người theo dõi (followers), số người được theo dõi (Followees), tỉ lệ tương tác, thời gian hoạt động, nội dung văn bản trong bài viết hoặc hình ảnh đại diện... Những đặc trưng này sau đó được đưa vào các mô hình học máy để huấn luyện

và phân loại. Tuy nhiên, các phương pháp truyền thống này bộc lộ nhiều hạn chế đáng kể, đặc biệt trong bối cảnh môi trường mạng xã hội ngày càng phức tạp và hành vi của các tài khoản giả mạo ngày càng tinh vi. Một trong những vấn đề lớn nhất là sự phụ thuộc mạnh mẽ vào đặc trưng định danh của từng tài khoản đơn lẻ. Trong thực tế, các tài khoản giả mạo có thể dễ dàng ngụy trang bằng cách giả lập hành vi giống với tài khoản thật như tăng số lượng bài đăng, làm giả ảnh đại diện, bắt chước phong cách ngôn ngữ trong bình luận hay thậm chí tạo ra lịch sử hoạt động trông có vẻ tự nhiên. Điều này làm cho việc dựa vào đặc trưng cá nhân trở nên kém hiệu quả trong phân loại. Hơn nữa, mạng xã hội vốn không chỉ là tập hợp rời rạc của các tài khoản, mà là một hệ thống phức tạp với cấu trúc liên kết rõ rệt như tài khoản này theo dõi tài khoản kia, tương tác qua lại với nhau, bình luận, chia sẻ, nhắn tin... Những thông tin mang tính cấu trúc đóng vai trò cực kỳ quan trọng trong việc hiểu rõ hành vi của người dùng. Các phương pháp học máy truyền thống, vốn chỉ xử lý dữ liệu dạng bảng hoặc dữ liệu chuỗi, thường bỏ qua hoặc không tận dụng được đầy đủ thông tin cấu trúc này. Chẳng hạn, một tài khoản có thể trông giống như tài khoản thật nếu chỉ xét riêng lẻ, nhưng nếu hầu hết bạn bè hoặc người tương tác với nó đều là tài khoản giả, thì khả năng cao chính nó cũng là giả. Đây là một khía cạnh mà các mô hình phi cấu trúc không thể nắm bắt hiệu quả. Ngoài ra, các phương pháp truyền thống thường không có khả năng học từ mối quan hệ lặp lại và lan truyền trong mạng, ví dụ như một tài khoản giả có thể liên tục tạo các tài khoản phụ và xây dựng mạng lưới tương tác giả để đánh lừa hệ thống. Nếu không có cơ chế học trên cấu trúc mạng, mô hình học máy sẽ không thể phát hiện được các cụm hành vi đáng ngờ như vậy.

Trong khi đó, mạng xã hội thực chất là một đồ thị lớn, nơi mỗi người dùng là một đỉnh, các kết nối giữa họ là các cạnh và thông tin lan truyền qua từng cạnh mang theo đặc trưng hành vi, sở thích, nhóm cộng đồng... Chính cấu trúc đồ thị này cung cấp các tín hiệu quan trọng để phân biệt tài khoản thật và giả, đặc biệt khi kẻ tạo tài khoản ảo thường cố gắng sao chép hành vi "bề ngoài" nhưng khó tái hiện chính xác các mô hình tương tác phức hợp trong mạng. GNN ra đời như

một giải pháp tối ưu để khai thác triệt để thông tin cấu trúc và ngữ nghĩa trên đồ thị. Qua cơ chế "message passing" truyền và tổng hợp thông tin giữa các đỉnh lân cận, GNN có khả năng học được biểu diễn (embedding) giàu ý nghĩa cho từng đỉnh, phản ánh không chỉ đặc trưng riêng lẻ mà còn cả mối quan hệ đa chiều với cộng đồng xung quanh. Nhờ vậy, GNN hứa hẹn nâng cao độ chính xác trong việc phát hiện tài khoản giả so với các phương pháp chỉ dựa trên đặc trưng cục bộ. Chương 2 sẽ tập trung phân tích và so sánh các kiến trúc GNN tiêu biểu bao gồm GCN, GAT, GraphSAGE cũng như các biến thể hỗn hợp và đa chiều để đánh giá mức độ phù hợp của chúng với bài toán phân loại tài khoản mạng xã hội. Đồng thời, xem xét các kỹ thuật tối ưu hóa như sampling, clustering, hierarchical pooling… để giải quyết bài toán quy mô lớn và các phương pháp giải thích mô hình nhằm tăng cường tính minh bạch. Mục tiêu cuối cùng là lựa chọn và đề xuất một mô hình GNN có hiệu quả cao, khả năng mở rộng tốt, đồng thời đảm bảo tính giải thích đủ để ứng dụng thực tiễn trong hệ thống phát hiện tài khoản giả trên các nền tảng mạng xã hội.

2.2. Lịch sử ra đời và sự phát triển của Graph Neural Network

GNN là một lớp mô hình học sâu được thiết kế để xử lý dữ liệu có cấu trúc đồ thị, đóng vai trò quan trọng trong các ứng dụng như phân tích mạng xã hội, sinh học, hệ thống khuyến nghị… Sự phát triển của GNN là kết quả của quá trình tiến hóa từ các phương pháp xử lý đồ thị truyền thống đến các kỹ thuật học sâu hiện đại, vượt qua những thách thức về tính toán và khả năng biểu diễn dữ liệu phi cấu trúc. Ý tưởng sơ khai về GNN được đề xuất lần đầu tiên vào năm 2005 bởi *Franco Scarselli* và các cộng sự trong bài báo *"A new model for learning in graph domains"*[1] và được công bố chính thức năm 2009 trong bài báo *"The Graph Neural Network Model"*[2]. Mô hình này giới thiệu ý tưởng lan truyền thông tin (message passing) giữa các đỉnh trong đồ thị, cho phép mỗi đỉnh cập nhật biểu diễn của mình dựa trên đặc trưng của các đỉnh lân cận. GNN ban đầu sử dụng một cơ chế lặp để đạt trạng thái ổn định, nhưng phương pháp này có nhược điểm là tốn kém về tính toán và khó mở rộng cho các đồ thị lớn. Dù vậy, công

trình này đã đặt nền móng lý thuyết cho các phát triển sau này, nhấn mạnh tầm quan trọng của việc kết hợp cấu trúc đồ thị vào quá trình học máy.

Năm 2016, *"Thomas Kipf, Max Welling"* công bố bài báo *"Semi-supervised classification with Graph Convolutional Networks"*[3] đánh dấu một bước ngoặt lớn cho GNN. GCN đơn giản hóa cơ chế lan truyền thông tin bằng cách sử dụng phép tích chập phổ (spectral convolution) dựa trên ma trận Laplace của đồ thị. Mô hình này cho phép học bán giám sát hiệu quả trên đồ thị, tận dụng cả dữ liệu có nhãn và không nhãn. GCN nhanh chóng trở thành tiêu chuẩn vàng nhờ tính đơn giản và hiệu quả mà nó mang lại.

Năm 2017, "*Graph Attention Network"*[4] được *Veličković* và các cộng sự giới thiệu, mang đến cơ chế chú ý (attention mechanism) để gán trọng số linh hoạt cho các đỉnh lân cận, thay vì xử lý đồng đều như GCN. Cùng năm, GraphSAGE của *Hamilton* và cộng sự giới thiệu với bài báo *"Inductive representation learning on large graphs"*[5], tập trung vào học quy nạp (Inductive Learning), cho phép mô hình tổng quát hóa trên các đỉnh mới mà không cần huấn luyện lại. GraphSAGE sử dụng kỹ thuật lấy mẫu lân cận, giúp mở rộng GNN cho các đồ thị lớn như mạng xã hội với hàng tỷ người dùng.

Trong những năm gần đây, GNN tiếp tục phát triển với các biến thể như Relational GCN (R-GCN) cho đồ thị quan hệ, Graph Isomorphism Network (GIN) cho phân loại đồ thị và các mô hình xử lý đồ thị động (dynamic graph). Những tiến bộ này giải quyết các vấn đề như quá mịn (oversmoothing), khả năng mở rộng và xử lý đồ thị không đồng nhất, phù hợp với các ứng dụng thực tế như phát hiện tài khoản giả mạo trong thời gian thực. Các hội nghị lớn như NeurIPS và ICML đã ghi nhận sự gia tăng các nghiên cứu về GNN, đặc biệt trong việc tích hợp GNN với các kỹ thuật học sâu khác như học tăng cường (Reinforcement Learning) hoặc học đối kháng (Adversarial Learning).

2.3. Một số kiến trúc của Graph Neural Network

Ý tưởng cốt lõi của mạng nơ-ron đồ thị là cập nhật tuần tự (lần lượt) các biểu diễn của đỉnh bằng cách kết hợp biểu diễn của các đỉnh lân cận với biểu diễn

của chính nó. Bắt đầu từ biểu diễn đỉnh ban đầu $\boldsymbol{H^0 = X}$, ở mỗi lớp chúng ta có hai hàm quan trọng:

- **Aggregate (hàm tổng hợp)**: hàm này tổng hợp thông tin từ các đỉnh lân cận của mỗi đỉnh.

- **Combine (hàm kết hợp)**: hàm này cập nhật biểu diễn của đỉnh bằng cách kết hợp thông tin đã tổng hợp từ các đỉnh lân cận với biểu diễn hiện tại của đỉnh.

Về mặt toán học, có thể định nghĩa khung tổng quát của mạng nơ-ron đồ thị như sau:

Khởi tạo: $\boldsymbol{H^0 = X}$

Với $\boldsymbol{k = 1, 2, \dots, K}$

$$\boldsymbol{a_v^k = \mathbf{Aggregate}^k\{H_u^{k-1} : u \in N(v)\}} \quad (2.1)$$

$$\boldsymbol{H_v^k = \mathbf{Combine}^k\{H_v^{k-1}, a_v^k\}} \quad (2.2)$$

với $\boldsymbol{N(v)}$ là tập các đỉnh lân cận của đỉnh thứ $\boldsymbol{v}$. Các biểu diễn đỉnh $\boldsymbol{H^K}$ ở lớp cuối cùng có thể được xem như biểu diễn cuối cùng của các đỉnh.

Khi đã có biểu diễn của các đỉnh, chúng có thể được sử dụng cho các tác vụ downstream. Ví dụ phân loại đỉnh, nhãn của đỉnh $\boldsymbol{v}$ (ký hiệu $\boldsymbol{\hat{y}_v}$) có thể được dự đoán thông qua hàm Softmax, tức là

$$\boldsymbol{\hat{y}_v = \mathbf{Softmax}(W H_v^{\top})} \quad (2.3)$$

với $\boldsymbol{W \in \mathbb{R}^{|\mathcal{L}| \times F}}$, và $\boldsymbol{|\mathcal{L}|}$ là số nhãn trong không gian đầu ra.

Với một tập các đỉnh đã được gán nhãn, toàn bộ mô hình có thể được huấn luyện bằng cách cực tiểu hóa hàm mất mát sau:

$$\boldsymbol{O = \frac{1}{n_l} \sum_{i=1}^{n_l} \mathbf{loss}(\hat{y}_i, y_i)} \quad (2.4)$$

với $\boldsymbol{y_i}$ là nhãn thật của đỉnh $\boldsymbol{i}$, $\boldsymbol{n_l}$ là số lượng đỉnh đã được gán nhãn, loss $(\cdot,\cdot)$ là một hàm mất mát như hàm mất mát cross-entropy. Toàn bộ mạng nơ-ron có thể được tối ưu bằng cách thu nhỏ giá trị $\boldsymbol{O}$ thông qua thuật toán lan truyền ngược (backpropagation).

2.3.1. Graph Convolutional Network - GCN

GCN (mạng tích chập đồ thị) hiện là kiến trúc mạng đồ thị phổ biến nhất nhờ tính đơn giản và hiệu quả trong nhiều tác vụ và ứng dụng khác nhau. Cụ thể, biểu diễn của các đỉnh ở mỗi lớp được cập nhật theo quy tắc truyền dẫn sau:

$$\boldsymbol{H^{k+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{k}W^{k}\right)} \quad (2.5)$$

với:

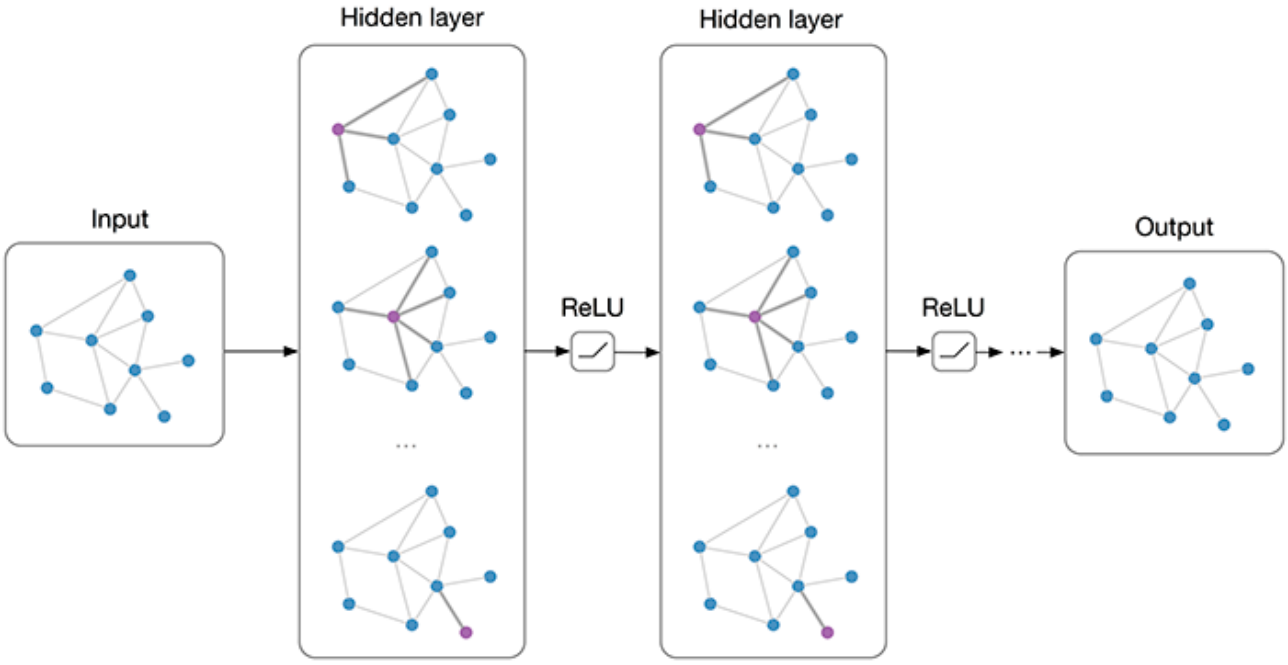
- $\boldsymbol{\tilde{A} = A + I}$: là ma trận kề của đồ thị vô hướng $\boldsymbol{\mathcal{G}}$ đã cộng thêm các tự kết nối self-loops, cho phép kết hợp luôn đặc trưng của chính đỉnh đó khi cập nhật biểu diễn đỉnh.

- $\boldsymbol{I} \in \mathbb{R}^{\boldsymbol{N}\times\boldsymbol{N}}$: là ma trận đơn vị.

- $\boldsymbol{\tilde{D}}$: là ma trận đường chéo với $\boldsymbol{\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}}$.

- **Hàm** $\boldsymbol{\sigma}(\cdot)$: là hàm kích hoạt như ReLU hoặc Tanh. Hàm ReLU thường dùng là $\mathbf{ReLU}(\boldsymbol{x}) = \mathbf{max}(\mathbf{0}, \boldsymbol{x})$.

- **Ma trận** $\boldsymbol{W^k} \in \mathbb{R}^{\boldsymbol{F}\times\boldsymbol{F'}}$: là tham số tuyến tính sẽ được huấn luyện trong quá trình tối ưu.



Hình 2.1: Graph Convolutional Network

Ta có thể bóc tách phương trình (2.5) để thấy hai thành phần Aggregate và Combine trong GCN. Với một đỉnh $\boldsymbol{i}$, phương trình cập nhật biểu diễn đỉnh có thể viết lại như sau:

$$\boldsymbol{H_i^k = \sigma\left(\sum_{j\in\mathcal{N}(i)\cup\{i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k\right)} \quad (2.6)$$

$$\boldsymbol{H_i^k = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k + \frac{1}{\tilde{D}_{ii}} H_i^{k-1} W^k\right)} \quad (2.7)$$

Trong (2.7) ta thấy hàm Aggregate được định nghĩa như trung bình các trọng số của các đỉnh lân cận. Trọng số của đỉnh lân cận $\boldsymbol{j}$ được xác định bởi trọng số của cạnh nối giữa $\boldsymbol{i}$ và $\boldsymbol{j}$ (tức là $\boldsymbol{A_{ij}}$ đã được chuẩn hóa theo bậc của cả hai đỉnh). Hàm Combine là tổng của các thông điệp đã được tổng hợp và chính biểu diễn của đỉnh, trong đó biểu diễn đỉnh được chuẩn hóa theo bậc của nó.

2.3.2. Graph Attention Network - GAT

Trong GCN, đối với một đỉnh mục tiêu $\boldsymbol{i}$, tầm quan trọng của một đỉnh lân cận $\boldsymbol{j}$ được xác định bởi trọng số của cạnh $\boldsymbol{A_{ij}}$ (được chuẩn hóa theo bậc của các đỉnh). Tuy nhiên, trong thực tế, đồ thị đầu vào có thể bị nhiễu. Trọng số các cạnh có thể không phản ánh đúng độ mạnh thực sự giữa hai đỉnh. Do đó, một phương pháp có cơ sở vững chắc hơn là tự động học tầm quan trọng của từng đỉnh lân cận. GAT được xây dựng dựa trên ý tưởng này và cố gắng học tầm quan trọng của mỗi đỉnh lân cận dựa trên cơ chế Attention. Cơ chế Attention đã được sử dụng rộng rãi trong nhiều tác vụ về ngôn ngữ tự nhiên và thị giác máy tính.

Graph Attention Layer: định nghĩa cách chuyển các biểu diễn ẩn của các đỉnh tại lớp $\boldsymbol{k-1}$ (ký hiệu là $\boldsymbol{H^{k-1}} \in \mathbb{R}^{\boldsymbol{N\times F}}$) sang các biểu diễn đỉnh mới là $\boldsymbol{H^k} \in \mathbb{R}^{\boldsymbol{N\times F'}}$. Để đảm bảo đủ khả năng biểu đạt khi biến đổi các biểu diễn đỉnh cấp thấp lên cấp cao hơn, một phép biến đổi tuyến tính chung được áp dụng cho mọi đỉnh, ký hiệu là $\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{F'\times F}}$. Sau đó, self-attention được định nghĩa trên các

đỉnh, đo lường hệ số attention cho bất kỳ cặp đỉnh nào thông qua một cơ chế attention chung $\boldsymbol{a}$: $\mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$:

$$\boldsymbol{e_{ij}} = \boldsymbol{a}\left(\boldsymbol{WH_i^{k-1}}, \boldsymbol{WH_j^{k-1}}\right) = \mathbf{LeakyReLU}\left(\vec{\boldsymbol{a}}^{\boldsymbol{T}}\left[\boldsymbol{WH_i^{k-1}} \| \boldsymbol{WH_j^{k-1}}\right]\right) \quad (2.8)$$

với $\boldsymbol{e_{ij}}$ biểu thị độ mạnh quan hệ giữa đỉnh $\boldsymbol{i}$ và $\boldsymbol{j}$. Lưu ý trong phần này dùng $\boldsymbol{H_i^{k-1}}$ để biểu diễn một vector theo cột thay vì vector theo hàng. Về lý thuyết, với mỗi đỉnh ta có thể cho nó "chú ý" đến mọi đỉnh khác trên đồ thị, nhưng điều này lại bỏ qua thông tin cấu trúc của đồ thị. Giải pháp hợp lý hơn là chỉ chú ý đến các đỉnh lân cận của mỗi đỉnh. Trong thực tế, chỉ các đỉnh bậc nhất (bao gồm cả chính nó) được sử dụng. Để các hệ số có thể so sánh được giữa các đỉnh khác nhau, các hệ số attention thường được chuẩn hóa bằng hàm softmax:

$$\boldsymbol{\alpha_{ij}} = \mathbf{Softmax_j}\left(\{\boldsymbol{e_{ij}}\}\right) = \frac{\exp\left(\boldsymbol{e_{ij}}\right)}{\sum_{\boldsymbol{l} \in \boldsymbol{\mathcal{N}(i)}} \exp\left(\boldsymbol{e_{il}}\right)} \quad (2.9)$$

Ta thấy rằng với một đỉnh $\boldsymbol{i}, \boldsymbol{\alpha_{ij}}$ về cơ bản định nghĩa một phân phối đa thức trên các đỉnh lân cận, cũng có thể hiểu như xác suất chuyển tiếp từ $\boldsymbol{i}$ tới từng đỉnh lân cận của nó. Cơ chế attention $\boldsymbol{a}$ được định nghĩa như một mạng nơ-ron truyền thẳng một lớp gồm một phép biến đổi tuyến tính với vector trọng số $\boldsymbol{W_2} \in \mathbb{R}^{\boldsymbol{1 \times 2F'}}$ và hàm kích hoạt phi tuyến LeakyReLU (với hệ số dốc âm $\boldsymbol{\alpha} = 0.2$). Cụ thể, ta tính hệ số attention theo kiến trúc:

$$\boldsymbol{\alpha_{ij}} = \frac{\exp\left(\mathbf{LeakyReLU}\left(\boldsymbol{W_2}\left[\boldsymbol{WH_i^{k-1}} \| \boldsymbol{WH_j^{k-1}}\right]\right)\right)}{\sum_{\boldsymbol{l} \in \boldsymbol{\mathcal{N}(i)}} \exp\left(\mathbf{LeakyReLU}\left(\boldsymbol{W_2}\left[\boldsymbol{WH_i^{k-1}} \| \boldsymbol{WH_l^{k-1}}\right]\right)\right)} \quad (2.10)$$

với || biểu thị phép ghép nối hai vector. Biểu diễn đỉnh mới là một tổ hợp tuyến tính của các vector lân cận với trọng số là các hệ số attention (có thể là một biến đổi phi tuyến), tức là:

$$\boldsymbol{H_i^k} = \boldsymbol{\sigma}\left(\sum_{\boldsymbol{j} \in \boldsymbol{\mathcal{N}(i)}} \boldsymbol{\alpha_{ij}} \boldsymbol{WH_j^{k-1}}\right) \quad (2.11)$$

Multi-head Attention: Trong thực tế, thay vì chỉ sử dụng một cơ chế attention duy nhất, ta có thể dùng cơ chế multi-head attention, trong đó mỗi

attention head sẽ xác định một hàm đo độ tương đồng khác nhau giữa các đỉnh. Đối với mỗi attention head, ta độc lập thu được một biểu diễn đỉnh mới theo phương trình (2.11). Biểu diễn cuối cùng của mỗi đỉnh sẽ là phép ghép nối (concatenation) các biểu diễn đỉnh do từng attention head học được. Về mặt toán học, ta có:

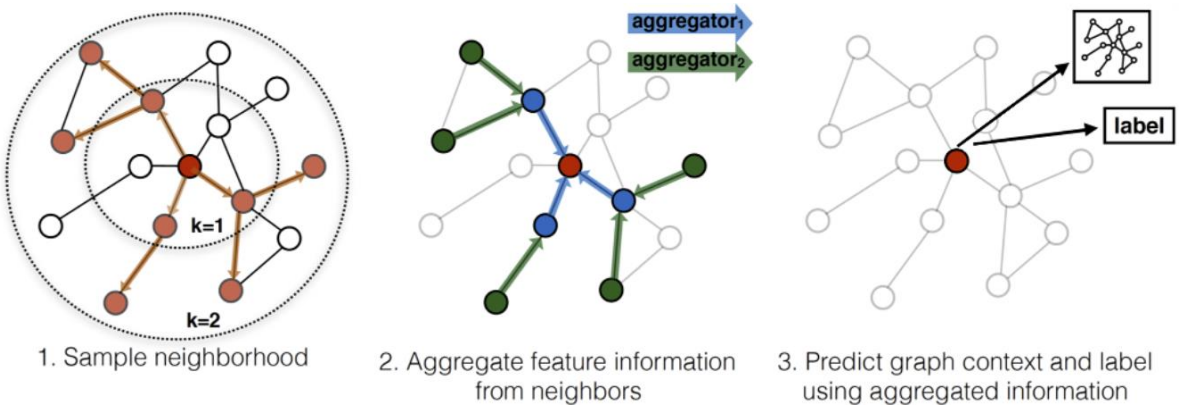
$$\boldsymbol{H_i^k = \|_{t=1}^{T} \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t W^t H_j^{k-1} \right)} \quad (2.12)$$

với $\boldsymbol{T}$ là tổng số attention head, $\boldsymbol{\alpha_{ij}^t}$ là hệ số attention được tính từ head thứ $\boldsymbol{t}$, và $\boldsymbol{W^t}$ là ma trận biến đổi tuyến tính của head thứ $\boldsymbol{t}$. Ở lớp cuối cùng, khi cố gắng kết hợp các biểu diễn đỉnh từ các attention head khác nhau, thay vì dùng concatenation, có thể sử dụng các kỹ thuật pooling khác như chỉ cần lấy trung bình các biểu diễn đỉnh từ các attention head khác nhau. Khi đó:

$$\boldsymbol{H_i^k = \sigma\left(\frac{1}{T} \sum_{t=1}^{T} \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t W^t H_j^{k-1} \right)} \quad (2.13)$$

2.3.3. Graph Sample and Aggregate - GraphSAGE

Ở giai đoạn đầu của sự phát triển GNN, phần lớn các nghiên cứu tập trung vào học chuyển tiếp (Transductive Learning) trên đồ thị có kích thước cố định, trong khi phương pháp học quy nạp (Inductive Learning) lại thực tiễn hơn trong nhiều trường hợp. GraphSAGE là một mô hình học biểu diễn cho các đỉnh trong đồ thị theo kiểu quy nạp, tức là có thể tổng quát hóa để biểu diễn các đỉnh mới chưa từng xuất hiện trong quá trình huấn luyện. Kiến trúc tổng thể của GraphSAGE được minh họa trong hình 2.2.



Hình 2.2: Kiến trúc tổng thể của GraphSAGE

GraphSAGE có thể được xem là một phần mở rộng của GNN. Sự mở rộng đầu tiên là hàm Aggregate được khái quát hóa. Cho đồ thị $\boldsymbol{\mathcal{G}(\mathcal{V},\mathcal{E})}$, $\boldsymbol{\mathcal{N}(v)}$ là tập lân cận của đỉnh $\boldsymbol{v}$, $\mathbf{h}$ là biểu diễn của đỉnh, thì quá trình tạo embedding tại lớp $\boldsymbol{(l+1)}$ của đỉnh $\boldsymbol{v \in \mathcal{V}}$ có thể được biểu diễn như sau:

$$\mathbf{h}_{\mathcal{N}(v)}^{(l+1)} = \mathbf{Aggregate}_l\left(\left\{\mathbf{h}_u^{(l)}, \forall u \in \mathcal{N}(v)\right\}\right) \quad (2.14)$$

khác với hàm tổng hợp trung bình gốc trong GCN, GraphSAGE đề xuất các bộ tổng hợp LSTM và Pooling để tổng hợp thông tin từ các đỉnh lân cận.

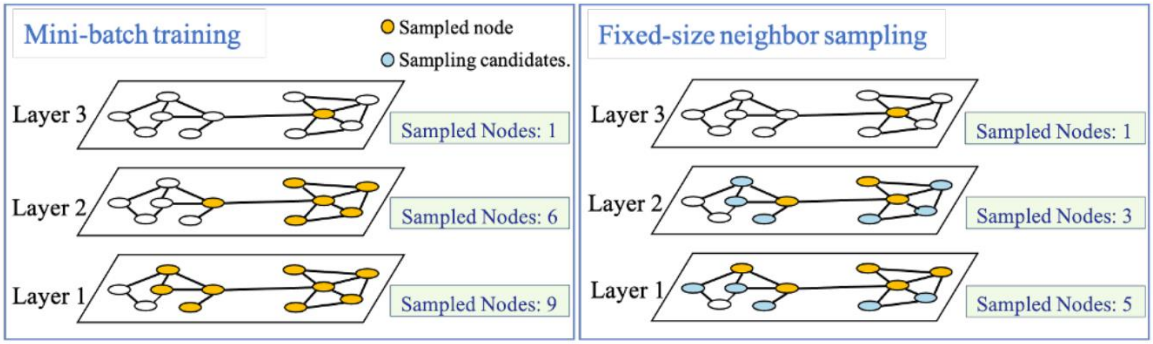
Sự mở rộng thứ hai là thay vì sử dụng hàm Combine GraphSAGE áp dụng concatenation để kết hợp thông tin của đỉnh mục tiêu và các đỉnh lân cận như sau:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(W^{(l+1)} \cdot \mathbf{CONCAT}\left(\mathbf{h}_v^{(l)}, \mathbf{h}_{\mathcal{N}(v)}^{(l+1)}\right)\right) \quad (2.15)$$

trong đó $\boldsymbol{W^{(l+1)}}$ là ma trận trọng số, và $\boldsymbol{\sigma}$ là hàm kích hoạt.

Để làm cho GNN phù hợp với các đồ thị lớn, GraphSAGE giới thiệu chiến lược huấn luyện mini-batch nhằm giảm chi phí tính toán trong giai đoạn huấn luyện. Cụ thể, trong mỗi vòng huấn luyện, chỉ các đỉnh được sử dụng để tính toán biểu diễn trong batch mới được xét đến, điều này giúp giảm đáng kể số lượng đỉnh được lấy mẫu. Lấy lớp 2 trong hình 2.3(a) làm ví dụ. Không giống như huấn luyện toàn phần (full-batch training), vốn xét đến toàn bộ 11 đỉnh, chỉ có 6 đỉnh được sử dụng cho huấn luyện mini-batch. Tuy nhiên, cách triển khai đơn giản của huấn luyện mini-batch gặp phải vấn đề mở rộng lân cận. Như minh họa ở Layer 1 của hình 2.3(a), phần lớn các đỉnh được lấy mẫu bởi vì số lượng đỉnh lân cận tăng

theo cấp số nhân nếu tất cả các đỉnh lân cận đều được lấy mẫu ở mỗi lớp. Do đó, tất cả các đỉnh cuối cùng sẽ được chọn nếu mô hình có nhiều lớp.



(a) Mini-batch training, Batch Size=1 (b) Fix-size neighbor sampling, S=2

Hình 2.3: So sánh mini-batch training và lấy mẫu lân cận kích thước cố định

Để cải thiện hiệu quả đào tạo và khắc phục một phần vấn đề mở rộng lân cận, GraphSAGE áp dụng chiến lược lấy mẫu lân cận có kích thước cố định (fixed-size neighbor sampling). Cụ thể, một tập hợp các đỉnh lân cận có kích thước cố định được lấy mẫu cho mỗi lớp để tính toán, thay vì sử dụng toàn bộ các tập hợp đỉnh lân cận. Ví dụ, người ta có thể đặt tập hợp có kích thước cố định là hai đỉnh lân cận, được minh họa trong hình 2.3(b), các đỉnh màu vàng biểu diễn các đỉnh được lấy mẫu và các đỉnh màu xanh là các đỉnh ứng viên. Ta thấy rằng số lượng các đỉnh được lấy mẫu giảm đáng kể, đặc biệt đối với Layer 1.

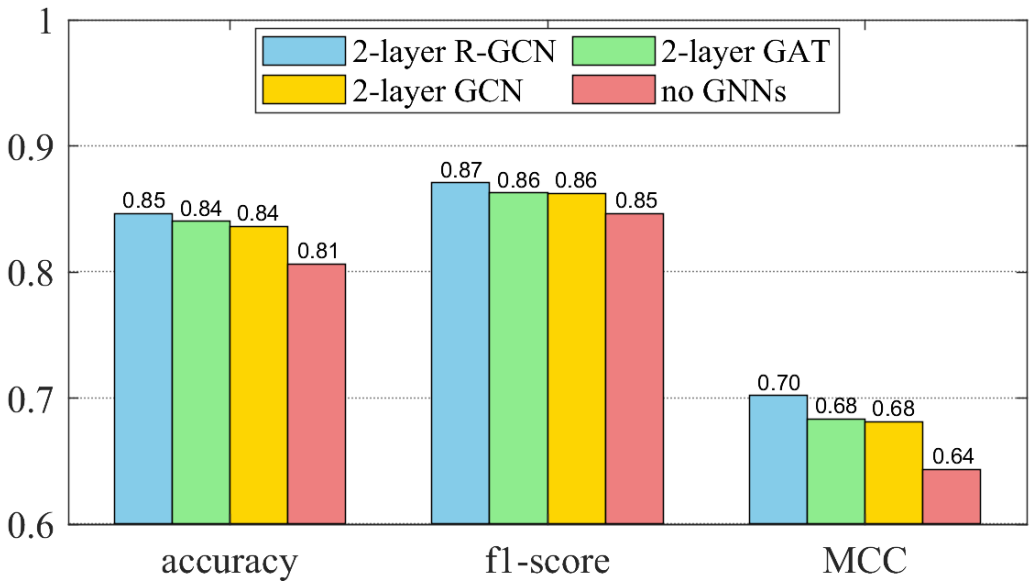
Tóm lại, GraphSAGE là phương pháp học biểu diễn theo hướng quy nạp trên các đồ thị lớn. Phương pháp này giới thiệu một hàm tổng hợp được khái quát hóa, huấn luyện theo mini-batch và thuật toán lấy mẫu lân cận với kích thước cố định nhằm tăng tốc quá trình huấn luyện. Tuy nhiên, chiến lược lấy mẫu lân cận cố định không thể hoàn toàn tránh được vấn đề bùng nổ số lượng lân cận. Ngoài ra, không có đảm bảo lý thuyết nào cho chất lượng của việc lấy mẫu.

2.4. Hiệu năng các kiến trúc Graph Neural Network trong các nghiên cứu thực nghiệm về phân loại tài khoản mạng xã hội

Các nghiên cứu gần đây đã áp dụng GNN trên dữ liệu mạng xã hội thực tế (Twitter, Facebook…) để phân loại tài khoản thật - giả. Trong các phương pháp này, GCN, GAT và GraphSAGE là những mô hình GNN tiêu biểu được so sánh.

Những mô hình này xem mỗi tài khoản người dùng là một đỉnh trong đồ thị và sử dụng các cạnh (quan hệ kết bạn, số lượt tương tác...) để học biểu diễn cho đỉnh, kết hợp với các đặc trưng của tài khoản (hồ sơ, nội dung bài viết…) nhằm dự đoán tài khoản đó là thật hay giả.

Nhiều benchmark đã được xây dựng từ dữ liệu thực tế để đánh giá hiệu năng mô hình. Chẳng hạn, TwiBot-20 (Twitter, 2020) có hơn 229.000 người dùng với khoảng (~5.800) bot. Kết quả tổng quan cho thấy các phương pháp dựa trên đồ thị (GNN) nhìn chung vượt trội hơn so với phương pháp truyền thống dùng đặc trưng rời rạc hoặc nội dung văn bản. Thực nghiệm trên TwiBot-20 cho thấy top 5 mô hình tốt nhất đều là GNN, với F1 trung bình cao hơn (~13.8%) so với mức trung bình của tất cả các phương pháp. Điều này khẳng định việc khai thác cấu trúc mạng xã hội bằng GNN giúp cải thiện đáng kể khả năng nhận diện tài khoản giả. Trong bài báo khoa học trên *arXiv* với mã số *2106.13092*, có tiêu đề: *"BotRGCN: Twitter bot detection with Relational Graph Convolutional Networks"*[6] được công bố bởi nhóm tác giả *"Shangbin Feng, Herun Wan, Ningnan Wang, Minnan Luo"* đến từ *"Xi'an Jiaotong University"* với kết quả thực nghiệm trên TwiBot-20 như sau:



Hình 2.4: Kết quả thực nghiệm bài báo mã số 2106.13092 trên arXiv

Kết quả cho thấy R-GCN đạt F1 cao nhất (~87%), nhỉnh hơn một chút so với GCN và GAT (~86%). Độ chính xác của các mô hình trong thử nghiệm này (~85%) và cao hơn đáng kể so với không sử dụng GNN (~81%). Tuy nhiên, hiệu năng tương đối của các kiến trúc trên còn phụ thuộc vào đặc điểm dữ liệu.

2.4.1. Graph Convolutional Network - GCN

GCN áp dụng tích chập trên đồ thị bằng cách trung bình các hàng xóm của một đỉnh (theo ma trận Laplacian). Ưu điểm nổi bật của GCN là kiến trúc đơn giản, số lượng tham số ít nên dễ huấn luyện và ít nguy cơ quá khớp (overfitting) hơn trên dữ liệu ít. Thực tế cho thấy GCN có đủ cấu trúc phổ (spectrum) của đồ thị, phù hợp với các mạng đồng nhất cao (nơi kết nối giữa các đỉnh cùng nhãn rất phổ biến).

Tuy nhiên, nhược điểm của GCN là tính chuyển tiếp, mô hình cần được huấn luyện lại khi thêm đỉnh mới, khó áp dụng trực tiếp cho tài khoản mới xuất hiện. GCN cũng đánh trọng số các láng giềng một cách đồng đều, không phân biệt được tầm quan trọng của từng hàng xóm. Điều này có thể làm giảm hiệu quả trên đồ thị mà thông tin từ các láng giềng không đồng nhất. Hơn nữa, phép tích chập dựa trên Laplacian khiến GCN bị ràng buộc bởi cấu trúc toàn cục (mọi láng giềng đóng góp ngang nhau theo bậc của đỉnh). Trong trường hợp các tài khoản giả tinh vi ẩn mình giữa nhiều hàng xóm thật, GCN có thể bỏ lỡ tín hiệu đặc trưng do không biết tập trung vào một số kết nối quan trọng.

2.4.2. Graph Attention Network - GAT

GAT bổ sung cơ chế self-attention vào GNN, cho phép mô hình học trọng số khác nhau cho mỗi hàng xóm của một đỉnh. Trong lớp GAT, mỗi đỉnh lắng nghe thông tin từ các láng giềng với mức độ quan trọng khác nhau, được điều chỉnh qua hệ số attention học được. Ưu điểm rõ rệt của GAT là khả năng nhấn mạnh các kết nối quan trọng và giảm nhiễu từ các kết nối kém liên quan. Một ưu thế khác của GAT là linh hoạt với dữ liệu không đồng nhất với đồ thị nhiều kết nối khác nhãn hoặc có các loại quan hệ khác nhau, GAT vẫn có thể học trọng số phù hợp để phân biệt ảnh hưởng của từng loại hàng xóm. Nhờ khắc phục ràng

buộc Laplacian, GAT tích hợp tốt hơn tương quan giữa các đặc trưng đỉnh và đạt hiệu suất cao, đặc biệt trong việc phát hiện bot có hành vi khác biệt so với các hàng xóm.

Nhược điểm chính của GAT là chi phí tính toán cao hơn. Cơ chế attention đòi hỏi tính trọng số cho từng cạnh trong mỗi bước lan truyền, khiến độ phức tạp tăng lên đáng kể khi một đỉnh có nhiều hàng xóm. Trên đồ thị lớn (như mạng xã hội hàng triệu đỉnh…) GAT có thể chậm hơn đáng kể so với GCN và GraphSAGE. Ngoài ra, GAT có nhiều tham số hơn (ma trận trọng số và vector attention), do đó mô hình phức tạp hơn và cần nhiều dữ liệu huấn luyện hơn để tránh overfitting. Trong một số thử nghiệm với dữ liệu ít, GAT ban đầu cho kết quả kém GCN, nhưng khi bổ sung thêm đặc trưng hoặc huấn luyện lâu hơn thì lại đạt hiệu quả cao hơn. Tóm lại, GAT thường đạt hiệu năng phân loại cao nhất nhờ trọng số linh hoạt nhưng đánh đổi bằng tốc độ và độ phức tạp triển khai.

2.4.3. Graph Sample and Aggregate - GraphSAGE

GraphSAGE là mô hình GNN quy nạp, học một hàm Aggregate để trích xuất đặc trưng từ mẫu lân cận của một đỉnh. Lợi thế lớn nhất của GraphSAGE là khả năng tổng quát hóa cho các đỉnh chưa từng thấy, rất hữu ích khi phát hiện tài khoản giả mới xuất hiện hoặc áp dụng mô hình qua các thời điểm khác nhau. GraphSAGE cũng cho phép huấn luyện theo mini-batch trên đồ thị lớn thay vì dùng toàn bộ đồ thị, GraphSAGE chỉ lấy một lượng hàng xóm ngẫu nhiên cho mỗi đỉnh tại mỗi bước, giúp giảm yêu cầu bộ nhớ và tăng tốc độ trên mạng lớn. Tính năng lấy mẫu lân cận này là then chốt để triển khai GNN trong môi trường thực tế, nơi cần cập nhật mô hình nhiều lần trong ngày.

Nhược điểm của GraphSAGE là việc lựa chọn chiến lược lấy mẫu lân cận và hàm tổng hợp có thể ảnh hưởng đến kết quả. Nếu mẫu lân cận quá nhỏ, mô hình có thể bỏ lỡ thông tin; nếu quá lớn, lại mất ưu thế tốc độ. So với GAT, GraphSAGE không học trọng số riêng cho từng cạnh, nên trong trường hợp một đỉnh có nhiều hàng xóm và chỉ một vài trong số đó là quan trọng (ví dụ vài bạn bè thân thiết trong mạng xã hội…) GraphSAGE có thể không nhấn mạnh đúng

mức các kết nối đó. Dù vậy, do không cần tính attention cho từng cạnh, GraphSAGE tiết kiệm tính toán hơn GAT trên đồ thị rất lớn và dễ triển khai song song. Tổng thể, GraphSAGE được đánh giá là giải pháp cân bằng giữa hiệu quả và chi phí, hiệu năng cao gần mức GAT mà huấn luyện nhanh hơn, phù hợp cho hệ thống quy mô lớn, động.

2.4.4. Đánh giá tổng quan

Các nghiên cứu thực nghiệm dựa trên dữ liệu thu thập từ các nền tảng mạng xã hội phổ biến như Twitter, Facebook và nhiều hệ thống khác đã cho thấy GNN là một phương pháp rất hứa hẹn trong việc phân loại các tài khoản thật - giả. Trong nhóm các mô hình GNN, ba kiến trúc phổ biến là GCN, GraphSAGE và GAT đã được áp dụng thành công giúp phát hiện nhiều tài khoản bot ẩn danh mà phương pháp truyền thống bỏ sót. Về mặt hiệu năng, GAT thường đạt kết quả nhỉnh hơn một chút so với hai mô hình còn lại khi xét về các thước đo như độ chính xác và F1-score. Lợi thế này đến từ cơ chế attention tích hợp trong GAT, cho phép mô hình học được mức độ quan trọng khác nhau của các nút láng giềng trong đồ thị thay vì coi chúng như nhau. Điều này giúp GAT có khả năng tập trung vào các mối quan hệ quan trọng và bỏ qua nhiễu. GraphSAGE, mặc dù không vượt qua GAT về độ chính xác, nhưng lại bám sát về hiệu năng và nổi bật ở khả năng tổng quát hóa cho dữ liệu mới. Khả năng này đặc biệt hữu ích trong các hệ thống mà dữ liệu đồ thị liên tục thay đổi. GCN là mô hình ra đời sớm hơn và cấu trúc đơn giản hơn so với hai mô hình trên, nhưng vẫn tỏ ra rất hiệu quả trong nhiều tình huống nhờ sự gọn nhẹ, ít tham số và ít nguy cơ overfitting khi dữ liệu hạn chế. Việc lựa chọn mô hình tối ưu phụ thuộc rất nhiều vào đặc thù của dữ liệu. Nếu đồ thị có quy mô rất lớn và thay đổi liên tục, GraphSAGE là lựa chọn phù hợp nhờ khả năng xử lý dữ liệu ngoài phạm vi huấn luyện ban đầu. Nếu mục tiêu là đạt độ chính xác cao nhất và có thể chấp nhận chi phí tính toán lớn hơn, GAT có khả năng vượt trội nhờ cơ chế attention. Trong trường hợp dữ liệu hạn chế hoặc cần một mô hình gọn nhẹ để triển khai nhanh chóng, GCN vẫn là giải pháp đáng tin cậy. Ngoài ra, các kết quả nghiên cứu mới nhất cũng chỉ ra rằng việc kết hợp nhiều

kỹ thuật chẳng hạn như R-GCN hoặc các mô hình lai giữa GCN và GAT có thể giúp nâng cao hơn nữa hiệu năng của hệ thống. Trong các mô hình phức hợp này, GCN, GAT và GraphSAGE thường đóng vai trò là nền tảng cơ bản để từ đó phát triển các kiến trúc mạnh hơn, đáp ứng được cả yêu cầu về độ chính xác lẫn khả năng mở rộng.

Tuy nhiên, việc ứng dụng GNN cho bài toán phân loại tài khoản mạng xã hội cũng đặt ra không ít thách thức cần giải quyết. Thứ nhất, quy mô của đồ thị mạng xã hội trong thực tế thường rất lớn, có thể lên tới hàng triệu hoặc hàng trăm triệu đỉnh và cạnh. Điều này không chỉ tạo áp lực về bộ nhớ và thời gian xử lý, mà còn đòi hỏi mô hình GNN phải được tối ưu hóa về kỹ thuật lưu trữ, thuật toán sampling và tính toán song song trên GPU. Nếu không có chiến lược xử lý dữ liệu hợp lý, việc huấn luyện mô hình có thể trở nên không khả thi về mặt tài nguyên. Thứ hai, đặc trưng của tài khoản giả mạo trên mạng xã hội rất đa dạng và biến đổi liên tục. Chúng có thể bao gồm các tài khoản bot tự động đăng bài, tài khoản bán tự động, tài khoản do con người điều hành nhằm phục vụ mục đích quảng bá hoặc tấn công tâm lý, thậm chí là mạng lưới các tài khoản giả liên kết và tương tác với nhau để che giấu hành vi bất thường. Sự đa dạng này khiến dữ liệu đầu vào không đồng nhất và đặt ra yêu cầu cho GNN phải thích nghi với nhiều dạng cấu trúc đồ thị, bao gồm đồ thị đồng nhất (homogeneous graph) và đồ thị hỗn hợp (heterogeneous graph), cũng như các loại đặc trưng thuộc tính khác nhau (văn bản, hình ảnh, hành vi tương tác…). Thứ ba, tồn tại một mâu thuẫn cố hữu giữa hiệu suất phân loại và khả năng giải thích của mô hình. Các kiến trúc GNN nhiều tầng sâu, kết hợp nhiều hàm chú ý hoặc cơ chế tổng hợp đặc trưng phức tạp, thường mang lại độ chính xác cao hơn nhưng lại làm giảm khả năng lý giải cách mô hình đưa ra quyết định. Trong khi đó, việc truy xuất nguồn gốc và giải thích kết quả là yếu tố quan trọng trong các ứng dụng phòng chống tin giả, phát hiện hành vi bất thường và đảm bảo tính minh bạch của hệ thống. Thứ tư, dữ liệu mạng xã hội thường nhiễu và thiếu nhãn. Số lượng tài khoản được gán nhãn chính xác thường rất hạn chế, đồng thời, dữ liệu có thể chứa thông tin sai lệch hoặc bị thao

túng bởi chính các tài khoản giả, gây ảnh hưởng tiêu cực tới quá trình huấn luyện mô hình. Thứ năm, yếu tố thời gian và động lực học của mạng xã hội cũng là một thách thức. Cấu trúc mạng và hành vi của người dùng có thể thay đổi nhanh chóng, khiến mô hình huấn luyện ở thời điểm trước đó trở nên kém hiệu quả khi áp dụng vào dữ liệu mới. Điều này đòi hỏi mô hình GNN phải có khả năng cập nhật và học liên tục (Continual Learning) để thích ứng với sự thay đổi của môi trường dữ liệu. Những thách thức này cho thấy, để áp dụng GNN hiệu quả trong bài toán phân loại tài khoản mạng xã hội, cần phải kết hợp giữa tối ưu kiến trúc mô hình, quản lý dữ liệu thông minh và cân nhắc yếu tố giải thích kết quả nhằm vừa đạt hiệu suất cao vừa đảm bảo tính minh bạch và khả năng ứng dụng thực tiễn.

Chương 3:
CÀI ĐẶT, HUẤN LUYỆN MÔ HÌNH, ĐÁNH GIÁ HIỆU QUẢ

3.1. Môi trường và công cụ cài đặt

Quá trình nghiên cứu, triển khai, huấn luyện các mô hình học máy nói chung và các mô hình mạng nơ-ron đồ thị nói riêng đòi hỏi sự chuẩn bị kỹ lưỡng. Trong đó việc thiết lập một môi trường và công cụ cài đặt phù hợp đóng vai trò vô cùng quan trọng. Môi trường không chỉ là nền tảng đảm bảo tính ổn định trong suốt quá trình triển khai, huấn luyện mô hình mà còn là yếu tố then chốt quyết định khả năng mở rộng và tối ưu hiệu suất xử lý của hệ thống. Một mô hình mạng nơ-ron đồ thị được xây dựng trên môi trường cài đặt hiệu quả sẽ giúp giảm thiểu các lỗi phát sinh, tăng tốc quá trình huấn luyện, đồng thời tạo điều kiện thuận lợi cho việc cập nhật và nâng cấp các mô hình trong tương lai. Chính vì vậy, việc lựa chọn và thiết lập đúng đắn các công cụ, thư viện cùng với các thông số kỹ thuật phù hợp là một bước quan trọng không thể bỏ qua.

3.1.1. Thư viện và Framework sử dụng

Ngôn ngữ lập trình chính được sử dụng trong toàn bộ đồ án là **Python** phiên bản 3.12.11. Đây là một trong những ngôn ngữ phổ biến nhất trong lĩnh vực học máy (Machine Learning) và học sâu (Deep Learning) nhờ cú pháp rõ ràng, dễ đọc, giúp rút ngắn thời gian phát triển, giảm thiểu lỗi lập trình và nâng cao hiệu quả triển khai. Python sở hữu cộng đồng người dùng và nhà phát triển rộng lớn, thường xuyên chia sẻ tài liệu, kinh nghiệm và hỗ trợ kỹ thuật, tạo điều kiện thuận lợi cho quá trình nghiên cứu và phát triển. Bên cạnh đó, Python có hệ sinh thái thư viện phong phú như NumPy, Pandas, Matplotlib, Scikit-learn, TensorFlow, PyTorch…, đáp ứng toàn diện các nhu cầu từ xử lý dữ liệu, trực quan hóa, cho tới xây dựng và huấn luyện mô hình trí tuệ nhân tạo. Việc lựa chọn phiên bản 3.12.11 giúp tận dụng các cải tiến về hiệu năng, quản lý bộ nhớ và khả năng tương thích với các thư viện mới, đảm bảo quá trình triển khai và thử nghiệm của đồ án diễn ra ổn định, mượt mà.

Python hoàn toàn phù hợp với định hướng nghiên cứu của đồ án, nhờ khả năng tích hợp linh hoạt với các thuật toán phức tạp, hỗ trợ thử nghiệm và tinh chỉnh ý tưởng nhanh chóng. Python còn cho phép triển khai mô hình trên nhiều nền tảng từ máy tính cá nhân, máy chủ GPU, tới các dịch vụ điện toán đám mây… mà không cần thay đổi đáng kể cấu trúc mã nguồn. Hệ sinh thái thư viện của Python đóng vai trò quan trọng trong việc cung cấp các công cụ cho xử lý dữ liệu, xây dựng mô hình, đánh giá hiệu suất và trực quan hóa kết quả. Trong khuôn khổ đồ án, các thư viện chính được sử dụng để hỗ trợ triển khai mô hình bao gồm:

- PyTorch: Là một trong những Framework Deep Learning mạnh mẽ và được sử dụng rộng rãi nhất hiện nay. PyTorch nổi bật nhờ tính linh hoạt và khả năng định nghĩa mô hình dưới dạng đồ thị tính toán động (dynamic computation graph). Cơ chế này cho phép các nhà phát triển thay đổi cấu trúc mạng nơ-ron trong quá trình chạy, giúp việc xây dựng, gỡ lỗi và thử nghiệm các mô hình phức tạp trở nên nhanh chóng và trực quan. PyTorch sở hữu API rõ ràng, cú pháp gần gũi với Python và tương thích tốt với các thư viện xử lý dữ liệu như NumPy hay Pandas, giúp quá trình tiền xử lý và huấn luyện dữ liệu được liền mạch. Bên cạnh đó, PyTorch hỗ trợ tính toán tăng tốc trên GPU thông qua CUDA, tận dụng tối đa sức mạnh phần cứng để rút ngắn thời gian huấn luyện mô hình, đặc biệt khi làm việc với các tập dữ liệu lớn hoặc mô hình có nhiều tham số. Một điểm mạnh khác của PyTorch là cộng đồng phát triển rộng lớn và năng động, cung cấp nhiều tài nguyên học tập, mã nguồn mở và các module mở rộng cho các lĩnh vực chuyên biệt. Với sự kết hợp giữa tính linh hoạt, hiệu suất cao và khả năng mở rộng, PyTorch là nền tảng lý tưởng để phát triển, thử nghiệm và tối ưu các mô hình học sâu trong khuôn khổ đồ án này.

- PyTorch Geometric (PyG): Đây là một thư viện mở rộng của PyTorch được thiết kế chuyên biệt cho các bài toán xử lý dữ liệu dạng đồ thị (Graph data). PyG cung cấp tập hợp phong phú các lớp mạng nơ-ron GNN như GCNConv, SAGEConv, GATConv, GraphConv… cùng nhiều biến thể và cải tiến, cho phép triển khai nhanh chóng các kiến trúc mô hình hiện đại trong lĩnh vực học sâu trên

đồ thị. Bên cạnh các lớp mạng, PyG tích hợp các hàm tiền xử lý và chuyển đổi dữ liệu mạnh mẽ, hỗ trợ đọc dữ liệu từ nhiều định dạng khác nhau, xây dựng đồ thị từ bảng dữ liệu hoặc từ các nguồn phi cấu trúc, cũng như chuẩn hóa và tối ưu hóa dữ liệu đầu vào. Thư viện còn cung cấp các kỹ thuật sampling hiệu quả như NeighborSampler hay GraphSAINT, giúp xử lý các đồ thị có quy mô lớn mà không vượt quá giới hạn bộ nhớ, đồng thời rút ngắn thời gian huấn luyện. PyG cũng được đánh giá cao nhờ API trực quan, dễ tích hợp với quy trình huấn luyện PyTorch tiêu chuẩn, đồng thời hỗ trợ các công cụ tối ưu hóa mô hình, tính toán song song trên GPU và kiểm thử mô hình trên các bộ dữ liệu chuẩn như Cora, PubMed hay Reddit... Cộng đồng người dùng và nhà phát triển của PyG hoạt động sôi nổi, liên tục cập nhật các tính năng mới và nghiên cứu tiên tiến, giúp nó trở thành một công cụ mạnh mẽ, đáng tin cậy trong các dự án nghiên cứu và ứng dụng thực tế.

- Scikit-learn: Là một thư viện mã nguồn mở phổ biến trong lĩnh vực Machine Learning, Scikit-learn cung cấp bộ công cụ toàn diện để xây dựng, đánh giá và triển khai mô hình. Trong khuôn khổ đồ án này, Scikit-learn được sử dụng chủ yếu để tính toán các độ đo đánh giá như Accuracy, Precision, Recall, F1-score… giúp định lượng hiệu suất của mô hình một cách khách quan và so sánh giữa các phương án thử nghiệm. Ngoài ra, Scikit-learn hỗ trợ xây dựng ma trận nhầm lẫn (Confusion Matrix), cho phép phân tích chi tiết cách mô hình dự đoán trên từng lớp, từ đó nhận diện những điểm mạnh và điểm yếu trong khả năng phân loại. Thư viện còn cung cấp nhiều công cụ hữu ích khác như cross-validation, grid search… để tìm kiếm siêu tham số tối ưu, và các hàm tiền xử lý dữ liệu như chuẩn hóa, mã hóa nhãn hay trích chọn đặc trưng. Ưu điểm của Scikit-learn là API đơn giản, thống nhất, dễ dàng tích hợp với các thư viện khác như NumPy, Pandas hoặc PyTorch, cùng với tài liệu chi tiết và cộng đồng lớn. Điều này giúp việc đánh giá mô hình diễn ra thuận lợi, nhanh chóng, và đảm bảo tính minh bạch, tái lập kết quả trong quá trình nghiên cứu.

- Matplotlib và Seaborn: Đây là hai thư viện trực quan hóa dữ liệu mạnh mẽ, được sử dụng để biểu diễn kết quả huấn luyện và đánh giá mô hình một cách trực quan. Matplotlib đóng vai trò như nền tảng cơ bản, hỗ trợ vẽ các loại biểu đồ đa dạng như biểu đồ đường, cột, scatter plot, heatmap…, giúp minh họa quá trình thay đổi của loss và accuracy theo từng epoch huấn luyện. Seaborn được xây dựng trên Matplotlib nhưng cung cấp giao diện thân thiện hơn, khả năng tùy chỉnh cao và các kiểu biểu đồ có tính thẩm mỹ tốt ngay từ mặc định. Trong đồ án, Seaborn đặc biệt hữu ích khi trực quan hóa Confusion Matrix dưới dạng heatmap với màu sắc trực quan, giúp người đọc dễ dàng nhận biết các điểm mà mô hình dự đoán sai. Sự kết hợp của Matplotlib và Seaborn cho phép trình bày số liệu một cách rõ ràng, nâng cao khả năng phân tích, hỗ trợ việc rút ra kết luận từ kết quả thực nghiệm. Ngoài ra, cả hai thư viện đều tích hợp tốt với Pandas và NumPy, giúp quá trình xử lý và trực quan hóa dữ liệu diễn ra liền mạch.

- NetworkX: Là một thư viện Python mã nguồn mở, chuyên dùng để biểu diễn, phân tích và trực quan hóa các cấu trúc đồ thị ở mức cơ bản. NetworkX cho phép xây dựng đồ thị với nhiều loại cấu trúc khác nhau như đồ thị vô hướng, có hướng, có trọng số hoặc không trọng số… đồng thời hỗ trợ gán thuộc tính cho nút và cạnh nhằm lưu trữ thêm thông tin liên quan đến dữ liệu. Trong đồ án này, NetworkX được sử dụng để trực quan hóa cấu trúc đồ thị ban đầu trước khi đưa vào các mô hình GNN, giúp dễ dàng kiểm tra tính đúng đắn của dữ liệu và phát hiện các đặc điểm nổi bật trong mạng lưới. Thư viện còn cung cấp nhiều thuật toán cơ bản như tìm đường đi ngắn nhất, phát hiện thành phần liên thông, tính bậc nút hay các chỉ số trung tâm (centrality), hỗ trợ phân tích sơ bộ đặc trưng của đồ thị. Ưu điểm của NetworkX là cú pháp đơn giản, dễ học và tích hợp tốt với Matplotlib, cho phép vẽ các sơ đồ đồ thị với khả năng tùy chỉnh cao giúp phân tích định lượng, trực quan hóa cấu trúc dữ liệu, tạo nền tảng cho các bước xử lý và huấn luyện mô hình Deep Learning.

Ngoài ra, một số thư viện phụ trợ như **NumPy** và **Pandas** cũng được sử dụng để xử lý dữ liệu đầu vào và dữ liệu trung gian trong quá trình thực nghiệm.

NumPy là thư viện nền tảng cho tính toán khoa học trong Python, cung cấp các cấu trúc mảng (array) đa chiều hiệu quả và nhiều hàm toán học tối ưu, giúp thao tác với dữ liệu số nhanh chóng và tiết kiệm bộ nhớ. Pandas được xây dựng trên NumPy, cung cấp các cấu trúc dữ liệu mạnh mẽ như DataFrame và Series, hỗ trợ đọc, ghi dữ liệu từ nhiều định dạng phổ biến (CSV, Excel, JSON…), đồng thời cung cấp các phương thức lọc, nhóm, gộp và biến đổi dữ liệu linh hoạt. Trong đồ án, Pandas được dùng để tiền xử lý dữ liệu trước khi xây dựng đồ thị, bao gồm làm sạch dữ liệu, chuyển đổi kiểu dữ liệu... Việc kết hợp NumPy và Pandas giúp đảm bảo toàn bộ quy trình xử lý dữ liệu diễn ra nhanh chóng, chính xác, đồng thời tạo ra các tập dữ liệu sẵn sàng cho bước huấn luyện mô hình GNN trên PyTorch và PyTorch Geometric.

3.1.2. Cấu hình phần cứng và phần mềm

Việc lựa chọn cấu hình phần cứng và phần mềm phù hợp đóng vai trò then chốt trong việc đảm bảo hiệu quả huấn luyện và thử nghiệm các mô hình mạng nơ-ron đồ thị. Hiệu suất xử lý không chỉ phụ thuộc vào sức mạnh phần cứng, mà còn liên quan đến khả năng tối ưu của môi trường phần mềm, từ đó ảnh hưởng trực tiếp đến tốc độ huấn luyện, độ ổn định và khả năng tái lập kết quả. Toàn bộ quá trình triển khai trong đồ án được thực hiện trên máy tính cá nhân với cấu hình phần cứng cụ thể như sau:

- Hệ điều hành Windows 11 Pro 64-bit: cung cấp môi trường làm việc ổn định, tương thích tốt với các thư viện và công cụ học máy hiện đại.

- Bộ xử lý (CPU) Intel(R) Core(TM) i7-4910MQ: vi xử lý 4 nhân 8 luồng, xung nhịp tối đa 3.90 GHz, đảm bảo khả năng xử lý đa nhiệm và tính toán trên CPU khi cần thiết, đặc biệt hữu ích trong các bước tiền xử lý dữ liệu.

- Card đồ họa (GPU) NVIDIA Quadro K2100M: hỗ trợ tính toán song song thông qua CUDA, cho phép tăng tốc quá trình huấn luyện các mô hình học sâu, mặc dù hiệu năng không cao như các GPU dòng mới nhưng vẫn đáp ứng yêu cầu xử lý cho các mô hình GNN ở quy mô vừa.

- Bộ nhớ RAM 32 GB: dung lượng bộ nhớ lớn giúp xử lý các tập dữ liệu kích thước trung bình đến lớn mà không gặp hiện tượng thiếu bộ nhớ, đồng thời hỗ trợ nhiều tiến trình chạy song song.

Về môi trường phần mềm, tất cả các thư viện học máy và học sâu được cài đặt trong môi trường Python 3.12.11, được quản lý thông qua Conda, một hệ thống quản lý môi trường ảo và gói phần mềm mạnh mẽ. Việc sử dụng Conda giúp cô lập môi trường làm việc, đảm bảo các phiên bản thư viện và gói phụ thuộc được đồng bộ, tránh xung đột giữa các dự án, đồng thời giúp quá trình triển khai trở nên dễ dàng tái lập trên các máy khác. Cấu hình phần cứng và môi trường phần mềm trên tuy không thuộc nhóm hiệu năng cao nhưng vẫn đáp ứng tốt yêu cầu của đồ án, đảm bảo quá trình nghiên cứu, thử nghiệm và đánh giá mô hình GNN diễn ra ổn định, liên tục và có thể mở rộng nếu cần.

3.2. Chuẩn bị dữ liệu

3.2.1. Mô tả bộ dữ liệu sử dụng

Trong đồ án này, bộ dữ liệu **Cresci-2015** được sử dụng để nghiên cứu giải quyết các yêu cầu đặt ra. Đây là một trong những bộ dữ liệu có uy tín và được sử dụng rộng rãi trong cộng đồng nghiên cứu về phát hiện tài khoản giả mạo (social bots) trên mạng xã hội, đặc biệt là trên nền tảng Twitter (X). Bộ dữ liệu này được phát triển và công bố bởi nhóm nghiên cứu đến từ Viện Tin học và Viễn thông thuộc Hội đồng Nghiên cứu Quốc gia Ý *(Institute of Informatics and Telematics, CNR - Consiglio Nazionale delle Ricerche)*, dưới sự dẫn dắt của *Fabio Cresci*, một nhà nghiên cứu hàng đầu trong lĩnh vực phát hiện bot trên mạng xã hội. Bộ dữ liệu lần đầu tiên được giới thiệu trong bài báo khoa học có tựa đề *"Fame for sale: Efficient detection of Fake Twitter followers"*[7] và từ khi ra đời, nó đã trở thành một chuẩn tham chiếu cho nhiều phương pháp phát hiện bot khác nhau, từ các kỹ thuật truyền thống đến những phương pháp hiện đại dựa trên học sâu và học đồ thị. Bộ dữ liệu Cresci-2015 bao gồm tổng cộng 5 tập con, mỗi tập đại diện cho một nhóm tài khoản khác nhau, phản ánh hai nhóm chính là tài khoản thật và tài khoản giả:

- TFP (The Fake Project): Gồm các tài khoản người dùng thật, được thu thập từ dự *án "The Fake Project"* vào tháng 12 năm 2012. Dữ liệu bao gồm các tweet công khai, thông tin profile, cùng các mối quan hệ (friends, followers).

- E13 (Elezioni 2013): Gồm các tài khoản người dùng thật, thu thập từ Twitter trong bối cảnh cuộc bầu cử tại Ý năm 2013. Đây là một trong những tập dữ liệu phản ánh hành vi của người dùng thật trong môi trường chính trị nhạy cảm, nơi mà tính xác thực của tài khoản là yếu tố quan trọng.

- INT (InterTwitter): Là tập dữ liệu chứa các tài khoản giả mạo (fake followers). Những tài khoản này được thu thập từ các dịch vụ mua bán followers trên mạng, và thường có đặc điểm hoạt động không tự nhiên như số lượng lớn các follow trong thời gian ngắn, thiếu nội dung thực tế, và hoạt động giống nhau.

- FSF (FastFollowerz): Cũng là tập tài khoản giả, thu thập từ dịch vụ FastFollowerz - một nền tảng chuyên cung cấp các tài khoản bot với mục tiêu tăng số lượng người theo dõi cho người dùng thật.

- TWT (TwitterTechnology): Tập dữ liệu này cũng chứa các tài khoản giả, được thu thập từ một nhóm tài khoản công nghệ có hoạt động đáng ngờ.

Sub-Datasets	Accounts	Tweets	Friend Relationships	Follow Relationships
TFP	469	563,693	241,710	258,494
E13	1481	2,068,037	667,225	1,526,944
FSF	1169	22,910	253,026	11,893
INT	1337	58,925	517,485	23,173
TWT	845	114,192	729,839	28,588
Total number of accounts	5301			
Number of bots	3351 (63%)			

Hình 3.1: Sơ lược về bộ dữ liệu Cresci-2015

Trong số năm tập con nêu trên, hai tập con là E13 và INT được lựa chọn để phục vụ mục tiêu huấn luyện và đánh giá các mô hình học sâu trên đồ thị trong bài toán phân loại tài khoản mạng xã hội của đồ án này. E13 đại diện cho nhóm người dùng thật, trong khi INT đại diện cho nhóm tài khoản giả (bot, fake followers…). Cả hai tập con đều có quy mô vừa phải và chứa đầy đủ thông tin cần thiết như profile, hoạt động và đặc biệt là mối quan hệ (friends, followers) giúp xây dựng đồ thị xã hội một cách rõ ràng. Khi kết hợp hai tập con E13 và INT,

ta thu được một tập dữ liệu gồm 2.818 tài khoản, trong đó phân bố giữa tài khoản thật và giả là đủ để đào tạo các mô hình học sâu mà không bị mất cân bằng nghiêm trọng giữa 2 nhóm tài khoản.

3.2.2. Tiền xử lý dữ liệu

Quy trình tiền xử lý đóng vai trò quan trọng trong việc chuẩn bị dữ liệu cho huấn luyện mô hình, bao gồm 3 giai đoạn chính:

- Xử lý, làm sạch dữ liệu người dùng
- Xử lý dữ liệu kết nối bạn bè (hoặc followers)
- Chuyển đổi định dạng dữ liệu đồ thị

a. Xử lý, làm sạch dữ liệu người dùng

Xử lý, làm sạch dữ liệu người dùng bước quan trọng đầu tiên nhằm đảm bảo chất lượng và tính sẵn sàng của dữ liệu cho các bước phân tích và mô hình hóa sau này. Dữ liệu người dùng được thu thập từ hai tập tin *users.csv* khác nhau thuộc hai thư mục riêng biệt (E13 và INT). Để đơn giản hóa và chuẩn hóa quy trình xử lý, hai tập dữ liệu này được hợp nhất tạo ra một tập dữ liệu duy nhất với định dạng đồng nhất. Trong quá trình này, chỉ những cột dữ liệu được đánh giá là hữu ích mới được giữ lại, bao gồm:

- id: mã định danh người dùng

- statuses_count: số lượng trạng thái đã đăng

- followers_count: số lượng người theo dõi

- friends_count: số người dùng theo dõi

- favourites_count: số lượt thích

- listed_count: số danh sách mà người dùng xuất hiện

- lang: ngôn ngữ

- time_zone: múi giờ

- location: vị trí địa lý

- dataset: nhãn phân biệt nguồn dữ liệu.

Đây là những thuộc tính đóng vai trò quan trọng trong việc phân tích hành vi và đặc điểm của người dùng trên Twitter.

Sau khi trích xuất các cột cần thiết, bước tiếp theo là xử lý dữ liệu thiếu và mã hóa các giá trị thuộc tính. Đối với các cột dạng số, giá trị bị thiếu được điền bằng 0 nhằm tránh gây lỗi trong các bước tính toán về sau. Với các cột kiểu chuỗi (dạng object) như ngôn ngữ hoặc vị trí, các giá trị thiếu được thay thế tạm thời bằng chuỗi "NaN" sau đó toàn bộ cột được mã hóa bằng LabelEncoder, một kỹ thuật phổ biến trong tiền xử lý dữ liệu để chuyển đổi giá trị phân loại thành số nguyên. Bước này đặc biệt quan trọng khi làm việc với các thuật toán học máy, vốn yêu cầu đầu vào là dữ liệu dạng số.

```
In [2]: # Chọn các cột hữu ích từ dữ liệu người dùng
useful_cols = ['id', 'statuses_count', 'followers_count', 'friends_count', 'favourites_count', 'listed_count', 'lang', 'time_zone', 'location', 'dataset']

# Đọc dữ liệu người dùng từ các tập tin CSV, kết hợp chúng và chỉ giữ lại các cột hữu ích
users = pd.concat([
    pd.read_csv("./cresci-2015/E13/users.csv"),
    pd.read_csv("./cresci-2015/INT/users.csv")],
    ignore_index=True)[useful_cols]

# Lấp đầy các giá trị thiếu và mã hóa dữ liệu người dùng
for col in users.columns:
    if users[col].dtype == 'object':
        users[col] = LabelEncoder().fit_transform(users[col].fillna('NaN'))
    else:
        users[col] = users[col].fillna(0)
display(users)
```

	id	statuses_count	followers_count	friends_count	favourites_count	listed_count	lang	time_zone	location	dataset
0	3610511	20370	5470	2385	145	52	5	29	1006	0
1	5656162	3131	506	381	9	40	1	29	1022	0
2	5682702	4024	264	87	323	16	1	29	551	0
3	6067292	40586	640	622	1118	32	1	29	793	0
4	6015122	2016	62	64	13	0	5	29	1459	0
...	...	...	...	...	...	...	...	...	...	...
2813	1391497074	1	0	17	0	0	1	23	776	1
2814	1391544607	0	1	17	0	0	1	23	793	1
2815	1391622127	2	0	15	0	0	1	23	1024	1
2816	1391832212	2	0	16	0	0	1	23	793	1
2817	1391998039	0	0	17	0	0	1	23	793	1

2818 rows × 10 columns

Hình 3.2: Xử lý, làm sạch dữ liệu người dùng

Kết quả sau quá trình xử lý, làm sạch dữ liệu người dùng là một DataFrame trong đó các dòng đại diện cho từng người dùng và các cột chứa thông tin đã được chuẩn hóa sẵn sàng cho các phân tích thống kê hoặc huấn luyện mô hình dự đoán.

b. Xử lý dữ liệu kết nối bạn bè (hoặc followers)

Mục tiêu chính của bước này là tái hiện cấu trúc mạng xã hội dưới dạng một đồ thị, trong đó mỗi người dùng được biểu diễn như một đỉnh và mỗi mối quan hệ bạn bè (hoặc followers) được biểu diễn như một cạnh nối giữa hai đỉnh đó.

Để thực hiện việc này, toàn bộ dữ liệu về quan hệ bạn bè được thu thập từ hai tập tin *friends.csv* khác nhau thuộc hai thư mục riêng biệt (E13 và INT). Mỗi

tệp chứa danh sách các mối quan hệ giữa người dùng, trong đó mỗi bản ghi mô tả một liên kết từ một người dùng đến một người khác. Hai tệp dữ liệu này sau đó được hợp nhất thành một tập duy nhất bao gồm tất cả các mối quan hệ hiện có.

Tiếp theo, để có thể xây dựng đồ thị mạng xã hội từ dữ liệu vừa thu thập, cần thực hiện một bước ánh xạ quan trọng. Mỗi người dùng trong dữ liệu ban đầu có một mã định danh duy nhất, tuy nhiên các mã định danh này thường là những chuỗi ký tự số hoặc số nguyên lớn và không liên tiếp, khiến cho việc xử lý và biểu diễn đồ thị trở nên kém hiệu quả. Do đó, một danh sách đánh số thứ tự cho toàn bộ người dùng được tạo ra, trong đó mỗi người dùng được gán một chỉ số duy nhất và liên tiếp bắt đầu từ 0. Điều này giúp chuẩn hóa cách biểu diễn các đỉnh trong đồ thị, đồng thời tối ưu hóa về mặt bộ nhớ và tốc độ tính toán cho các bước xử lý tiếp theo. Sau khi hoàn tất bước đánh chỉ số, dữ liệu về các mối quan hệ được xử lý bằng cách thay thế các mã định danh người dùng ban đầu bằng chỉ số tương ứng đã gán. Với mỗi bản ghi trong danh sách bạn bè, thông tin về người tạo liên kết (source_id) và người bị liên kết (target_id) đều được chuyển đổi từ dạng mã định danh sang dạng chỉ số chuẩn hóa. Nhờ đó, mỗi cặp mối quan hệ hiện giờ đã được mô tả bằng hai chỉ số đại diện cho một cạnh trong đồ thị.

Một yếu tố được đặc biệt lưu ý trong quá trình xử lý là tính hai chiều của các mối quan hệ bạn bè. Với mỗi mối quan hệ, một bản ghi phản ánh hướng ngược được bổ sung, tạo nên hai cạnh đối xứng giữa hai đỉnh (đồ thị vô hướng). Việc này đảm bảo rằng nếu có một người kết bạn với người khác, thì trong cấu trúc đồ thị, cả hai được coi là có kết nối qua lại *(lưu ý, thao tác trên không áp dụng với mối quan hệ followers).*

Sau khi tạo ra danh sách các cạnh bao gồm cả hai chiều, một số bước kiểm tra và làm sạch dữ liệu được thực hiện nhằm loại bỏ các bản ghi không hợp lệ. Trước hết, các mối quan hệ mà trong đó một trong hai người dùng không tồn tại trong danh sách người dùng đã được chuẩn hóa sẽ bị loại bỏ. Điều này đảm bảo rằng tất cả các cạnh trong đồ thị đều nối giữa những đỉnh có thông tin đầy đủ. Tiếp theo, các mối quan hệ trùng lặp, tức là các cạnh giống nhau xuất hiện nhiều

lần cũng được loại bỏ để tránh ảnh hưởng đến phân tích thống kê. Những kết nối dạng này không có ý nghĩa thực tiễn trong phần lớn các bài toán phân tích mạng xã hội và thường gây nhiễu cho các thuật toán về đồ thị.

```
In [3]: # Tải dữ liệu bạn bè từ các tập tin CSV và kết hợp chúng
friends = pd.concat([
    pd.read_csv("./cresci-2015/E13/friends.csv"),
    pd.read_csv("./cresci-2015/INT/friends.csv")],
    ignore_index=True)
display(friends)

# Tạo chỉ mục cho người dùng để ánh xạ ID người dùng sang chỉ mục
users_index = {id: index for index, id in enumerate(users.id)}

# Tạo DataFrame cho các cạnh (edge) từ dữ liệu bạn bè
edges = pd.concat([
    pd.DataFrame({
        'source_index': friends['source_id'].map(users_index),
        'target_index': friends['target_id'].map(users_index)
    }),
    pd.DataFrame({
        'source_index': friends['target_id'].map(users_index),
        'target_index': friends['source_id'].map(users_index)
    })
]).dropna().drop_duplicates().query("source_index != target_index").reset_index(drop=True).astype({'source_index': 'long', 'target_index': 'long'})
display(edges)
```

Hình 3.3: Xử lý dữ liệu kết nối bạn bè

Kết quả của toàn bộ quá trình xử lý là một bảng dữ liệu mới (dạng DataFrame) chứa danh sách các cặp chỉ số, mỗi cặp đại diện cho một cạnh trong đồ thị mạng xã hội. Đây chính là phần biểu diễn cấu trúc kết nối giữa người dùng, có thể trực tiếp sử dụng để xây dựng mô hình đồ thị.

3.2.3. Chuyển đổi định dạng dữ liệu đồ thị

Sau khi đã hoàn thiện việc xây dựng tập dữ liệu biểu diễn cấu trúc mạng xã hội dưới dạng đồ thị với các đỉnh là người dùng và các cạnh là mối quan hệ bạn bè giữa họ, bước tiếp theo trong quy trình tiền xử lý dữ liệu là chuyển đổi toàn bộ dữ liệu này sang một định dạng phù hợp để sử dụng trong các mô hình học sâu trên đồ thị. Cụ thể, hệ thống được thiết kế để sử dụng thư viện **PyTorch Geometric**, một công cụ mạnh mẽ hỗ trợ triển khai các mô hình mạng nơ-ron đồ thị. Việc chuyển đổi dữ liệu sang định dạng tensor là yêu cầu bắt buộc, vì đây là cấu trúc dữ liệu cơ bản trong cả PyTorch và PyTorch Geometric.

```
In [4]: # Chuyển đổi dữ liệu người dùng thành tensor
x = users.drop(['id','dataset'], axis=1).values
x = torch.tensor(x, dtype=torch.float)

# Chuyển đổi dữ liệu cạnh thành tensor và thêm thêm các cạnh tự nối (self-loops)
edge_index = torch.tensor(edges[['source_index', 'target_index']].values, dtype=torch.long).t().contiguous()
edge_index, _ = add_self_loops(edge_index, num_nodes=x.size(0))

# Chuyển đổi nhãn người dùng thành tensor
y = torch.tensor(users.dataset, dtype=torch.long)

# Tạo đối tượng graph từ dữ liệu người dùng, các cạnh và nhãn
graph = Data(x=x, edge_index=edge_index, y=y)
display(graph)
```

```
Data(x=[2818, 8], edge_index=[2, 7378], y=[2818])
```

Hình 3.4: Chuyển đổi định dạng dữ liệu đồ thị

Trước hết, toàn bộ thông tin thuộc tính của người dùng (các đặc trưng đầu vào) được trích xuất bằng cách loại bỏ các trường không cần thiết như mã định danh và nhãn phân loại. Các trường như số lượng bài đăng, số lượng người theo dõi, bạn bè, số lượt thích, số lần được thêm vào danh sách, ngôn ngữ, múi giờ, và vị trí đã được xử lý, mã hóa và chuẩn hóa ở các bước trước được giữ lại để xây dựng thành một ma trận đặc trưng. Mỗi dòng trong ma trận này đại diện cho một người dùng, còn mỗi cột là một đặc trưng định lượng tương ứng. Tập dữ liệu này được chuyển đổi thành một tensor để làm đầu vào cho mạng học sâu.

Sau đó, cấu trúc liên kết của đồ thị, tức là danh sách các cặp chỉ số biểu diễn các mối quan hệ bạn bè giữa người dùng cũng được chuyển đổi sang dạng tensor. Dữ liệu này bao gồm hai dòng, trong đó dòng đầu tiên chứa chỉ số của các người tạo mối quan hệ (source_index) và dòng thứ hai chứa các chỉ số của người bị liên kết đến (target_index). Điều này tạo ra một ma trận cạnh (edge_index) có kích thước hai dòng và số cột bằng tổng số mối quan hệ. Đây là định dạng chuẩn để PyTorch Geometric có thể hiểu và xây dựng biểu diễn đồ thị.

Một thao tác quan trọng được thực hiện ở bước này là thêm vào các liên kết self-loops. Đây là các mối liên kết mà mỗi đỉnh được nối với chính nó. Việc thêm self-loops không phản ánh một đặc điểm xã hội thực tế mà mang ý nghĩa kỹ thuật trong huấn luyện mạng nơ-ron đồ thị. Khi các self-loops được bổ sung, mỗi đỉnh trong quá trình lan truyền thông tin sẽ có thể giữ lại một phần đặc trưng của chính nó. Điều này giúp mạng học sâu không làm mất đi thông tin gốc của đỉnh khi cập nhật qua các lớp, đồng thời giúp cải thiện độ ổn định và hiệu quả của mô hình. Đây là một kỹ thuật phổ biến được áp dụng trong hầu hết các kiến trúc mạng nơ-ron đồ thị.

Bên cạnh đó, dữ liệu về nhãn của người dùng biểu thị theo phân loại ban đầu từ hai tập dữ liệu khác nhau cũng được chuyển đổi sang tensor. Mỗi giá trị trong tensor này tương ứng với một người dùng và thể hiện thuộc tính mục tiêu (label) mà mô hình cần học. Trong trường hợp này, nhãn là nhị phân, với hai giá trị 0 và 1 tương ứng với tập dữ liệu gốc E13 và INT. Tập nhãn này đóng vai trò là

thông tin huấn luyện để mô hình có thể học được cách phân biệt giữa hai nhóm người dùng dựa trên đặc trưng và cấu trúc liên kết.

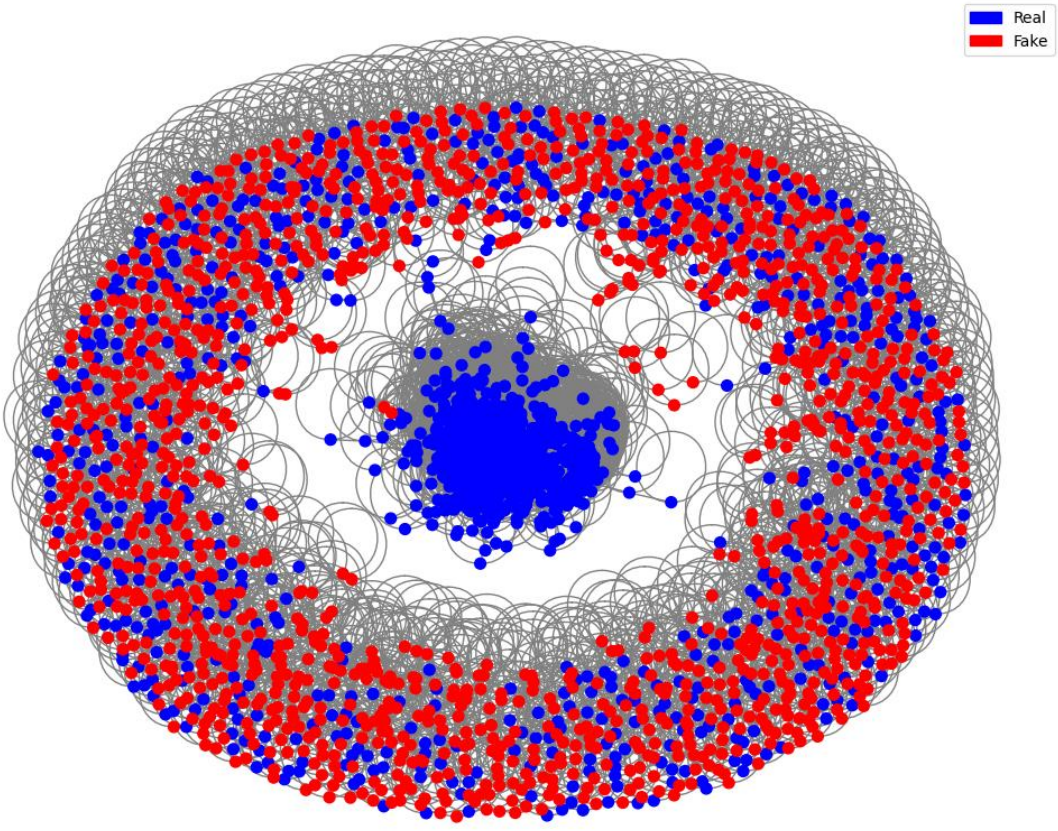
Khi tất cả các thành phần đã được chuyển đổi sang tensor bao gồm đặc trưng của đỉnh, cấu trúc đồ thị và nhãn mục tiêu một đối tượng đồ thị hoàn chỉnh được tạo ra bằng cách kết hợp chúng. Đối tượng này chứa ba thành phần chính:

- Ma trận đặc trưng của các đỉnh (x), có kích thước tương ứng với số người dùng và số đặc trưng.

- Ma trận chỉ số cạnh (edge_index), mô tả toàn bộ các mối quan hệ trong mạng nơ-ron đồ thị.

- Tensor nhãn mục tiêu (y), đại diện cho đầu ra cần dự đoán.

Đây chính là đầu vào chuẩn để đưa vào các mô hình mạng nơ-ron đồ thị Quá trình này đánh dấu giai đoạn chuyển đổi từ dữ liệu dạng bảng sang dạng đồ thị mở đường cho việc áp dụng các thuật toán học sâu khai thác cấu trúc mạng nơ-ron đồ thị như GCN, GAT hoặc GraphSAGE.



Hình 3.5: Đồ thị trực quan

3.3. Cấu hình huấn luyện

3.3.1. Lựa chọn mô hình Graph Neural Network

a. Graph Convolutional Network - GCN

In [6]:
```python
# Định nghĩa mô hình GCN
class GCN(torch.nn.Module):
    # Phương thức khởi tạo kiến trúc mô hình GCN
    def __init__(self, in_channels, hidden_channels, num_classes):
        super().__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, num_classes)

    # Phương thức forward để thực hiện quá trình lan truyền qua các lớp GCNConv
    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return x

# Khởi tạo mô hình GCN cụ thể
model = GCN(graph.num_node_features, 64, len(np.unique(users.dataset)))
display(model)
```

Hình 3.6: Mô hình GCN được lựa chọn

GCN dựa trên một nguyên lý đơn giản nhưng hiệu quả, biểu diễn của một đỉnh không chỉ phụ thuộc vào chính bản thân nó, mà còn phụ thuộc vào các đỉnh lân cận trong đồ thị. Ý tưởng này phản ánh nguyên lý lan truyền thông tin tức là mỗi đỉnh thu thập và tổng hợp thông tin từ các hàng xóm của mình để xây dựng một biểu diễn giàu ngữ nghĩa hơn.

Ở đây, mô hình GCN được lựa chọn xây dựng theo kiến trúc hai lớp GCNConv. Ý tưởng của việc sử dụng hai lớp là để khai thác mối quan hệ không chỉ trong phạm vi hàng xóm bậc một, mà còn lan rộng đến hàng xóm bậc hai. Ở lớp đầu tiên, đặc trưng đầu vào $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{d}}$ được biến đổi thành biểu diễn trung gian $\boldsymbol{H}^{(\boldsymbol{1})} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{h}}$, với $\boldsymbol{h}$ là số chiều ẩn do người thiết kế chỉ định. Lớp thứ hai tiếp tục chuyển đổi từ $\boldsymbol{H}^{(\boldsymbol{1})}$ thành $\boldsymbol{Z} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{c}}$, với $\boldsymbol{c}$ là số lớp phân loại (số nhãn đầu ra).

Việc lựa chọn số chiều đầu ra (output dimension) sau mỗi lớp tích chập là một yếu tố quan trọng trong thiết kế kiến trúc GCN, vì nó ảnh hưởng trực tiếp đến khả năng biểu diễn và hiệu quả huấn luyện của mô hình. Với lớp GCNConv đầu tiên, mục tiêu là biến đổi đặc trưng ban đầu của mỗi đỉnh thành một biểu diễn trừu tượng hơn, giàu ngữ nghĩa hơn. Do đó, số chiều đầu ra thường được chọn là một

giá trị tương đối cao như 64, 128 hoặc 256... Việc lựa chọn con số cụ thể phụ thuộc vào độ phức tạp của dữ liệu, khả năng tổng quát hóa, tài nguyên và tốc độ tính toán mong muốn. Ví dụ, nếu dữ liệu có nhiều loại quan hệ phức tạp giữa các đỉnh, việc dùng nhiều chiều hơn sẽ giúp mô hình học được nhiều mẫu hình tinh vi hơn. Về mặt toán học, khi chuyển từ đặc trưng đầu vào $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{d}}$ sang lớp ẩn đầu tiên $\boldsymbol{H}^{(\mathbf{1})} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{h}}$, ta thực hiện một phép biến đổi tuyến tính bởi trọng số $\boldsymbol{W}^{(\mathbf{0})} \in \mathbb{R}^{\boldsymbol{d}\times\boldsymbol{h}}$. Số chiều $\boldsymbol{h}$ ở đây là một siêu tham số mà người thiết kế mô hình cần điều chỉnh. Nếu chọn quá nhỏ, mô hình có thể không đủ khả năng biểu diễn, nếu quá lớn mô hình dễ bị overfitting và tốn tài nguyên.

Sau lớp tích chập đầu tiên, mô hình áp dụng hàm kích hoạt ReLU:

$$\mathbf{ReLU}(\boldsymbol{x}) = \mathbf{max}(\mathbf{0}, \boldsymbol{x}) \quad (3.1)$$

ReLU không chỉ giúp mô hình học được các đặc trưng phi tuyến mà còn tránh được vấn đề gradient biến mất (vanishing gradient) vốn thường gặp trong các hàm Sigmoid hoặc Tanh. Khi áp dụng ReLU sau lớp tích chập đầu tiên, mô hình loại bỏ các tín hiệu âm và chỉ giữ lại các giá trị dương có ý nghĩa, làm tăng độ biểu đạt của các đỉnh trong không gian đặc trưng.

Một trong những kỹ thuật quan trọng để ngăn chặn việc mô hình bị overfitting là sử dụng dropout. Trong kiến trúc này, dropout được áp dụng sau ReLU với xác suất $\boldsymbol{p} = 0.5$, nghĩa là 50% các node trong lớp ẩn sẽ bị "bỏ qua" (vô hiệu hóa) trong mỗi lần huấn luyện. Về mặt toán học, dropout là một phép nhân ngẫu nhiên giữa đầu ra của lớp và một vector nhị phân $\boldsymbol{m} \in \{0,1\}^h$ với mỗi phần tử tuân theo phân phối Bernoulli:

$$\tilde{\boldsymbol{H}}^{(\mathbf{1})} = \boldsymbol{m} \odot \boldsymbol{H}^{(\mathbf{1})}, \ \textbf{với } \boldsymbol{m_i} \sim \mathbf{Bernoulli}(\mathbf{1} - \boldsymbol{p}) \quad (3.2)$$

Kỹ thuật này giúp mô hình học được các biểu diễn ổn định và giảm hiện tượng phụ thuộc vào một vài node hoặc đặc trưng cụ thể.

Với lớp GCNConv thứ hai (lớp đầu ra), số chiều cần được chọn đúng bằng số lớp phân loại $\boldsymbol{c}$. Với bài toán yêu câu phân loại người dùng thành 2 nhóm khác nhau (real và fake) thì đầu ra của lớp cuối cùng phải có 2 chiều. Điều này là cần

thiết vì đầu ra cuối cùng thường sẽ được đưa qua hàm softmax để chuyển đổi thành xác suất cho từng lớp:

$$\hat{y}_i = \mathbf{softmax}(Z_i) = \frac{\exp\left(Z_{i,j}\right)}{\sum_{k=1}^{c} \exp\left(Z_{i,k}\right)} \text{ với } Z_i \in \mathbb{R}^c \quad (3.3)$$

Sự cân bằng giữa số chiều ở lớp ẩn và số lớp ở đầu ra là chìa khóa để mô hình vừa có khả năng học đặc trưng tốt, vừa tránh được việc học quá mức hoặc quá đơn giản hóa vấn đề. Đây là đầu ra chính để đánh giá mô hình trên bài toán phân loại đỉnh, và là đầu vào cho hàm mất mát cross-entropy khi huấn luyện.

Việc lựa chọn hai lớp GCN thể hiện một sự cân bằng hợp lý giữa độ sâu mạng và khả năng biểu diễn. Nếu chỉ có một lớp, mô hình sẽ chỉ học được thông tin từ hàng xóm bậc một, chưa đủ sâu để hiểu mối liên kết phức tạp giữa các nhóm người dùng. Nếu nhiều hơn hai lớp, mô hình có nguy cơ rơi vào hiện tượng oversmoothing, khi đó các biểu diễn của các nút trở nên gần giống nhau, làm mất khả năng phân biệt. Hướng thiết kế này cũng để ngỏ khả năng mở rộng. Ví dụ, có thể thay thế hàm tích chập GCN bằng các biến thể nâng cao như GAT, GraphSAGE… để cải thiện hiệu suất. Tương tự, dropout và các kỹ thuật regularization khác có thể được tinh chỉnh để phù hợp hơn với từng loại dữ liệu và nhiệm vụ cụ thể.

b. Graph Attention Network - GAT

In [6]:
```python
# Định nghĩa mô hình GAT
class GAT(torch.nn.Module):
    # Phương thức khởi tạo kiến trúc mô hình GAT
    def __init__(self, in_channels, hidden_channels, num_classes, heads=4):
        super().__init__()
        self.conv1 = GATConv(in_channels, hidden_channels, heads=heads)
        self.conv2 = GATConv(hidden_channels*heads, num_classes, heads=1, concat=False)

    # Phương thức forward để thực hiện quá trình lan truyền qua các lớp GATConv
    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.elu(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return x

# Khởi tạo mô hình GAT cụ thể
model = GAT(graph.num_node_features, 64, len(np.unique(users.dataset)))
display(model)
```

Hình 3.7: Mô hình GAT được lựa chọn

GAT cũng dựa trên nguyên lý lan truyền thông tin tương tự như GCN, tuy nhiên điểm khác biệt quan trọng của GAT so với GCN là ở cách tổng hợp thông

tin có trọng số, tức là mỗi đỉnh hàng xóm không đóng góp đồng đều mà được gán một hệ số trọng số phản ánh mức độ ảnh hưởng của nó đối với đỉnh trung tâm. Nhờ đó, mô hình GAT có thể thích ứng tốt hơn với cấu trúc không đồng đều của đồ thị, đặc biệt là khi một số đỉnh có vai trò quan trọng hơn nhiều so với các đỉnh lân cận khác.

Ở đây, mô hình GAT được lựa chọn xây dựng theo kiến trúc hai lớp GATConv. Tương tự như kiến trúc hai lớp GCN, việc sử dụng hai lớp GATConv cho phép mô hình học thông tin không chỉ từ các đỉnh hàng xóm bậc một mà còn từ các đỉnh hàng xóm bậc hai. Tại lớp đầu tiên, đặc trưng đầu vào $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{d}}$ được biến đổi thành biểu diễn trung gian $\boldsymbol{H}^{(\boldsymbol{1})} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{h}'}$, với $\boldsymbol{h}'$ là số chiều đầu ra sau khi kết hợp các attention head. Lớp thứ hai tiếp tục chuyển đổi từ $\boldsymbol{H}^{(\boldsymbol{1})}$ thành $\boldsymbol{Z} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{c}}$, với $\boldsymbol{c}$ là số lớp phân loại.

Điểm khác biệt lớn của GAT so với GCN là cách tính toán biểu diễn mới cho mỗi đỉnh không còn là trung bình (hoặc tổng có chuẩn hóa) của đặc trưng các hàng xóm, mà là tổ hợp tuyến tính có trọng số phụ thuộc vào cơ chế attention học được từ dữ liệu. Một điểm đáng chú ý khác trong GAT là việc sử dụng multi-head attention tức là tại mỗi lớp tích chập, thay vì chỉ có một bộ attention, mô hình sử dụng nhiều attention head song song, mỗi đầu học một cách nhìn khác nhau về các mối quan hệ trong đồ thị. Ví dụ, trong lớp đầu tiên, nếu số chiều ẩn là $\boldsymbol{h} = 64$ và số attention head là $\boldsymbol{K} = 4$, thì đầu ra của lớp sẽ có kích thước $\boldsymbol{h}' = \boldsymbol{h} \times \boldsymbol{K} = 256$ nếu concat = True hoặc kích thước $\boldsymbol{h}' = \boldsymbol{h}$ nếu *concat* = *False* (với giá trị trung bình các đầu). Cụ thể, trong mô hình đang xét, lớp GAT đầu tiên sử dụng *heads* = *4* và concat = True , nên đầu ra có kích thước $64 \times 4 = 256$. Điều này yêu cầu lớp thứ hai phải nhận đầu vào là 256 chiều.

Sau lớp tích chập đầu tiên, thay vì sử dụng ReLU như trong GCN, GAT sử dụng hàm kích hoạt ELU được định nghĩa như sau:

$$\mathbf{ELU}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x} & \text{nếu } \boldsymbol{x} > \boldsymbol{0} \\ \boldsymbol{\alpha}(\boldsymbol{exp}\,(\boldsymbol{x}) - \boldsymbol{1}) & \text{nếu } \boldsymbol{x} \leq \boldsymbol{0} \end{cases} \quad (3.4)$$

Hàm ELU giúp giữ lại một phần các giá trị âm, tránh hiện tượng "chết nút" như ReLU, đồng thời cải thiện độ trơn của hàm và độ lan truyền gradient. Đây là lựa chọn phổ biến trong các mạng có attention vì sự mềm dẻo trong biểu diễn, đặc biệt khi các đầu ra từ các lớp attention có thể mang ý nghĩa âm.

Tương tự như mô hình GCN, GAT cũng sử dụng dropout để giảm overfitting. Sau khi áp dụng hàm ELU, đầu ra được đưa qua dropout với xác suất $\boldsymbol{p} = 0.5$. Cơ chế dropout hoàn toàn giống GCN.

Ở lớp GATConv thứ hai được thiết lập với *heads = 1* và *concat = False*, do đó đầu ra chỉ có $\boldsymbol{c}$ chiều phù hợp với số lớp phân loại. Điều này đảm bảo đầu ra có thể trực tiếp đưa vào hàm softmax để tính xác suất phân lớp, tương tự như cách làm trong GCN .

Việc lựa chọn hai lớp GAT với cơ chế multi-head attention thể hiện một sự cân bằng hợp lý giữa độ sâu mạng và khả năng biểu diễn linh hoạt của mô hình. Nhờ multi-head attention, mỗi nút có thể tổng hợp thông tin từ hàng xóm theo nhiều góc nhìn khác nhau, giúp mô hình học được các đặc trưng đa dạng và giàu ngữ nghĩa hơn. Nếu chỉ sử dụng một lớp, mô hình có thể chưa đủ sâu để nắm bắt được các mối liên kết phức tạp giữa các nhóm người dùng. Tuy nhiên, nếu sử dụng quá nhiều lớp, đặc biệt trong bối cảnh attention được áp dụng đồng thời trên nhiều attention head, mô hình dễ rơi vào hiện tượng oversmoothing hoặc overfitting với dữ liệu huấn luyện. Do đó, thiết kế hai lớp GAT với multi-head attention là một lựa chọn hợp lý để vừa đảm bảo khả năng học biểu diễn mạnh, vừa hạn chế rủi ro từ mô hình quá sâu. Đồng thời, kiến trúc này cũng tạo điều kiện thuận lợi cho việc mở rộng và tùy biến, ví dụ như điều chỉnh số lượng attention head, kết hợp với các kỹ thuật như residual connection, layer normalization hay dropout nhằm cải thiện hiệu suất trên các tác vụ và tập dữ liệu khác nhau.

c. Graph Sample and Aggregate - GraphSAGE

In [6]:
```
# Định nghĩa mô hình GraphSAGE
class GraphSAGE(torch.nn.Module):
    # Phương thức khởi tạo kiến trúc mô hình GraphSAGE
    def __init__(self, in_channels, hidden_channels, num_classes):
        super().__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, num_classes)

    # Phương thức forward để thực hiện quá trình lan truyền qua các lớp SAGEConv
    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return x

# Khởi tạo mô hình GraphSAGE cụ thể
model = GraphSAGE(graph.num_node_features, 64, len(np.unique(users.dataset)))
display(model)
```

Hình 3.8: Mô hình GraphSAGE được lựa chọn

GraphSAGE là một biến thể quan trọng trong họ các mô hình mạng nơ-ron tích chập trên đồ thị. Mô hình này được phát triển với mục tiêu mở rộng khả năng học biểu diễn trên các đồ thị lớn, đặc biệt là trong bối cảnh dữ liệu liên kết ngày càng phức tạp và đồ thị có thể phát triển theo thời gian. Về mặt tổng thể, GraphSAGE cũng dựa trên nguyên lý lan truyền thông tin như GCN. Tuy nhiên, điểm khác biệt quan trọng nằm ở cách mô hình này lấy mẫu hàng xóm và tổng hợp thông tin, giúp nó có thể được huấn luyện theo mini-batch và áp dụng cho cả đồ thị động.

Ở đây, mô hình GraphSAGE được lựa chọn xây dựng theo kiến trúc hai lớp SAGEConv. Việc sử dụng hai lớp tích chập giống như GCN nhằm kết hợp thông tin không chỉ trong phạm vi hàng xóm bậc một mà còn lan truyền sang hàng xóm bậc hai. Tại lớp đầu tiên, đặc trưng đầu vào của các đỉnh được biến đổi thành một biểu diễn trung gian giàu ngữ nghĩa hơn, sau đó lớp thứ hai tiếp tục xử lý để cho ra biểu diễn cuối cùng phục vụ cho bài toán phân loại. Một đặc điểm nổi bật của GraphSAGE là cơ chế tổng hợp hàng xóm. Thay vì sử dụng công thức chuẩn hóa ma trận kề như GCN, GraphSAGE được thiết kế cơ chế lan truyền với hướng tiếp cận cục bộ và linh hoạt hơn bằng việc áp dụng một hàm tổng hợp phi tuyến, có thể tùy biến để kết hợp thông tin từ các hàng xóm của một đỉnh. Một số hàm Aggregate có thể được lựa chọn như:

**** Mean Aggregator:***

Đây là dạng mặc định sử dụng trong SAGEConv của PyTorch Geometric và được áp dụng trong mô hình được đề xuất:

$$\boldsymbol{h_v^{(k)} = \sigma\left(w^{(k)} \cdot MEAN\left(\left\{h_v^{(k-1)}\right\} \cup \left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right)\right)} \quad (3.5)$$

Trong đó:

$\boldsymbol{\mathcal{N}(v)}$: tập các đỉnh lân cận của đỉnh $\boldsymbol{v}$

$\boldsymbol{W^{(k)}}$: ma trận trọng số tại lớp $\boldsymbol{k}$

$\boldsymbol{\sigma}$: hàm kích hoạt (ví dụ: ReLU)

$\boldsymbol{h_v^{(k)}}$: biểu diễn của đỉnh $\boldsymbol{v}$ tại lớp $\boldsymbol{k}$

**** Max-Pooling Aggregator:***

Tổng hợp bằng cách áp dụng hàm ReLU sau đó lấy giá trị lớn nhất theo từng chiều.

$$\boldsymbol{h_v^{(k)} = \sigma\left(W^{(k)} \cdot MAX\left(\left\{ReLU\left(W_{pool} \cdot h_u^{(k-1)} + b\right), \forall u \in \mathcal{N}(v)\right\}\right)\right)} \quad (3.6)$$

Trong đó:

$\boldsymbol{W_{pool}}$: trọng số cho lớp ẩn của hàm **pooling**

MAX: lấy giá trị lớn nhất theo từng chiều

**** LSTM Aggregator:***

Một mạng LSTM (RNN) được áp dụng trên các vector hàng xóm, thứ tự của các vector có thể được xử lý ngẫu nhiên hoặc cố định

$$\mathbf{h}_v^{(k)} = \text{LSTM}\left(\left\{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right) \quad (3.7)$$

**** Mean Pooling Aggregator:***

Áp dụng một lớp tuyến tính trước khi trung bình.

$$\boldsymbol{h_v^{(k)} = \sigma\left(W^{(k)} \cdot MEAN\left(\left\{\phi\left(h_u^{(k-1)}\right), \forall u \in \mathcal{N}(v)\right\}\right)\right)} \quad (3.8)$$

Trong đó:

$$\boldsymbol{\phi\left(h_u^{(k-1)}\right) = Linear\left(h^{\downarrow 1)}\right) = W_\phi \cdot h_u^{(k-1)} + b} \quad (3.9)$$

Lựa chọn hàm tổng hợp phù hợp sẽ ảnh hưởng trực tiếp đến biểu diễn và hiệu quả mô hình, nhất là khi làm việc với các đồ thị không đồng nhất hoặc có tính chất riêng biệt mạnh.

Quá trình lý lan truyền thông tin trên mô hình GraphSAGE được mô tả:

$$\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{d}} \overset{\boldsymbol{SAGEConv}}{\rightarrow} \boldsymbol{H}^{(\boldsymbol{1})} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{h}} \overset{\boldsymbol{SAGEConv}}{\rightarrow} \boldsymbol{Z} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{c}}$$

Trong đó:

- $\boldsymbol{W}^{(\boldsymbol{0})} \in \mathbb{R}^{\boldsymbol{d}\times\boldsymbol{h}}$ là trọng số lớp đầu ($\boldsymbol{W}^{(\boldsymbol{0})} \in \mathbb{R}^{\boldsymbol{2d}\times\boldsymbol{h}}$ nếu sử dụng *concat*)
- $\boldsymbol{W}^{(\boldsymbol{1})} \in \mathbb{R}^{\boldsymbol{h}\times\boldsymbol{c}}$ là trọng số lớp thứ hai ($\boldsymbol{W}^{(\boldsymbol{1})} \in \mathbb{R}^{\boldsymbol{2h}\times\boldsymbol{c}}$ nếu sử dụng *concat*)

Cũng như GCN việc lựa chọn số chiều đầu ra sau mỗi lớp tích chập là một yếu tố quan trọng trong thiết kế kiến trúc GraphSAGE, vì nó ảnh hưởng trực tiếp đến khả năng biểu diễn và hiệu quả huấn luyện của mô hình. Đặc biệt với lớp SAGEConv thứ nhất, chọn số chiều đầu ra $\boldsymbol{h}$ phù hợp sẽ giúp mô hình có khả năng học các đặc trưng phức tạp đồng thời giảm overfitting, đặc biệt nếu số lượng đỉnh hoặc mẫu huấn luyện là hạn chế. Sau lớp tính chập đầu tiên, mô hình cũng áp dụng hàm kích hoạt ReLU và sau đó đưa qua dropout với xác suất $\boldsymbol{p} = 0.5$ với chức năng tương tự như GCN.

Ở lớp SAGEConv thứ hai, số chiều cần được chọn đúng bằng số lớp phân loại $\boldsymbol{c}$ tương tự GCN để đảm bảo đầu ra có thể trực tiếp đưa vào hàm softmax để tính toán xác suất phân lớp.

Việc lựa chọn hai lớp GraphSAGE thể hiện một sự cần bằng hợp lý giữa khả năng biểu diễn và hiệu quả tính toán. Nếu chỉ sử dụng một lớp, mô hình chỉ có thể lan truyền thông tin trong phạm vi hàng xóm bậc một, điều này chưa đủ để học các mối liên hệ sâu rộng hơn. Ngược lại, dùng nhiều hơn hai lớp, mô hình có thể bị oversmoothing, làm giảm khả năng phân biệt giữa các lớp. Điều này cũng tương tự như trong GCN. Một trong những ưu điểm vượt trội nhất của GraphSAGE là khả năng mở rộng tốt hơn. Bằng cách lấy mẫu hàng xóm ngẫu nhiên trong quá trình huấn luyện, GraphSAGE có thể áp dụng cho đồ thị rất lớn mà không cần tải toàn bộ đồ thị vào bộ nhớ. Ngoài ra, có thể mở rộng lựa chọn huấn luyện GraphSAGE theo mini-batch hoặc lấy mẫu lân cận kích thước cố định

giúp giảm đáng kể chi phí bộ nhớ và tăng tốc độ huấn luyện, đặc biệt hiệu quả trong các bài toán thời gian thực hoặc đồ thị có quy mô hàng triệu đỉnh.

3.3.2. Phân chia dữ liệu huấn luyện

Bước tiếp theo trong quy trình huấn luyện mô hình là chia tập dữ liệu thành các phần huấn luyện, xác thực và kiểm tra. Việc chia tách này đóng vai trò quan trọng trong việc đánh giá hiệu quả tổng quát hóa của mô hình, nâng cao khả năng trong việc đưa ra dự đoán đúng trên dữ liệu chưa từng thấy trong quá trình huấn luyện. Cụ thể, toàn bộ tập dữ liệu người dùng được phân chia làm ba phần:

- Tập huấn luyện dùng để cập nhật tham số của mô hình.
- Tập xác thực giúp theo dõi hiệu suất trong quá trình huấn luyện.
- Tập kiểm tra được sử dụng sau cùng để đánh giá khả năng thực tế của mô hình khi áp dụng vào môi trường mới.

Cách phân chia hợp lý và cân bằng các nhãn giữa các tập sẽ đảm bảo rằng mô hình không bị thiên lệch và có khả năng tổng quát tốt. Trong giai đoạn này, tổng số lượng đỉnh (tức là người dùng trong đồ thị) được xác định từ đối tượng đồ thị đã tạo. Đồng thời, nhãn phân loại của người dùng cũng được chuyển sang định dạng mảng để thuận tiện cho việc chia tách. Quá trình phân chia cần đảm bảo sự cân bằng giữa các lớp, nghĩa là các tập con được phân chia sao cho tỉ lệ giữa các nhãn được giữ ổn định. Trước tiên, khoảng 30% số lượng đỉnh được tách riêng ra để làm tập xác thực và kiểm tra. Sau đó, một nửa sẽ được dùng làm tập xác thực, còn một nửa làm tập kiểm tra. Như vậy, cuối cùng ta có 70% dữ liệu cho huấn luyện và 15% cho mỗi phần xác thực và kiểm tra.

Để thực hiện việc đánh dấu các đỉnh thuộc từng tập, ba mặt nạ Boolean được tạo ra. Mỗi mặt nạ là một danh sách các giá trị đúng - sai (True - False), có chiều dài bằng tổng số đỉnh trong đồ thị. Mặt nạ huấn luyện sẽ có giá trị “True” tại các chỉ số của các đỉnh thuộc tập huấn luyện và “False” ở các vị trí còn lại. Tương tự, mặt nạ xác thực và kiểm tra cũng đánh dấu những đỉnh thuộc về hai tập tương ứng. Việc sử dụng mặt nạ thay vì chia nhỏ dữ liệu là phương pháp đặc trưng trong huấn luyện mô hình học sâu trên đồ thị. Trong các mô hình truyền thống,

dữ liệu có thể được chia nhỏ bằng cách lọc hoặc cắt mảng. Tuy nhiên, trong học sâu trên đồ thị, tất cả các đỉnh và cạnh vẫn cần được giữ nguyên trong cấu trúc để đảm bảo tính toàn vẹn và kết nối của mạng. Do đó, việc sử dụng mặt nạ cho phép mô hình học trên toàn bộ đồ thị nhưng chỉ cập nhật tham số hoặc tính toán lỗi dựa trên các đỉnh được đánh dấu.

Sau khi tạo xong ba mặt nạ, chúng được gán trực tiếp vào đối tượng đồ thị dưới tên gọi riêng cho từng loại: *train_mask, val_mask và test_mask*. Điều này giúp mô hình truy cập nhanh chóng đến thông tin về các đỉnh nào cần được sử dụng trong từng giai đoạn huấn luyện, kiểm định hoặc kiểm tra. Nhờ đó, quá trình xử lý dữ liệu trong mỗi epoch trở nên rõ ràng, tách biệt và hiệu quả hơn.

```
In [5]: # Chia dữ liệu thành tập huấn luyện và tập kiểm tra
num_nodes = graph.num_nodes
labels = users.dataset.to_numpy()
train, temp = train_test_split(
    range(num_nodes),
    test_size=0.3,
    random_state=42,
    stratify=labels
)
val, test = train_test_split(
    temp,
    test_size=0.5,
    random_state=42,
    stratify=labels[temp]
)

# Tạo mask cho tập huấn luyện và tập kiểm tra
train_mask = torch.zeros(num_nodes, dtype=torch.bool)
val_mask = torch.zeros(num_nodes, dtype=torch.bool)
test_mask = torch.zeros(num_nodes, dtype=torch.bool)

# Gán giá trị True cho các chỉ mục trong tập huấn luyện và tập kiểm tra
train_mask[train] = True
val_mask[val] = True
test_mask[test] = True

# Gán các mask vào đối tượng graph
graph.train_mask = train_mask
graph.val_mask = val_mask
graph.test_mask = test_mask
print(f"Train nodes: {graph.train_mask.sum()}, Val nodes: {graph.val_mask.sum()}, Test nodes: {graph.test_mask.sum()}")
```

```
Train nodes: 1972, Val nodes: 423, Test nodes: 423
```

Hình 3.9: Phân chia dữ liệu huấn luyện

Việc chia dữ liệu một cách có kiểm soát như vậy không chỉ giúp đảm bảo tính khách quan trong đánh giá mô hình mà còn tạo điều kiện để thực hiện các kỹ thuật tinh chỉnh như điều chỉnh siêu tham số, chọn kiến trúc mạng phù hợp hoặc thực hiện kiểm chứng chéo nếu cần. Ngoài ra, với mô hình học sâu trên đồ thị, nơi mà mối quan hệ giữa các đỉnh đóng vai trò thiết yếu, việc giữ nguyên toàn bộ cấu trúc đồ thị và phân tách bằng mặt nạ là lựa chọn tối ưu để đảm bảo tính chính xác và toàn vẹn trong suốt quá trình huấn luyện và đánh giá.

3.4. Cài đặt huấn luyện mô hình

In [7]:
```
# Khởi tạo bộ tối ưu hóa Adam với mô hình và các tham số học
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
display(optimizer)

# Phương thức huấn luyện mô hình
def train():
    model.train()
    optimizer.zero_grad()
    out = model(graph.x, graph.edge_index)
    loss = F.cross_entropy(out[graph.train_mask], graph.y[graph.train_mask])
    loss.backward()
    optimizer.step()
    return loss.item()

# Phương thức kiểm tra mô hình
def test():
    model.eval()
    with torch.no_grad():
        out = model(graph.x, graph.edge_index)
        pred = out.argmax(dim=1)
    accs = []
    for mask in [graph.train_mask, graph.val_mask, graph.test_mask]:
        correct = pred[mask] == graph.y[mask]
        acc = int(correct.sum()) / int(mask.sum())
        accs.append(acc)
    return accs
```

```
Adam (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    capturable: False
    decoupled_weight_decay: False
    differentiable: False
    eps: 1e-08
    foreach: None
    fused: None
    lr: 0.01
    maximize: False
    weight_decay: 0.0005
)
```

Hình 3.10: Cài đặt các tham số, phương thức huấn luyện, kiểm thử mô hình

3.4.1. Các siêu tham số huấn luyện

Trong quá trình huấn luyện mô hình, một trong những yếu tố then chốt ảnh hưởng đến hiệu quả học tập và khả năng khái quát hóa của mô hình là lựa chọn bộ tối ưu (optimizer) cùng với các siêu tham số đi kèm. Thuật toán tối ưu *Adam* cùng với hai siêu tham số quan trọng *learning rate* và *weight decay* sẽ được sử dụng cho các mô hình GNN đã được đề xuất. Việc hiểu rõ bản chất và vai trò của hai siêu tham số này là cần thiết để đảm bảo mô hình học tốt và tránh các vấn đề như học quá chậm, hội tụ sai, overfitting.

Thuật toán Adam là một trong những phương pháp tối ưu phổ biến và hiệu quả nhất trong học sâu hiện nay. Adam kết hợp giữa hai phương pháp truyền thống là momentum và RMSProp, cho phép điều chỉnh tốc độ học (learning rate) một cách thích nghi cho từng tham số của mô hình. Điều này giúp Adam hội tụ

nhanh hơn và ổn định hơn so với các thuật toán đơn giản như Gradient Descent hoặc SGD truyền thống. Cốt lối của Adam là sử dụng hai trung bình động:

- Trung bình động bậc nhất của gradient (momentum):

$$\boldsymbol{m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t} \quad (3.10)$$

- Trung bình động bậc hai của bình phương gradient:

$$\boldsymbol{v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2} \quad (3.11)$$

Trong đó $\boldsymbol{g_t}$ là gradient tại thời điểm $\boldsymbol{t}$, và $\boldsymbol{\beta_1, \beta_2}$ là các hệ số điều chỉnh tốc độ cập nhật. Các giá trị này được cài đặt mặc định trong PyTorch là $\boldsymbol{\beta_1} = 0.9$ và $\boldsymbol{\beta_2} = 0.999$. Cuối cùng, bước cập nhật tham số được thực hiện như sau:

$$\boldsymbol{\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}} \quad (3.12)$$

Với:

- $\boldsymbol{\eta}$ là learning rate
- $\boldsymbol{\hat{m}_t}$ và $\boldsymbol{\hat{v}_t}$ là các giá trị bias-corrected tương ứng với $\boldsymbol{m_t}$ và $\boldsymbol{v_t}$
- $\boldsymbol{\epsilon}$ là một hằng số nhỏ để tránh chia cho 0 (thường là 10^{-8})

Learning rate là một trong những siêu tham số quan trọng nhất trong mọi mô hình học sâu. Nó quyết định mức độ thay đổi của trọng số mô hình trong mỗi bước cập nhật. Nếu tốc độ học quá lớn, mô hình có thể vượt quá cực tiểu toàn cục và không hội tụ. Ngược lại, nếu tốc độ học quá nhỏ, quá trình học sẽ rất chậm hoặc mắc kẹt tại cực tiểu cục bộ. Tốc độ học được lựa chọn là 0.01, tức là mức điều chỉnh khá điển hình đối với Adam trong các bài toán học có cấu trúc dữ liệu đồ thị. Lý do của lựa chọn này có thể được lý giải như sau:

- Dữ liệu đồ thị thường có độ phức tạp cao, nhiều lớp liên kết, dẫn đến gradient có thể dao động mạnh. Adam đã giúp ổn định điều này thông qua các trung bình động, nên ta có thể sử dụng một tốc độ học tương đối cao như 0.01 mà vẫn đảm bảo an toàn.

- Trong các nghiên cứu và benchmark với các mạng GCN, GAT giá trị learning rate phổ biến nằm trong khoảng từ 0.005 đến 0.02. Việc chọn 0.01 nằm

ở giữa khoảng này và đã được chứng minh là ổn định cho nhiều tập dữ liệu chuẩn như Cora, Citeseer, PubMed.

- Một tốc độ học vừa phải như vậy giúp mô hình học nhanh trong các epoch đầu tiên nhưng không quá lớn để gây mất ổn định trong các giai đoạn về sau.

Weight decay là một kỹ thuật chuẩn hóa giúp mô hình không học quá phức tạp và tránh overfitting. Về bản chất, weight decay thêm một thành phần phạt (penalty term) vào hàm mất mát, thường là L2-norm của trọng số:

$$\boldsymbol{L_{\text{total}} = L_{\text{original}} + \lambda \cdot \|\theta\|^2} \quad (3.13)$$

Trong đó:

- $\boldsymbol{L_{\text{original}}}$ là hàm mất mát (ví dụ cross-entropy)

- $\boldsymbol{\lambda}$ là hệ số weight decay

- $\boldsymbol{\|\theta\|^2}$ là tổng bình phương các trọng số của mô hình

Mục tiêu của weight decay là làm nhỏ giá trị trọng số, từ đó tránh việc mô hình học quá nhiều đặc điểm nhiễu từ dữ liệu huấn luyện. Trong bối cảnh phân loại tài khoản mạng xã hội, dữ liệu thường rất đa dạng, thậm chí chứa nhiễu như tài khoản bot ngụy trang hay tài khoản người thật hoạt động ít. Việc sử dụng weight decay giúp mô hình tập trung vào các đặc trưng tổng quát, thay vì học những tín hiệu riêng biệt dễ dẫn đến overfitting. Giá trị 0.0005 là mức nhẹ nhưng đủ hiệu quả, thường được sử dụng trong các mô hình GNN như GCN, GraphSAGE với các ưu điểm:

- Giảm hiện tượng gradient exploding.

- Tăng khả năng khái quát của mô hình khi đánh giá trên dữ liệu validation và test.

- Giảm rủi ro học sai trọng số lớn gây ra độ lệch trong dự đoán.

Dù là hai tham số riêng biệt, learning rate và weight decay có mối quan hệ tương hỗ với nhau. Chúng đều ảnh hưởng đến quá trình cập nhật trọng số nhưng theo cách khác nhau. Learning rate điều chỉnh mức độ thay đổi của trọng số theo gradient. Weight decay điều chỉnh xu hướng co nhỏ trọng số để kiểm soát độ phức tạp. Nếu learning rate lớn, thì tác động của weight decay sẽ bị "lấn át" vì mỗi bước

cập nhật trọng số sẽ lớn hơn hiệu ứng thu nhỏ của weight decay. Ngược lại, nếu learning rate quá nhỏ, mô hình sẽ bị điều chỉnh quá nhẹ và weight decay có thể làm cho trọng số bị "làm mòn" về 0 quá sớm. Việc lựa chọn learning rate là 0.01 và weight decay là 0.0005 phản ánh sự cân bằng hợp lý giữa tốc độ học đủ nhanh để giảm loss nhanh chóng và hiệu ứng regularization nhẹ nhưng hiệu quả để giảm overfitting mà không ngăn mô hình học được các đặc trưng quan trọng.

3.4.2. Phương thức huấn luyện và kiểm thử

Huấn luyện mô hình là quá trình mà hệ thống học máy sử dụng để tự động điều chỉnh các tham số bên trong mạng nơ-ron thông qua việc quan sát dữ liệu đầu vào kèm theo nhãn đúng. Trong ngữ cảnh cụ thể của bài toán, các đối tượng cần phân loại là các tài khoản mạng xã hội, được biểu diễn dưới dạng đỉnh trong đồ thị, còn mối liên kết bạn bè giữa các tài khoản (hoặc followers) là các cạnh nối giữa các đỉnh đó. Như vậy, mỗi đỉnh sẽ mang theo một số đặc trưng đầu vào và một nhãn phân loại (real hoặc fake). Trong quá trình này, mô hình sẽ trải qua nhiều chu kỳ truyền và tích lũy thông tin, từ đó xây dựng được biểu diễn đặc trưng mới cho mỗi đỉnh, phản ánh ngữ cảnh cấu trúc và nội dung của đỉnh đó trong đồ thị. Khi có được biểu diễn đặc trưng này, hệ thống sẽ tiếp tục sử dụng chúng để dự đoán nhãn đầu ra cho các đỉnh, đồng thời so sánh với nhãn đúng để tính toán sai số dự đoán (hàm mất mát).

Cụ thể, quy trình huấn luyện bắt đầu bằng việc đưa mô hình vào trạng thái sẵn sàng học. Tiếp theo, các tham số trong mô hình được thiết lập lại về giá trị khởi tạo nhằm đảm bảo không có ảnh hưởng từ các lần học trước. Sau đó, dữ liệu đồ thị bao gồm đặc trưng các đỉnh và cấu trúc cạnh sẽ được đưa vào mô hình để tính toán đầu ra dự đoán. Sai số giữa đầu ra này và nhãn thực tế được đo lường thông qua một hàm mất mát. Ở đây, hàm mất mát được lựa chọn là cross-entropy được sử dụng phổ biến trong các bài toán phân loại, đặc biệt là phân loại nhiều lớp. Nó đo lường sự khác biệt giữa phân phối xác suất dự đoán của mô hình và phân phối thực tế. Với một mẫu đầu vào, nếu nhãn thật là $\boldsymbol{y}$ và đầu ra của mô hình là xác suất $\hat{\boldsymbol{y}}$, thì:

$$\boldsymbol{Loss} = -\sum_{\boldsymbol{i=1}}^{\boldsymbol{C}} \boldsymbol{y_i log\ (\hat{y}_i)} \quad (3.14)$$

Trong đó:

- $\boldsymbol{C}$ là số lớp
- $\boldsymbol{y}$ là vector biểu diễn nhãn thật
- $\boldsymbol{\hat{y}}$ là vector dự đoán sau softmax (phân phối xác suất trên các lớp)

Khi chỉ có một nhãn đúng (one-hot), công thức rút gọn thành:

$$\boldsymbol{Loss} = -\boldsymbol{log\ (\hat{y}_{true})} \quad (3.15)$$

Giá trị loss càng nhỏ mô hình dự đoán đúng với xác suất cao. Sai số này đóng vai trò là tín hiệu phản hồi, giúp hệ thống cập nhật các tham số bên trong mô hình nhằm giảm thiểu sai số ở các vòng huấn luyện tiếp theo. Sai số thu được sẽ được lan ngược lại qua mạng nơ-ron theo thuật toán lan truyền ngược giúp điều chỉnh các tham số theo hướng tối ưu hóa mô hình. Cuối cùng, một bước tối ưu được thực hiện để cập nhật các tham số, đánh dấu sự kết thúc của một chu kỳ huấn luyện. Giá trị của sai số này sau mỗi chu kỳ được ghi lại để theo dõi tiến trình học.

Bên cạnh quá trình huấn luyện, quá trình kiểm thử mô hình là bước không thể thiếu nhằm đánh giá độ chính xác và khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy. Quá trình này đóng vai trò quan trọng trong việc xác định xem mô hình học được có thực sự hiểu bản chất của dữ liệu hay chỉ đơn thuần ghi nhớ các mẫu đã được huấn luyện. Trong quy trình kiểm thử, mô hình sẽ được đưa vào trạng thái chỉ sử dụng, tức là không có bất kỳ cập nhật tham số nào xảy ra để đảm bảo rằng quá trình kiểm thử không ảnh hưởng tới quá trình học đã diễn ra trước đó. Quy trình kiểm thử được tiến hành bằng cách đưa vào mô hình toàn bộ cấu trúc đồ thị, sau đó tiến hành dự đoán nhãn cho các đỉnh trong đồ thị. Dự đoán được so sánh với nhãn thực tế để tính toán tỷ lệ chính xác trên ba tập dữ liệu huấn luyện, xác thực và kiểm tra. Trong quá trình kiểm thử, với mỗi tập hệ thống sẽ so sánh dự đoán của mô hình với nhãn đúng để tính số lượng dự đoán chính xác. Từ đó, tỷ lệ chính xác được tính bằng số lượng dự đoán đúng chia cho tổng số đỉnh

trong tập tương ứng. Kết quả của các phép đo này được ghi nhận và trả về như là thước đo hiệu năng tổng quát của mô hình.

3.5. Huấn luyện, đánh giá hiệu suất mô hình

3.5.1. Chiến lược huấn luyện

```
In [8]: # Khởi tạo DataFrame để lưu lịch sử huấn luyện
history = pd.DataFrame(columns=['epoch', 'loss', 'train_acc', 'val_acc', 'test_acc'])

# Khởi tạo các giá trị phục vụ lưu kết quả tối ưu nhất
best_val_acc = 0
patience = 50
counter = 0
best_model_state = None

# Huấn luyện mô hình và lưu lịch sử huấn luyện
for epoch in range(1, 101):
    loss = train()
    train_acc, val_acc, test_acc = test()
    print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}, Train Acc: {train_acc:.4f},  Val Acc: {val_acc:.4f}, Test Acc: {test_acc:.4f}')
    history.loc[len(history)] = {
        'epoch': epoch,
        'loss': loss,
        'train_acc': train_acc,
        'val_acc': val_acc,
        'test_acc': test_acc
    }
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        counter = 0
        best_model_state = model.state_dict()
    else:
        counter += 1
    if counter >= patience:
        print(f"GCN Early stopped at epoch {epoch} with Val Acc = {best_val_acc:.4f}")
        break
    if epoch == 100:
        print(f"GCN stopped at epoch {epoch} with Val Acc = {best_val_acc:.4f}")
```

Hình 3.11: Chiến lược huấn luyện mô hình

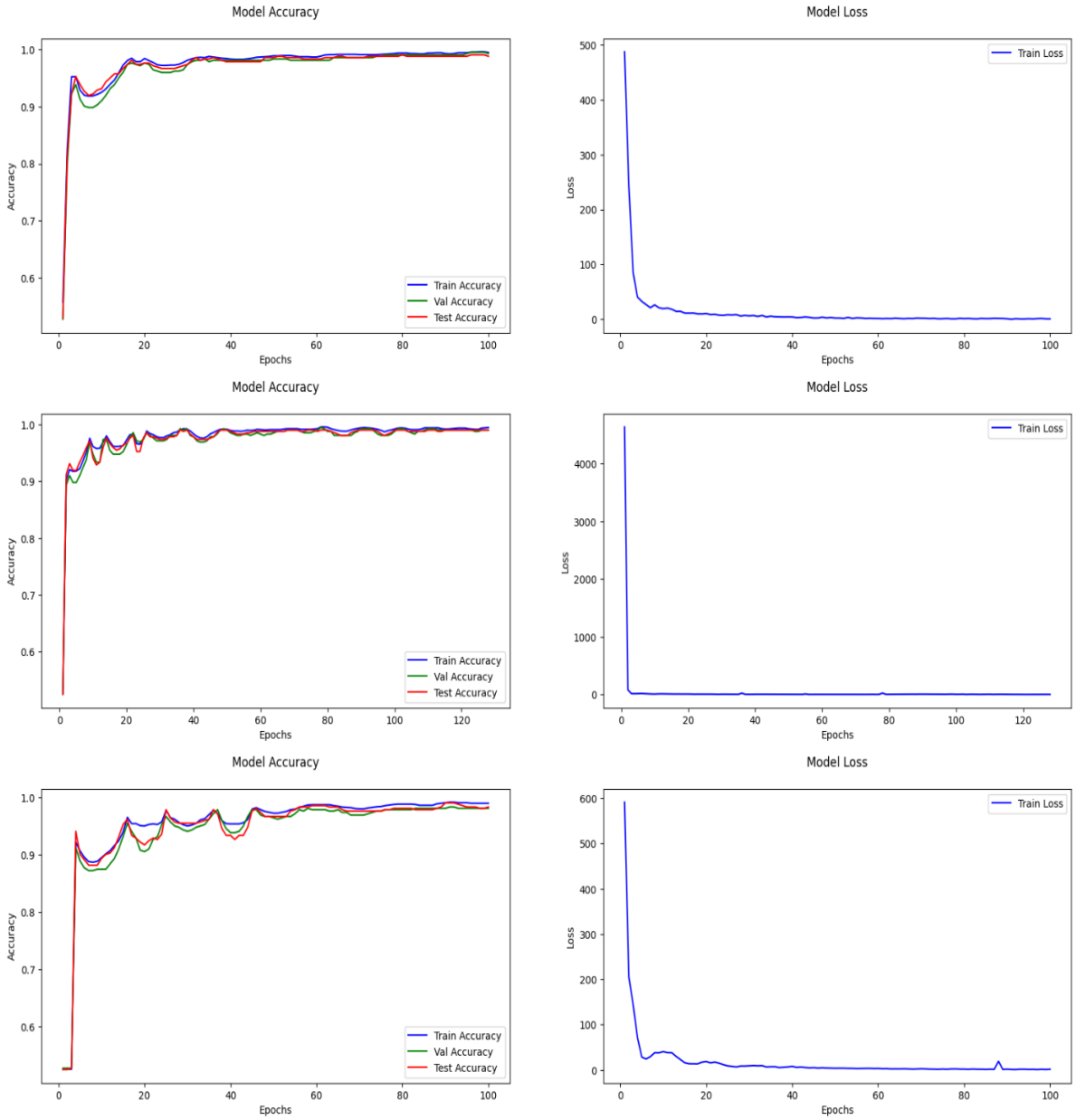
Mặc dù mỗi mô hình GNN được lựa chọn có kiến trúc và đặc điểm tính toán riêng biệt, điểm chung lớn nhất giữa chúng là đều có thể áp dụng chung một quy trình huấn luyện. Mục tiêu lớn nhất khi huấn luyện mô hình không đơn thuần là huấn luyện mô hình theo số epoch cố định, mà là xây dựng một cơ chế huấn luyện thông minh, có khả năng theo dõi, đánh giá, tối ưu hóa và tự động dừng lại khi điều kiện tốt nhất đã đạt được. Chính điều đó giúp quá trình huấn luyện tiết kiệm thời gian, tránh overfitting đảm bảo chất lượng mô hình đạt ngưỡng tối ưu. Mỗi epoch huấn luyện sẽ được theo dõi một cách chi tiết thông qua việc ghi lại các chỉ số quan trọng như độ mất mát (loss) và độ chính xác (accuracy) trên ba tập dữ liệu huấn luyện, xác thực và kiểm tra. Các chỉ số này phản ánh sự thay đổi của mô hình trong suốt quá trình học, từ đó cung cấp một cái nhìn toàn diện về khả năng học, mức độ tổng quát hóa và xu hướng hội tụ của mô hình. Điều này không chỉ hỗ trợ đánh giá hiệu suất huấn luyện, mà còn là cơ sở để đưa ra các quyết định liên quan đến việc điều chỉnh siêu tham số hay thay đổi chiến lược huấn luyện.

Trong huấn luyện mô hình, việc theo dõi tập xác thực có ý nghĩa quan trọng hơn cả, bởi lẽ mục tiêu cuối cùng không phải là đạt hiệu suất cao nhất trên tập huấn luyện mà là xây dựng một mô hình có khả năng khái quát tốt trên dữ liệu chưa từng thấy. Tập xác thực đóng vai trò như một "bài kiểm tra tạm thời" giúp theo dõi xem mô hình có đang học hiệu quả hay không, và nếu độ chính xác trên tập này không còn cải thiện qua nhiều epoch liên tiếp thì có thể mô hình đã đạt tới giới hạn học tập tự nhiên. Từ đó, chiến lược dừng sớm được triển khai như một biện pháp chủ động ngắt quá trình huấn luyện khi không còn cải thiện nào trên tập xác thực. Mỗi khi có vòng huấn luyện mới mà độ chính xác không cao hơn mức tốt nhất đã đạt được trước đó, biến đếm sẽ tăng thêm một đơn vị. Ngược lại, nếu có sự cải thiện, biến đếm sẽ được đưa về lại giá trị 0 và mô hình tại thời điểm đó sẽ được lưu lại như mô hình tốt nhất. Khi biến đếm vượt qua một ngưỡng xác định trước (ở đây là 50), quá trình huấn luyện sẽ bị ngắt ngay lập tức. Cách làm này không chỉ giúp tiết kiệm thời gian mà còn giúp tránh được tình trạng mô hình trở nên quá khớp với tập huấn luyện, làm giảm hiệu suất tổng quát trên dữ liệu thực tế. Tập xác thực trong trường hợp này chính là tiêu chuẩn đánh giá chất lượng mô hình tại mỗi thời điểm. Nếu mô hình thực sự đang học, thì độ chính xác trên tập xác thực phải tăng lên theo thời gian. Tuy nhiên, nếu mô hình chỉ lặp đi lặp lại những gì đã học, hoặc thậm chí bắt đầu ghi nhớ chi tiết của tập huấn luyện, thì độ chính xác trên tập xác thực sẽ không cải thiện hoặc thậm chí suy giảm. Dừng sớm hoạt động như một cơ chế bảo vệ mô hình khỏi việc học quá sâu vào dữ liệu huấn luyện và cho phép ta giữ lại mô hình tại thời điểm hiệu quả nhất.

Về số lượng epoch tối đa được lựa chọn là 100 cho các mô hình GCN, GraphSAGE, nhưng riêng với GAT, số epoch được tăng lên 200. Đây là một điều chỉnh hợp lý bởi mô hình GAT sử dụng cơ chế attention phức tạp hơn, thường yêu cầu nhiều thời gian huấn luyện hơn để có thể hội tụ tốt. Tuy nhiên, việc áp dụng dừng sớm vẫn được giữ nguyên trong cả ba mô hình, đảm bảo rằng dù số vòng tối đa cao hơn, mô hình vẫn sẽ không bị ép học đến hết toàn bộ số vòng nếu không còn mang lại hiệu quả. Sự kết hợp giữa vòng lặp tối đa và dừng sớm tạo nên một

hệ thống huấn luyện linh hoạt, vừa kiểm soát tốt thời gian, vừa đảm bảo chất lượng mô hình.

3.5.2. Kết quả huấn luyện và so sánh với các phương pháp truyền thống



Hình 3.12: Biểu đồ kết quả huấn luyện, thứ tự GCN, GAT, GraphSAGE

Kết quả huấn luyện các mô hình GNN được đề xuất và một số mô hình học máy truyền thống khác (nguồn: *https://github.com/Vinod-Ghanchi/Social-Media-Fake-Account-Detection?tab=readme-ov-file*) được trình bày trong **bảng 3.1**:

Model	Accuracy
GCN (test size 0.3)	**99.29%**
GAT (test size 0.3)	**99.29%**
GraphSAGE (test size 0.3)	**98.7%**
SVM (test size 0.2)	90.07%
Decision Tree (test size 0.4)	93.08%
ADA Boost (test size 0.3)	92.76%
Random Forest (test size 0.2)	94.50%
XG Boost (test size 0.3)	93.38%

Bảng 3.1: Kết quả huấn luyện một số mô hình

Trong quá trình thực nghiệm, ba mô hình mạng nơ-ron đồ thị gồm Graph Convolutional Network (GCN), Graph Attention Network (GAT) và GraphSAGE đã được huấn luyện và đánh giá trên bài toán phân loại tài khoản mạng xã hội. Kết quả cho thấy cả ba mô hình đều thể hiện hiệu suất học tập ấn tượng, với độ chính xác (accuracy) trên tập kiểm tra đạt mức rất cao:

- GCN (test size = 0.3): 99.29%
- GAT (test size = 0.3): 99.29%
- GraphSAGE (test size = 0.3): 98.70%

Quá trình huấn luyện được ghi nhận thông qua biểu đồ độ chính xác theo từng epoch cho thấy cả ba mô hình đều hội tụ nhanh, độ chính xác tăng ổn định qua từng epoch và không xuất hiện dấu hiệu overfitting rõ rệt. Điều này cho thấy khả năng tổng quát hóa tốt của các mô hình GNN trong việc học từ dữ liệu đồ thị phức tạp. Xem xét và so sánh chi tiết từng mô hình GNN ta nhận thấy:

GCN: Với kiến trúc đơn giản nhưng hiệu quả, GCN hoạt động dựa trên phép tích chập (convolution) trên đồ thị, cho phép truyền và tích lũy thông tin giữa các node thông qua cấu trúc liên kết (adjacency). Dù không sử dụng cơ chế attention, GCN vẫn đạt độ chính xác rất cao (99.29%), cho thấy sự phù hợp của kiến trúc này với bài toán.

GAT: GAT bổ sung thêm cơ chế attention, cho phép mô hình học được trọng số tương đối giữa các node lân cận thay vì coi chúng như nhau. Mặc dù GAT được kỳ vọng sẽ vượt trội hơn GCN nhờ khả năng chú ý (attention mechanism), kết quả trong thực nghiệm lại cho thấy độ chính xác không có sự khác biệt rõ rệt (cùng đạt 99.29%). Điều này có thể xuất phát từ đặc điểm của tập dữ liệu, trong đó các mối liên kết giữa các node đã mang tính phân biệt rõ ràng mà không cần trọng số bổ sung.

GraphSAGE: Khác với GCN và GAT vốn yêu cầu toàn bộ đồ thị trong quá trình huấn luyện, GraphSAGE sử dụng cơ chế lấy mẫu node lân cận (neighbor sampling) và tổng hợp thông tin từ các hàng xóm theo từng batch, giúp mô hình có khả năng mở rộng tốt hơn cho đồ thị lớn. Mặc dù độ chính xác thấp hơn một chút (98.70%), GraphSAGE vẫn là một lựa chọn rất tiềm năng cho các bài toán xử lý đồ thị quy mô lớn.

**** So sánh với các mô hình học máy truyền thống***

Để đánh giá tính ưu việt của các mô hình GNN, việc huấn luyện và đánh giá một loạt mô hình học máy truyền thống trên cùng bài toán cũng đã được tiến hành. Mặc dù các mô hình truyền thống như Random Forest và XGBoost đạt kết quả khá tốt (trên 93%), tất cả đều thấp hơn đáng kể so với các mô hình GNN. Sự khác biệt này phần lớn đến từ bản chất dữ liệu, trong mạng xã hội, thông tin kết nối giữa các tài khoản (edges) chứa đựng ý nghĩa quan trọng mà các mô hình truyền thống không thể khai thác hiệu quả, do chúng không có cơ chế xử lý dữ liệu có cấu trúc liên kết. Ngược lại, GNN tận dụng triệt để cấu trúc đồ thị của dữ liệu, mỗi tài khoản không chỉ được biểu diễn qua đặc trưng riêng mà còn qua mối quan hệ với các tài khoản khác. Khả năng học được từ cả đặc trưng nội tại (node features) và cấu trúc liên kết (graph structure) là điểm then chốt giúp GNN đạt hiệu quả vượt trội.

3.5.3. Đánh giá hiệu suất mô hình

Precision, **Recall** và **F1-score** là ba chỉ số quan trọng dùng để đánh giá hiệu quả của các mô hình phân loại trong học máy.

**** Precision*** (độ chuẩn xác): cho biết trong số tất cả các dự đoán dương tính (positive) của mô hình, có bao nhiêu phần trăm là đúng. Công thức:

$$\textbf{\textit{Precision}} = \textbf{\textit{TP / (TP + FP)}} \quad (3.16)$$

Trong đó:

- **TP** là số lượng dự đoán đúng dương tính (True Positive)
- **FP** là số lượng dự đoán sai dương tính (False Positive).

Precision cao nghĩa là mô hình ít đưa ra dự đoán sai khi nhận diện lớp dương.

**** Recall*** (độ bao phủ hay độ nhạy): đo lường khả năng phát hiện hết các trường hợp dương tính thực sự trong dữ liệu. Công thức:

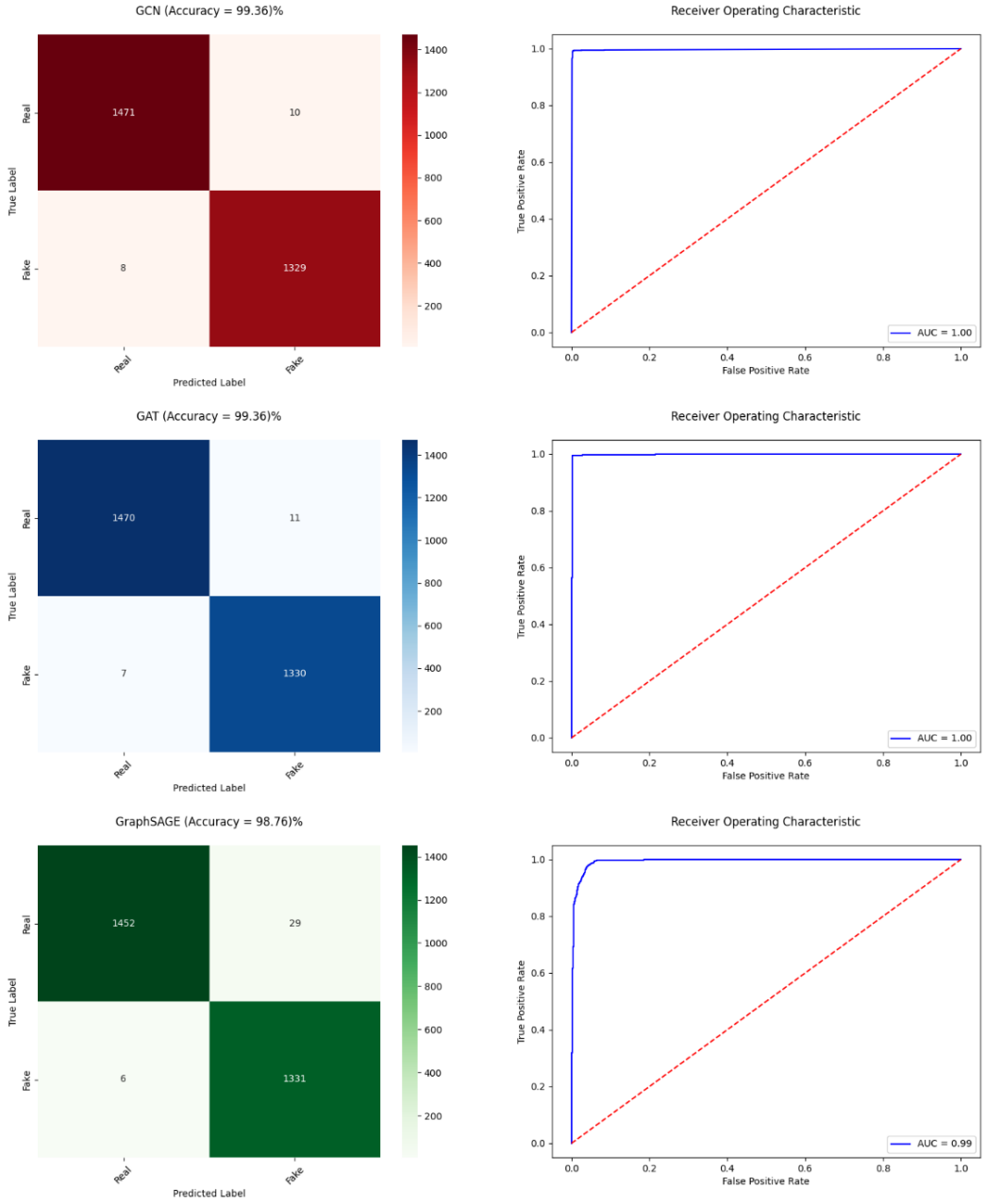
$$\textbf{\textit{Recall}} = \textbf{\textit{TP / (TP + FN)}} \quad (3.17)$$

với **FN** là số lượng bỏ sót (False Negative), tức các trường hợp dương tính thật nhưng mô hình không phát hiện được. Recall cao đồng nghĩa với việc mô hình ít bỏ sót các trường hợp cần phát hiện.

**** F1-score*:** là trung bình điều hòa giữa Precision và Recall, giúp cân bằng giữa hai chỉ số này, đặc biệt hữu ích khi không thể đánh đổi hoàn toàn độ chính xác để lấy độ bao phủ (hoặc ngược lại). Công thức:

$$\textbf{\textit{F1-score}} = 2 \times \textbf{\textit{(Precision}} \times \textbf{\textit{Recall) / (Precision + Recall)}} \quad (3.18)$$

F1-score càng gần 1 càng tốt, và thường được sử dụng khi dữ liệu không cân bằng (ví dụ: phân loại email spam, phát hiện bệnh hiếm gặp…). Nhìn chung, ba chỉ số này bổ sung cho nhau và cung cấp cái nhìn toàn diện về hiệu quả của mô hình phân loại.

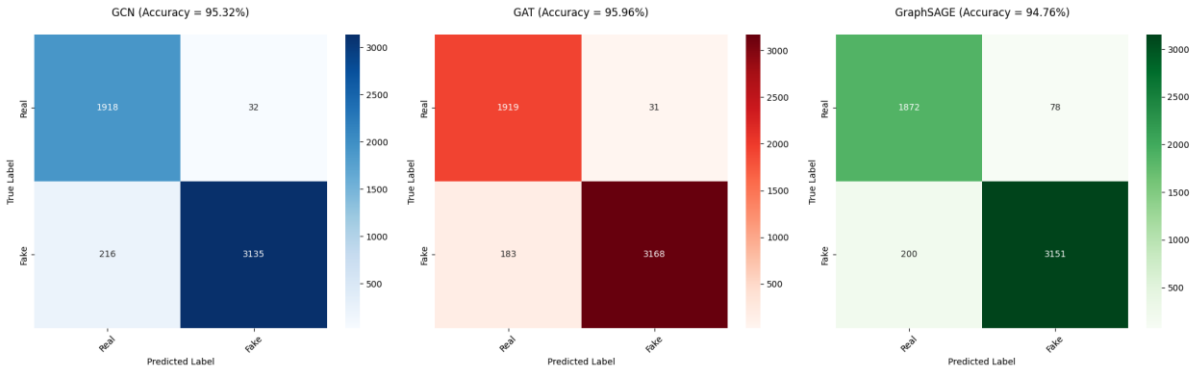


Hình 3.13: Kết quả kiểm tra trên toàn bộ tập dữ liệu huấn luyện

Model	Label	Precision	Recall	F1-score
GCN	Real	~ 0.99	~ 0.99	~ 0.99
	Fake	~ 0.99	~ 0.99	~ 0.99
GAT	Real	~ 1.00	~ 0.99	~ 0.99
	Fake	~ 0.99	~ 0.99	~ 0.99
GraphSAGE	Real	~ 1.00	~ 0.98	~ 0.99
	Fake	~ 0.98	~ 1.00	~ 0.99

Bảng 3.2: Chỉ số đánh giá các mô hình GNN

**** Thử nghiệm các mô hình trên toàn bộ tập dữ liệu Cresci-2015***



Hình 3.14: Kết quả thử nghiệm trên toàn bộ dữ liệu Cresci-2015

Ở giai đoạn huấn luyện (hình 3.13), cả ba mô hình đều đạt được kết quả ấn tượng. Mô hình GCN và GAT đạt độ chính xác (accuracy) ngang nhau ở mức 99,36%, còn GraphSAGE đạt 98,76%. Con số này gần như tiệm cận mức hoàn hảo trong học máy, đặc biệt khi kết hợp với giá trị AUC của đường ROC đều đạt gần như tuyệt đối (1.00 cho GCN và GAT, 0.99 cho GraphSAGE). Khi nhìn vào ma trận nhầm lẫn (confusion matrix), số lượng dự đoán sai là cực kỳ nhỏ. GCN và GAT chỉ dự đoán sai 18 mẫu trên tổng 2818 mẫu, GraphSAGE dù kém hơn một chút nhưng vẫn chỉ sai khoảng 35 mẫu. Những con số này cho thấy mô hình đã học rất tốt các đặc trưng của dữ liệu, nắm bắt rõ ràng mối quan hệ và đặc điểm của hai lớp "Real" và "Fake". Trong môi trường huấn luyện, cả ba mô hình đều chứng tỏ khả năng phân tách mạnh mẽ, thể hiện qua việc các điểm dữ liệu được xếp tách biệt gần như tuyệt đối trên mặt phẳng đặc trưng mà mô hình đã học.

Tuy nhiên, khi chuyển sang giai đoạn dự đoán trên toàn bộ tập dữ liệu Cresci-2015 (hình 3.14) đã có sự khác biệt rõ rệt. Độ chính xác của cả ba mô hình đều giảm đáng kể. GCN giảm từ 99,36% xuống còn 95,32%; GraphSAGE giảm từ 98,76% xuống 94,76%; GAT giữ được mức giảm ít nhất, từ 99,36% xuống 95,96%. Sự suy giảm này tuy vẫn để lại độ chính xác cao so với nhiều phương pháp học máy truyền thống khác, nhưng trong bối cảnh so sánh với kết quả huấn luyện, đây là một dấu hiệu cho thấy khả năng tổng quát hóa (generalization) của các mô hình chưa đạt mức lý tưởng. Khi phân tích sâu hơn vào ma trận nhầm lẫn ở giai đoạn này, chúng ta thấy rõ vấn đề, số lượng dự đoán nhầm "Fake" thành

“Real” tăng mạnh. GCN có tới 216 dự đoán nhầm “Fake” thành “Real”, tương tự GraphSAGE và GAT lần lượt có 200 và 183 dự đoán sai dạng này. Những sai số này phản ánh rằng mô hình gặp khó khăn hơn nhiều khi xử lý dữ liệu thực tế, nơi mà cấu trúc mạng và đặc trưng không còn sự phân biệt rõ ràng như trong dữ liệu huấn luyện.

Nguyên nhân chính của sự suy giảm hiệu suất nằm ở đặc trưng mối quan hệ của dữ liệu huấn luyện. Trong 2 tập dữ liệu con được chọn để huấn luyện, các mối liên kết phần lớn là Real - Real hoặc Fake - Fake, tức là đa số các tài khoản cùng loại liên kết với nhau. Trong khi đó mối liên kết giữa 2 tài khoản khác loại Real - Fake lại rất hạn chế. Điều này vô tình tạo ra một cấu trúc mạng xã hội lý tưởng và đồng nhất, nơi mô hình dễ dàng học được quy luật rằng “nếu hàng xóm là Real thì khả năng cao nút này cũng là Real” và ngược lại. Tuy nhiên, trong toàn bộ tập dữ liệu Cresci 2015, ngoài các liên kết cùng lớp, còn tồn tại khá nhiều liên kết Real - Fake. Khi gặp những trường hợp này, các nút bị “nhiễu” bởi hàng xóm khác lớp dẫn đến kết quả là tỷ lệ dự đoán sai tăng lên. Hiện tượng này là một ví dụ điển hình của distribution shift khi phân phối dữ liệu mà mô hình gặp phải trong quá trình triển khai thực tế có sự khác biệt so với phân phối dữ liệu mà nó được huấn luyện**.** Đi sâu hơn vào từng mô hình, chúng ta sẽ thấy mức độ ảnh hưởng khác nhau:

- **GCN** hoạt động dựa trên việc tổng hợp thông tin từ các nút láng giềng thông qua phép tích chập trên đồ thị. Khi dữ liệu huấn luyện có số lượng kết nối đồng lớp lớn, phép tổng hợp này hoạt động rất hiệu quả. Nhưng khi gặp nhiều kết nối chéo lớp (Real - Fake), thông tin từ hàng xóm khác lớp sẽ bị trộn vào và làm sai lệch biểu diễn đặc trưng, khiến dự đoán kém chính xác hơn.

- **GAT** khắc phục một phần hạn chế của GCN nhờ cơ chế attention, cho phép gán trọng số khác nhau cho từng láng giềng. Trong trường hợp có kết nối Real - Fake, GAT có khả năng giảm trọng số cho các hàng xóm “nhiễu”, từ đó giữ lại được phần lớn thông tin quan trọng và duy trì hiệu suất tốt hơn. Đây cũng là lý do GAT giảm ít nhất khi áp dụng trên toàn bộ dữ liệu.

- **GraphSAGE** sử dụng chiến lược lấy mẫu láng giềng và kết hợp đặc trưng để tạo ra biểu diễn nút. Tuy nhiên, phương pháp này có thể chưa đủ mạnh trong việc lọc bỏ nhiễu từ các kết nối chéo lớp. Khi có nhiều hàng xóm khác lớp, biểu diễn nút dễ bị pha trộn, khiến mô hình có nhiều dự đoán sai.

Tóm lại, ưu điểm nổi bật của các mô hình GCN, GAT và GraphSAGE đã được huấn luyện là khả năng khai thác cấu trúc mạng xã hội để phân loại nút với độ chính xác rất cao trong điều kiện dữ liệu huấn luyện đồng nhất. Tuy nhiên, chính sự "lý tưởng hóa" của dữ liệu huấn luyện lại trở thành hạn chế khi áp dụng mô hình vào môi trường thực tế phức tạp hơn, nơi các kết nối chéo lớp tồn tại phổ biến. Trong ba mô hình, GAT tỏ ra ổn định hơn nhờ cơ chế attention tuy nhiên nó đòi hỏi thời gian và chi phí tính toán cao hơn hai mô hình còn lại đặc biệt là khi dữ liệu mở rộng, số đỉnh trong đồ thị tăng lên đáng kể. GCN bị ảnh hưởng nhiều hơn khi có kết nối nhiễu và GraphSAGE chịu tác động mạnh nhất với tỷ lệ bỏ sót cao. Để cải tiến chất lượng các mô hình, cần mở rộng dữ liệu huấn luyện, tăng tỷ lệ mối liên kết Real - Fake để đảm bảo tính đa dạng của các mẫu kết nối. Bên cạnh đó, giám sát hiệu năng mô hình sau triển khai và cập nhật định kỳ với dữ liệu mới sẽ là yếu tố then chốt giúp duy trì độ chính xác lâu dài.

KẾT LUẬN

Sau thời gian tập trung nghiên cứu và nhờ sự giúp đỡ, chỉ bảo tận tình của các thầy, cô giáo hướng dẫn, đồ án đã hoàn thành mục tiêu đề ra là khảo sát, phân tích và đánh giá hiệu quả các giải pháp mạng nơ-ron đồ thị trong bài toán phân loại tài khoản mạng xã hội. Kết quả thực nghiệm trên bộ dữ liệu Cresci-2015 cho thấy các mô hình GNN có khả năng khai thác đồng thời thông tin thuộc tính và cấu trúc kết nối của tài khoản, từ đó mang lại hiệu suất vượt trội so với nhiều mô hình học máy truyền thống. Điều này khẳng định tiềm năng ứng dụng GNN trong việc phát triển các hệ thống giám sát, phát hiện tự động tài khoản giả mạo, góp phần nâng cao năng lực bảo vệ an toàn thông tin và phòng chống các hành vi gian lận, thao túng trên mạng xã hội.

Bên cạnh những kết quả đạt được, đồ án vẫn còn tồn tại những hạn chế nhất định như phạm vi dữ liệu còn bó hẹp, chưa triển khai trên môi trường giám sát thời gian thực và chưa tối ưu cho các đồ thị quy mô siêu lớn. Để kế thừa và phát huy những thành quả đã đạt được, trong thời gian tới tôi sẽ tiếp tục mở rộng và hoàn thiện nội dung nghiên cứu theo các hướng sau:

- Mở rộng thử nghiệm với nhiều bộ dữ liệu đa dạng hơn, bao gồm dữ liệu mới cập nhật và dữ liệu từ các nền tảng mạng xã hội khác nhau.

- Nghiên cứu các kiến trúc GNN tiên tiến hơn như GIN, R-GCN hay GNN cho đồ thị động, nhằm xử lý dữ liệu biến đổi theo thời gian.

- Tích hợp thêm cơ chế giải thích mô hình (Explainable AI) để tăng khả năng minh bạch và hỗ trợ phân tích kết quả.

- Xây dựng hệ thống giám sát trực tuyến, tối ưu hóa khả năng mở rộng, đáp ứng yêu cầu xử lý dữ liệu mạng xã hội thực tế với cộng đồng người dùng lớn, các mối quan hệ phức tạp hơn.

Với tiềm năng và kết quả bước đầu đạt được, nghiên cứu này sẽ là nền tảng mở ra nhiều cơ hội giúp tôi tiếp tục phát triển trong tương lai trong lĩnh vực an ninh mạng và phân tích dữ liệu.

TÀI LIỆU THAM KHẢO

[1] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 31 July-4 Aug. 2005 2005, vol. 2, pp. 729-734 vol. 2, doi: 10.1109/IJCNN.2005.1555942.

[2] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans Neural Netw,* vol. 20, no. 1, pp. 61-80, Jan 2009, doi: 10.1109/TNN.2008.2005605.

[3] T. Kipf, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907,* 2016.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903,* 2017.

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems,* vol. 30, 2017.

[6] S. Feng, H. Wan, N. Wang, and M. Luo, "BotRGCN: Twitter bot detection with relational graph convolutional networks," in *Proceedings of the 2021 IEEE/ACM international conference on advances in social networks analysis and mining*, 2021, pp. 236-239.

[7] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Fame for sale: Efficient detection of fake Twitter followers," *Decision Support Systems,* vol. 80, pp. 56-71, 2015.

[8] L. Wu, P. Cui, J. Pei, L. Zhao, and X. Guo, "Graph neural networks: foundation, frontiers and applications," in *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 2022, pp. 4840-4841.

[9] Đinh Mạnh Tường, "Trí tuệ nhân tạo: Cách tiếp cận hiện đại," Nhà xuất bản Khoa học và Kỹ thuật, 2024