Multirelational Twitter Bot Detection using Graph Neural Networks

1st Ketan Jadhav*, 3rd Chris Pollett†, and 4th Katerina Potika‡
Department of Computer Science
San Jose State University, San Jose, CA, USA
*ketan.jadhav@sjsu.edu, †chris.pollett@sjsu.edu, ‡katerina.potika@sjsu.edu

2nd Petros Potikas
Electrical and Computer Engineering
National Technical University of Athens, Athens, Greece
ppotik@cs.ntua.gr

***Abstract*—Social media is an important means of modern communication and information exchange. Ease of access and global reach are the primary factors contributing to the popularity of several social media platforms. Apart from human (real) users, they also have bots, automated programs developed to facilitate online engagement and interaction. However, these bots are increasingly being exploited for malicious purposes, such as disseminating false information and manipulating public opinion. Detecting and removing bots is essential for maintaining authentic human interactions and trustworthy public opinions.**

This paper proposes the Multirelational Bot Detection Graph Neural Network method (MultiBotGNNs) to detect bots. It treats the problem as a node classification problem in a graph that has two types of nodes (real and bot users) and multiple types of edges, e.g., the various relations (interactions) between users. To detect bots, we use their characteristics processed along the social network of the different user relations, such as follow, follower, and like. We experimented with the TwiBot22 dataset, which is a big dataset of bots on X (formerly Twitter). Our hypothesis is that by considering just the different relations and their characteristics for the prediction, and by ignoring the tweet contents, we can get good results without the extra process time for the tweets. Experimental results confirm our hypothesis.

***Index Terms*—Bot identification, node classification, embedding techniques, Twitter (X), graph neural networks, TwiBot22, misinformation detection, heterogeneous graphs.**

I. Introduction

The rise and evolution of the Internet has changed the way humans communicate and access information. From the early days of mailboxes to the growth of social media platforms like Facebook, X (former Twitter), humans have integrated the internet as a crucial part of their lives. Social Media has connected humans across the globe in ways only imagined in the past. Moreover, it has changed the way humans consume information and create information, such as through automated accounts called bots.

A bot is any automated program that is developed to carry out a fixed functionality. With the increasing use of social media, bots have gained popularity as organizations seek efficient ways to engage with users daily on these platforms. Initial uses of bots included posting relevant pre-determined updates or posts from the organization's accounts. With the growth of available Artificial Intelligence (AI) tools, the usage of bots became more widespread and complex, beyond automating routine tasks.

Although a vast amount of bots are used to post useful content and engage ethically, there are a lot of bots being deployed with malicious intent. Studies [1] have shown that bots are used to spread false information, negatively manipulate users, and cause disruptions to social media. Bots can operate with AI responses to manipulate and misdirect the user through social content or chat. Due to the critical implications of bots and the potential to cause damage, bot detection has become a crucial challenge for social media operations. In recent times, with the widespread use of AI, it has become a high priority to preserve integrity and safety throughout social media.

The exact content generated on Twitter by bots is unclear, but it is estimated to be a high percentage of all users. Platforms like Twitter used to employ tools like Botometer [2] (data until 2023) to help monitor user activity and identify bot accounts. Another example of a tool used in websites to detect bots from humans is the CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). However, the use of only limited features in these tools may not be able to detect bots correctly or in all settings, like in social media. More recent approaches, in addition to the account metadata, include information like the number of followers/following and tweets to calculate real-time scores based on the user ID. Unfortunately, algorithms that can recreate the metadata of real users have been developed, which can provide bot accounts' authenticity to pass through bot detectors. Training Machine Learning (ML) models on such data can lead to poor predictions on real accounts as well. Limited features make this method less accurate, which prompts the use of advanced graph-based methods to detect bots. In Figure 1, you can see the bot score from Botometer for a real user; a score of 2.3/5 indicates its lack of reliability to classify.

The rest of the paper is as follows: in Section II we explore different techniques in Natural Language Processing (NLP) for embedding text, and the task of node classification in

Fig. 1. Bot Score for a real user

graphs. In Section III, we talk about the various existing techniques in detecting social media bots. In Section IV, we discuss more about our dataset. Section V explains the proposed methodology for bot detection along with the pre-processing and the algorithms used. Section VI discusses the results of our method across the determined evaluation metrics. Finally, Section VII concludes the paper.

II. Techniques

In this section, we will discuss the necessary background in detail, along with the various techniques we use.

We will model our problem as a graph $G = (V, E)$ that has a set of users V and their connections, i.e., the set E is the different ways that users interact with each other. Graphs are a powerful and comprehensive model to represent and store information. This information consists of nodes, i.e., user accounts, and edges, indicating how they are connected over the social network. Nodes may contain various useful information that characterizes the node. In this case, the node may have information about the users, like bio, follower count, etc. The edge determines how the nodes are connected and may have different types of edges, i.e., relations. In our case, the edge types are follower, following, like, etc. Therefore, our graph can be a heterogeneous graph with different types of edges.

Bot detection is reduced to the node classification problem. It is a graph problem where the goal is to classify the nodes into some given categories by leveraging the structure of the graph and/or the node information. Many real-world problems can be solved as node classification on a graph. This information can be useful for solving problems and are a significant addition to using numeric data about problem properties to classify the graphs.

Some well-known word embedding techniques involve distributed representation methods like Word2Vec [3], GloVe [4], etc. These methods represent text as large, dense vectors based on weights or features obtained from a large corpus of words generated through training the feature extraction methods on a very large text. Bidirectional Encoder Representations from Transformers (BERT) [5] is a strong NLP model that is based on pre-training weights using a transformer architecture. Lastly, a Robustly Optimized BERT Approach (RoBERTa) [6] is an extension of BERT used to generate feature embeddings for NLP, and this is the model we use. RoBERTa builds upon the pretraining mechanism used by BERT and fine-tunes it to make it better by introducing a dynamic masking approach.

Graph Neural Networks (GNNs) have gained popularity because of their capability to analyze graph-structured data and capture intricate relationships by leveraging both node features as well as graph structure. GNNs leverage the entire graph structure as a whole by utilizing information from nodes as well as the edges connecting the nodes.

As discussed previously, the nodes in a graph have useful information, and so do the edges. For GNNs to learn the graph structure and analyze relationships, a message-passing mechanism is used. In this mechanism, all the information in a particular node is shared with its neighboring nodes. After passing the information, all the collective information received at a node is aggregated with the existing information, and in this way, neighboring nodes learn about each other and keep useful information. Through each iteration, more information is gained over larger distances in a graph, which allows the GNN to learn about the graph structure, node information, and understand complex relationships. In this case, user information is propagated throughout the graph structure along with the edge type (type of relation). Various methods of GNNs will be used and described later.

III. Related Work

Several studies have conducted research in the past regarding bot identification on Twitter (X), either as a node classification [7] or a graph classification problem [8]–[11], as well as on other social media platforms. Researchers were able to exploit the features obtained from the user's tweet contents through the use of several NLP techniques and improve the detection process.

BotFinder [12] uses the node data from traffic nodes and applies traditional ML algorithms to classify the graphs. This study has applications in network flow that can be used in bot identification as well. MRLBot [13] uses Deep Neural Networks (DL) for Twitter bot classification on the Twibot-22 [1] dataset. The study implements a transformer architecture and employs a CNN encoder-decoder feature to classify embeddings. This model incorporates not only the structural characteristics of node neighborhoods but also the internal organization of communities and community linkages.

Wei et al. [14] focus on proposing a Bi-directional LSTM architecture to process user tweets to understand and retain context of the tweets used to classify accounts that require no prior knowledge or assumption about users' profiles, networks, or historical behavior on the target account. Implementing Recurrent Neural Networks to use word embeddings, this method is an indication that tweet contents and their context are important features for classification.

Kudugunta et. al [15] propose a hybrid method including tweet content and metadata at the tweet level for bot identification using a contextual LSTM. This method also proposes and uses a technique based on synthetic minority oversampling to improve the dataset. This approach achieves good results on an oversampled dataset.

Alhosseini et. al. [16] conduct an extensive study on spam bots on the Twitter platform and employ representation learning for spam bot detection. The study leverages GCNNs, which use both the features of a node and aggregate the features of a node's neighborhood for classification. Results show that

this approach outperforms the state-of-the-art classification algorithms.

SATAR [17] in their novel approach for bot detection, use a self-supervised representational learning framework. SATAR leverages the semantics, property, and neighborhood information of specific users for classification and pre-trains on the massive number of users and later fine-tunes the model on specific cases. This study uses the tweet semantic subnetwork, profile property subnetwork, and follower-following subnetwork to gain information from data and later uses a co-influence aggregator to combine these features. An older dataset, Twibot-20 [18], is used.

Cresci et. al. [19] proposed a novel approach that was inspired by DNA sequencing to identify bot accounts from Twitter, which extracts online user behavior patterns. Based on the digital DNA obtained from the user behaviour and activity, standard DNA analysis techniques help differentiate between real users and bots. This system has proved to be simple yet effective for classification purposes, which has a wide range of applications.

BotRGT is proposed in [20], which uses relational graph transformers (RGT) for graph classification. The study constructs a heterogeneous information network with users as nodes and diversified relations as edges. RGTs are used to model heterogeneous influence between users to better learn node representations. Semantic attention networks are used to assign relative importance to relations and aggregate messages across users and relations, similar to GNNs. Again, the Twibot-20 dataset is used.

BotRGCN architecture proposed in [21] uses a relational graph convolutional network to address the community challenge by constructing a heterogeneous graph from user relationships. It uses both multimodal semantic and property information as feature sets for the RGCN architecture. Results show that it outperforms most approaches on benchmark datasets. The BotRGCN model uses the following data for classification: profile data, user tweets, and relation graphs. A Relational GCN model is used along with several layers of Sequential Linear layers to process each of the tensors. The resultant aggregation, along with the user graph, is used as input to the RGCN model.

IV. Dataset

We have used the Twibot-22 [1] dataset for our experiments. Twibot-22 is the most comprehensive dataset available on Twitter bot classification to date. Table I demonstrates the statistics about the latest Twibot-22 dataset used in our work and the older Twibot-20 dataset.

Attribute	Twibot-20	Twibot-22
Human	5237	860,057
Bot	6589	139,943
Tweet	33,488,192	86,764,167
Edge	33,716,171	170,185,937

TABLE I
Comparison of Twibot-20 and Twibot-22 datasets

The Twibot-20 has only three relation types: “follower", “following", and “post". We focus on the Twibot-22 dataset, which has 14 types of relations (see a few Table II). The Twibot-22 dataset has multiple files that we can use to extract features for our model. The “edge.csv” contains all edges and the relation type identified by “source id” and “target id”. The “user.json” has all the profile information, which is split into numerical and categorical data. We also obtain the user bio and use NLP to extract features. Further, “tweet.json" has the tweet content, which can be processed, but we will not use that.

Relation	Source	Target	Description
following	user	user	user A follows user B
follower	user	user	user A is followed by user B
post	user	tweet	user A posts tweet C
pin	user	tweet	user A pins tweet C
like	user	tweet	user A likes tweet C
mention	tweet	user	tweet C mentions user B
retweet	tweet	tweet	tweet C retweets tweet D
quote	tweet	tweet	tweet C quotes tweet D with comments
reply	tweet	tweet	tweet C replies to tweet D
own	user	list	user A is the creator of list L
member	user	list	user A is a member of list L
follow	user	list	user A follows list L
contain	list	tweet	list L contains tweet C
discuss	tweet	hashtag	tweet C discussed hastag H

TABLE II
Twibot-22 Relation Types

Table II demonstrates the structuring of user relation types in the edges. Multiple types of relations exist, including homogenous edges (user-user, tweet-tweet) and heterogenous edges (user-tweet, user-list). Some GNNs work only on homogeneous relationships, while some have the ability to handle heterogeneous relations. Within homogeneous, some models can handle multiple types of edges (like “follower", “following" or “retweet", “quote") and the proposed method explores this approach. Two or more heterogeneous relations can be processed together as homogeneous relations, and our method explores that as well. We are utilizing the “follower", “following", “post", and “like" relations in our models. While some relations are used directly, we work on some relations to process and obtain new relations to be used for our models.

Relation	Edges
following	$2,626,979$
follower	$1,116,655$
post	$40,887,365$
like	$595,794$

TABLE III
Twibot-22 Edge Count

Other than the edges file that contains relations between users, we have a few more files in the dataset that will be used for classification and are described in detail next:

- “edge.csv" Contains information about the edges in three columns, “source id” (indicating the source user node id), relation (type of relation), and “target id"(indicating the target user node id).

- "user.json": Contains all the information about the user, like the time of account creation, number of "followers", "following", "tweet count", whether the account is verified, the profile picture, etc. This file contains information for $1,000,000$ users. It contains the following columns: created at, description, entities, id, location, name, pinned tweet id, profile image URL, is protected, URL, username, is verified, and public metrics.
- "labels": Contains all the users' (human and bot) ids along with their labels.
- "tweets": There are 9 tweet files in the dataset, numbered from tweet0 to tweet8, containing all the information about the user tweets. The files contain information like tweet id, attachment, tweet content, hashtag, etc.

The public metrics further contain the following information: followers count, following count, tweet count, and list count.

V. Methodology

Identifying social media accounts as bots or humans will be solved as a node classification problem. Traditional ML methods involve using only the user profile data. Advanced methods include the use of various GNN models. In our approach, called Multirelational Bot Detection Graph Neural Network method (MultiBotGNNs), we leverage the graph structure formed by various user relations like "following"/"follower" networks and GNN models. Figure 2 demonstrates the flowchart for the process.

MultiBotGNNs combines the additional relations that the Twibot-22 dataset offers to discover complex relations between nodes (users) that can help in the prediction of bots. Algorithm 1 describes the three steps of our MultiBotGNNs. The input/output of **Input:** Feature Sets, Edge Index, Edge Weights/Edge Types (optional) **Output:** Node classification (Real/Bot user).

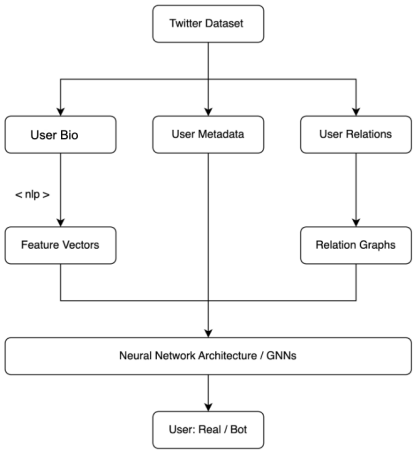

Fig. 2. Process Flow

The dataset contains data about users' profiles, tweet content, and user relations (edge.csv). This data will be processed to obtain the feature sets required for our GNN models.

Algorithm 1 MultiBotGNNs for Bot Detection

procedure BotDetection
Step 1: Preprocessing
Process large data files using Dataproc and PySpark on Google Cloud Platform
Extract numerical and categorical properties from user information
Extract textual descriptions from user node biographies
Process edge data to extract relations and generate edge indices per relation
Generate combined edge indices and edge types for multi-relational data
Create new synthetic relation types from heterogeneous relations
Step 2: GNN Model Training
Train the following GNN models on the processed data:
GCN: using individual relations
GraphSAGE: using individual and combined relations
GAT: using individual and weighted relations
RGCN: using multiple relation types
Step 3: Testing and Hyperparameter Tuning
Test each model on various types of generated datasets
Perform hyperparameter tuning
Evaluate the performance metrics of all models

The proposed methodology explores three major Graph Neural Network Models: GraphSAGE [22], Graph Attention Networks (GAT) [23], and Graph Convolutional Networks (GCN) [24]. GCNs can further be classified into two approaches: Single GCNs and Relational GCNs (RGCNs) [25], which can process multiple edge types. Each of these requires an input feature vector matrix X, and an edge index vector which contains edges between users marked by indexes. Additionally, to leverage RGCNs, which can process multiple types of relations, we have the edge index vector containing multiple types of edges, and we additionally require an edge type vector indicating the type of edge. Since GATs work on an attention mechanism to determine important connections, it also requires an edge weight parameter indicating the weight of the edge. Further, GraphSAGE can also process weighted graphs. The proposed method implements GraphSAGE using edge weights as well.

The following tensors are used for our GNN models as input features:

- Description Tensor
- Numerical Properties Tensor
- Categorical Properties Tensor
- Edge Index and Edge Type Tensors

A. Preprocessing

The GNN models require the feature vectors as input, along with the relation graphs in the form of an adjacency matrix A. The proposed method generates the input vector by

concatenating (a) numerical properties of the user profile, (b) categorical properties of the user profile, and (c) description or bio of the user processed using RoBERTa. The relations from the edges data are used to create adjacency matrices of individual relations (“follower", “following"), generate new relations from combining existing ones, and merge multiple relations.

The initial Twibot-22 dataset obtained had details of $1,000,000$ users. However, upon inspection, there was a huge bias in the data, with around 86% of the dataset being those of human users. Figure 3 shows the bias in the dataset.

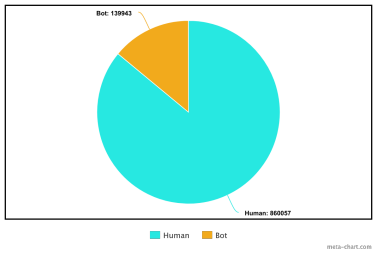

Fig. 3. Distribution of Humans and Bots in the dataset

The first step of the pre-processing is to balance the dataset. Given that the original dataset was too large to process with restricted computation resources, balancing reduced the size of the dataset to $279,886$ nodes, which made it easier to process and reduced the bias in the data. Sampling the dataset for balancing also led to sampling of the edges data to include only those edges that connected the users that we obtained after initial preprocessing.

For the numerical properties, we build a tensor from the following data:

- followers count
- active days (number of days active on Twitter)
- length of the user’s screen name
- following count
- statistics derived from the number of tweets posted by the user.

For the categorical properties, we use the following data:

- is the user account protected
- is the user account verified
- if the user has a default profile image or a custom one.

For the description property, we generate a tensor by processing the user bio from their profile using a RoBERTa model with a max length of 50. Similarly, we concatenate each tweet posted by the user and apply the same model to generate a tweet tensor.

We have the “edge.csv" file that contains the source node, the target node, and the relation type. This data, too, is too large to process locally. We use GCP services for edge data extraction by first uploading it to Cloud Storage. Then, we use the Dataproc service to process the large data. We create a Dataproc cluster and create several Virtual Machine nodes to process the data. We then create a Spark Job to use the Apache Spark big data processing module using PySpark. This Python-based job extracts the data into individual CSV files for each relation. These edge CSV files are processed individually to generate tensors for source and target user ids based on indexes.

When processing multiple relation types together using the RGCN model, we need an additional parameter to indicate the edge type. During the processing for obtaining the edge index, we also generate the edge type tensor in case of multiple relations.

Additionally, while applying the GraphSAGE and GAT models, we provide an optional parameter, edge weight, while processing some relations. We can generate the weight tensors while generating edge indexes.

In our MultiBotGNNs we create different graphs based on the type of edge relations we consider. We can use each relation by itself or combine two as a new one. First, we use “follower” and “following” relations, which are available in the dataset. Along with these two, we create a new relation, which we name “interaction” by combining the existing relations “post” and “like” that we have in the dataset.

The “post” relation is an edge between “userid” and “tweetid” indicating the user that posted the tweet. Similarly, the “like” relation is an edge between “userid” and “tweetid," indicating the user that liked the tweet. We join these two relations on the “inner-join” on “userid” to obtain the “interaction" relation. The resultant we obtain is a user-user relationship where “source id" indicates the user that likes the tweets, and the “target id" indicates the user whose tweets have been liked. We obtain the relation of which user interacts with which user using this technique. Additionally, if user A likes some user B’s multiple tweets, instead of having multiple entries, we have a new column called ’weight’ in our generated data. This indicates the strength of the relationship between two users. If a user likes multiple tweets of some other user, it indicates that the relationship between those two for interaction is stronger.

B. Model Description

Regularization Regularization is used to prevent overfitting in deep learning models. Overfitting is when a model learns the model too rigidly and, in turn, might end up learning not just the actual underlying patterns, but the noise too. A result of overfitting is that the model learns the training data too well but performs poorly on the testing data. Dropout is a regularization technique commonly used to prevent overfitting in neural networks. A dropout rate of 0.1 indicates that during training, 10 percent of the neurons in the network will be randomly set to zero, forcing the network to learn more robust features. It prevents specific neurons from being overly reliant on each other.

The embedding size parameter determines the dimensions of the embedding space for categorical variables or words. Each categorical variable or word will be represented as a vector of length 32.

The learning rate determines the size of the steps taken during the optimization process. A value of 1e-2 (0.01) is moderate and

is often chosen as a starting point. Higher learning rates may lead to faster convergence but risk overshooting the optimal solution, while lower rates may lead to slower convergence but potentially better fine-tuning.

Weight decay, also known as L2 regularization, is a regularization term added to the loss function to penalize large weights in the model. A weight decay of 5e-2 (0.05) indicates a moderate regularization strength. Higher values of weight decay increase the penalty for large weights, helping to prevent overfitting, but may also overly constrain the model's learning capacity.

The choice of loss function determines the objective that the model aims to minimize during training. The Cross-Entropy Loss is used in classification tasks. It measures the dissimilarity between the predicted probability distribution and the true distribution of class labels.

The optimizer determines the algorithm used to update the model parameters during training in order to minimize the loss function. Adam (Adaptive Moment Estimation) is a popular optimizer that combines ideas from RMSProp and Momentum. It adapts the learning rate for each parameter, allowing for faster convergence and better performance in many cases.

Each tensor is handled first by a linear layer with a ReLU activation function. The result of all the tensors is then concatenated. This forms our input feature for the GNN model.

The following hyperparameters are used to train our models:

- embedding size = 32
- dropout = 0.1
- learning rate = 1e-2
- weight decay = 5e-2
- loss = Cross Entropy Loss
- optimizer = Adam

PyTorch Geometric library has the GNN models that will be used. Each of the models was designed in several layers, taking in the different input feature vectors and processing them before merging them to generate the input feature vector for our Graph Neural Network model.

VI. Experiments and Analysis

A. Loss Function and Evaluation Metrics

Loss quantifies the difference between the predicted values and the actual values during the training of a machine learning model to improve the prediction. It serves for the backpropagation and learning a good model of the patterns in the data. Lower loss values indicate better alignment between predictions and actual values, reflecting improved model performance.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \quad (1)$$

where,

- N = number of samples in the dataset
- y_i = True label for the i-th sample
- $\hat{y}_i$ = Predicted probability for the i-th sample

Accuracy measures the proportion of correctly classified instances among all instances in the testing dataset, demonstrating the overall correctness of the model predictions in unseen data.

$$\text{Accuracy} = \frac{\text{Number of correctly classified instances}}{\text{Total number of instances}}$$

Precision evaluates the ability of a model to correctly identify positive cases out of all positive classified instances. It is the ratio of true positive predictions to the total number of positive predictions made by the model. Precision is particularly relevant in scenarios where minimizing false positives is critical.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

Recall, also known as sensitivity or true positive rate, measures the ability of a model to identify all relevant instances in a dataset. It is the ratio of true positive predictions to the total number of actual positive instances. Recall is crucial in applications where missing positive instances carries significant consequences.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

The F1 Score is a harmonic mean of precision and recall, providing a balanced measure of a model's performance. It addresses the trade-off between precision and recall, making it suitable for tasks where both false positives and false negatives need to be minimized. The F1 Score ranges from 0 to 1, with higher values indicating better model performance.

$$\text{F1 Score} = 2 \text{ x } \frac{\text{Precision x Recall}}{\text{Precision + Recall}}$$

B. Results

MultiBotGNNs involves processing the "follower", "following", and "interaction" relations individually and training the data using Single GCN, GraphSAGE, and GAT.

The next step involves combining "follower" and "following" and then training the data using the RGCN model in these two relations. Finally, all three relations are combined and processed using the RGCN model. To train GraphSAGE and GAT on the interactions data, which have weights as well, we have used an additional edge weight parameter to train those models. All the models were trained for 50 epochs with the parameters discussed earlier.

For the Twibot-22 dataset, both "follower" and "following" relations, along with the newly constructed "interaction" relation, were processed individually first, and the models GCN, GAT, and GraphSAGE were used. Combining both relations, we then used the RGCN model. The description, categorical properties, and numerical properties will stay common to all the types of relations. The edge indexes and edge types vary depending on the relation.

Evaluation metrics used for analysis are the accuracy with which the model can identify bot accounts from the data,

relation(s)	Accuracy	Precision	Recall	F1 Score
follower (fol)	0.6342	0.6342	0.4567	0.5546
following (foli)	0.6035	0.7017	0.3567	0.4729
interactions (like)	**0.7213**	**0.7289**	**0.7023**	**0.7154**

TABLE IV
GCN MODEL RESULTS FOR DIFFERENT RELATIONS PROCESSED INDIVIDUALLY

relation(s)	Accuracy	Precision	Recall	F1 Score
fol + foli	0.7293	0.7227	0.7419	0.7322
fol + foli + like	**0.7352**	**0.7308**	**0.7426**	**0.7367**

TABLE V
RGCN MODEL RESULTS FOR DIFFERENT RELATION COMBINATIONS

the precision with which it can identify bots, the recall for classification, and the F1 score of the model.

For the GCN model, individual relations were processed, and their edge index tensors were used to train the model along with the common features we have. The results of Accuracy, Precision, Recall, and F1 Score were observed in Table IV.

It can be observed from Table IV that the interactions (like) relation has a better accuracy, precision, recall, and F1-score compared to others.

For the RGCN model, we process multiple relation types together. Experimentation was done on two combinations of the relations for the RGCN model, keeping the common tensors the same. The edges were concatenated and processed to obtain the edge index and edge-type tensors. The results of Accuracy, Precision, Recall, and F1 Score were observed in Table V.

Table V demonstrates the RGCN model on combinations of relations. It can be observed that a combination of all relations used performs better compared to just follower-following in all aspects.

For the GraphSAGE model, relations were processed individually as well, and combinations were experimented with to evaluate the model. The results of Accuracy, Precision, Recall, and F1-score were observed in Table VI.

Table VI demonstrates the GraphSAGE model results for all relations, and it can be observed that this model has varying results for all parameters. While accuracy is the highest in a combination of relations, other relations outperform in other metrics.

relation(s)	Accuracy	Precision	Recall	F1 Score
fol (fol)	0.6739	0.6336	**0.8206**	**0.7151**
foli (foli)	0.6883	**0.6695**	0.7405	0.7032
like	0.6877	0.6682	0.7423	0.7033
fol & foli	0.6773	0.6445	0.7874	0.7088
fol & foli & like	**0.6884**	0.6640	0.7598	0.7087

TABLE VI
GRAPHSAGE MODEL RESULTS FOR DIFFERENT RELATION COMBINATIONS

relation(s)	Accuracy	Precision	Recall	F1 Score
follower (fol)	0.6518	0.7059	0.5174	0.5971
following (foli)	0.7060	0.6825	0.7674	0.7225
interactions (like)	**0.7383**	**0.7195**	**0.7790**	**0.7481**
fol & foli	0.7095	0.6853	0.7720	0.7261
fol & foli & like	0.7121	0.6744	0.8171	0.7390

TABLE VII
GAT MODEL RESULTS FOR DIFFERENT RELATION COMBINATIONS

relation(s)	GAT	GraphSage	RGCN	GCN
follower (fol)	0.6632	0.6667		0.6683
following (foli)	0.6289	0.6400		0.6778
interactions (like)	0.5879	0.6264		0.5976
fol & foli	0.6490	0.6227	0.6219	
fol & foli & like	0.6468	0.6391	0.6184	

TABLE VIII
LOSS OF VARIOUS MODELS FOR DIFFERENT RELATION COMBINATIONS

Similar to the GraphSAGE model, the GAT model also had relations processed individually, along with combinations as observed in Table VII.

The GAT model processed all relations and combinations and it can be observed that interactions relation processed individually performs the best, even outperforming combinations of relations which might indicate that the GAT is better at handling individual relations.

Another parameter we identified to judge our model training is the loss metric, see Table VIII. We observed the loss of each of our models across the 50 epochs and plotted graphs for all models for the Twibot-22 dataset for the final combined relation.

VII. CONCLUSION

The growth of social media has gotten people across the globe connected to each other and also has become the source of reliable information for many. With the growth of computational resources and algorithms to leverage them, a lot of focus has been made on automating various tasks, leading to the generation of bots. Bots often exist as automation products and to share information. However, in recent times, the purpose of bot use has been questionable. A lot of bots these days have been used with malicious intent, which has made regulating bots very important. Bot detection techniques have been developed to identify these bot accounts on social media platforms. Powerful algorithms have been able to mimic human traits and have made the bots harder to detect in recent times.

In the past, information such as user account information has been used to identify bots. With the increase in NLP usage, the textual information on or posted by the account has also been used to help identify bots. In recent times, with the advent of technologies like Graph Neural Networks, we have been able to leverage account relationships with others to potentially identify bots. This project focuses on using user account information, NLP techniques, as well as leveraging several Graph Neural Network algorithms to identify bots on the Twitter (X) social media platform.

We have used the Twibot-22 dataset, which contains account information, tweets posted by accounts, and the relationships that exist between accounts together to design Neural Network architectures for classification. Experimental analysis was conducted on four GNN algorithms. The experiments helped us understand in-depth the advantages of each algorithm and it works to identify bots. Several user relationships, like “follower”, “following”, “post", and “like" were used individually as well as in combinations to construct graphs for classification, which was the focus of the paper.

A comparative analysis of several evaluation metrics leads to the conclusion that Graph Attention Networks, with their powerful attention mechanism, and Relational GCNs, which are adept at processing multiple kinds of relationships, are performing better compared to the rest. A comparison was also made on the baseline models using just account data or just textual data, and it was found that the combination performed better. Further analysis showed that processing combinations of relations and constructing new relations using two relations gave our model better learning and produced better results. GAT model with interaction relation constructed by combining the post and like relation gave the highest accuracy of 73.83%.

We can conclude by affirming that graphs are a strong way to store information and with the abilities of GNNs to process graph based data, we can leverage relations existing across social media between users and entities to gain valuable information about users and their behaviors.

Future work in this domain can include leveraging Heterogeneous GCNs to leverage complex relationships between different entities. These can allow us to learn relations between users and tweets, reposts and help the model potentially perform better at identifying bots. With greater computation power, we can attempt to process multiple relationships as well as multiple types of relationships.

REFERENCES

[1] S. Feng, Z. Tan, H. Wan, N. Wang, Z. Chen, B. Zhang, Q. Zheng, W. Zhang, Z. Lei, S. Yang, X. Feng, Q. Zhang, H. Wang, Y. Liu, Y. Bai, H. Wang, Z. Cai, Y. Wang, L. Zheng, Z. Ma, J. Li, and M. Luo, “Twibot-22: Towards graph-based twitter bot detection,” 2023.

[2] K.-C. Yang, E. Ferrara, and F. Menczer, “Botometer 101: Social bot practicum for computational social scientists,” *Journal of Computational Social Science*, vol. 5, no. 2, pp. 1511–1528, 2022.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[4] J. Pennington, R. Socher, and C. Manning, “GloVe: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (A. Moschitti, B. Pang, and W. Daelemans, eds.), (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018.

[6] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized BERT pretraining approach,” *CoRR*, vol. abs/1907.11692, 2019.

[7] G. Tzoumanekas, M. Chatzianastasis, L. Ilias, G. Kiokes, J. Psarras, and D. Askounis, “A graph neural architecture search approach for identifying bots in social media,” *Frontiers in Artificial Intelligence*, vol. 7, p. 1509179, 2024.

[8] T. Bui and K. Potika, “Twitter bot detection using social network analysis,” in *2022 Fourth International Conference on Transdisciplinary AI (TransAI)*, pp. 87–88, IEEE, 2022.

[9] T. Bui and K. Potika, “Detecting twitter bots with machine learning and propagation graph analysis,” in *2023 Fifth International Conference on Transdisciplinary AI (TransAI)*, pp. 87–90, IEEE, 2023.

[10] T. Bui, A. Pham, W. Kulkarni, Y. Yao, and K. Potika, “Classifying real and bot users based on their news spread for combating misinformation,” in *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 272–279, IEEE, 2023.

[11] K. Gupta and K. Potika, “Fake news analysis and graph classification on a covid-19 twitter dataset,” in *2021 IEEE seventh international conference on big data computing service and applications (BigDataService)*, pp. 60–68, IEEE, 2021.

[12] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, “Botfinder: Finding bots in network traffic without deep packet inspection,” p. 349–360, 2012.

[13] F. Zeng, Y. Sun, and Y. Li, “Mrlbot: Multi-dimensional representation learning for social media bot detection,” *Electronics*, 2023.

[14] F. Wei and U. T. Nguyen, “Twitter bot detection using bidirectional long short-term memory neural networks and word embeddings,” in *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 101–109, 2019.

[15] S. Kudugunta and E. Ferrara, “Deep neural networks for bot detection,” *CoRR*, vol. abs/1802.04289, 2018.

[16] S. Alhosseini, R. Bin Tareaf, P. Najafi, and C. Meinel, “Detect me if you can: Spam bot detection using inductive representation learning,” 2019.

[17] S. Feng, H. Wan, N. Wang, J. Li, and M. Luo, “Satar: A self-supervised approach to twitter account representation learning and its application in bot detection,” 2021.

[18] S. Feng, H. Wan, N. Wang, J. Li, and M. Luo, “Twibot-20: A comprehensive twitter bot detection benchmark,” *CoRR*, vol. abs/2106.13088, 2021.

[19] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, “Dna-inspired online behavioral modeling and its application to spambot detection,” *IEEE Intelligent Systems*, vol. 31, no. 5, pp. 58–64, 2016.

[20] S. Feng, Z. Tan, R. Li, and M. Luo, “Heterogeneity-aware twitter bot detection with relational graph transformers,” 2021.

[21] S. Feng, H. Wan, N. Wang, and M. Luo, “Botrgcn: Twitter bot detection with relational graph convolutional networks,” *Association for Computing Machinery*, 2022.

[22] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *CoRR*, vol. abs/1706.02216, 2017.

[23] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” 2018.

[24] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *CoRR*, vol. abs/1609.02907, 2016.

[25] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, “Modeling relational data with graph convolutional networks,” 2017.