RoSGAS: Adaptive Social Bot Detection with Reinforced Self-supervised GNN Architecture Search

YINGGUANG YANG, University of Science and Technology of China, China and Key Laboratory of Cyberculture Content Cognition and Detection, Ministry of Culture and Tourism, China
RENYU YANG, University of Leeds, United Kingdom
YANGYANG LI, National Engineering Research Center for Risk Perception and Prevention, CAEIT, China; Key Laboratory of Cyberculture Content Cognition and Detection, Ministry of Culture and Tourism, China and Academy of Cyber, China
KAI CUI, University of Science and Technology of China, China, Key Laboratory of Cyberculture Content Cognition and Detection, Ministry of Culture and Tourism, China
ZHIQIN YANG and YUE WANG, Beihang University, China
JIE XU, University of Leeds, United Kingdom, and Beihang University, China
HAIYONG XIE, University of Science and Technology of China, China, Key Laboratory of Cyberculture Content Cognition and Detection, Ministry of Culture and Tourism, China

Social bots are referred to as the automated accounts on social networks that make attempts to behave like humans. While Graph Neural Networks (GNNs) have been massively applied to the field of social bot detection, a huge amount of domain expertise and prior knowledge is heavily engaged in the state-of-the-art approaches to design a dedicated neural network architecture for a specific classification task. Involving oversized nodes and network layers in the model design, however, usually causes the over-smoothing problem and the lack of embedding discrimination. In this article, we propose RoSGAS, a novel Reinforced and Self-supervised GNN Architecture Search framework to adaptively pinpoint the most suitable multi-hop neighborhood and the number of layers in the GNN architecture. More specifically, we consider the social bot detection problem as a user-centric subgraph embedding and classification task. We exploit the heterogeneous information network to present the user connectivity by leveraging account metadata, relationships, behavioral features, and

Renyu Yang Co-first author with equal contribution.

Zhiqin Yang and Yue Wang are supported by the National Key R&D Program of China through grant 2021YFB1714800 and S&T Program of Hebei through grant 20310101D. Yangyang Li is supported by NSFC through grant U20B2053. Yingguang Yang, Kai Cui, and Haiyong Xie are supported by the National Key R&D Program of China through grant SQ2021YFC3300088. Renyu Yang and Jie Xu are supported by UK EPSRC Grant (EP/T01461X/1), UK Turing Pilot Project funded by the Alan Turing Institute. Renyu Yang is also supported by the UK Alan Turing PDEA Scheme.

Authors' addresses: Y. Yang, K. Cui and H. Xie (corresponding author), School of Cyber Science and Technology, University of Science and Technology of China, 96 Jinzhai Road, Hefei, Anhui, China, 230026; emails: {dao, kaicui}@mail.ustc.edu.cn, hxie@ustc.edu.cn; R. Yang, University of Leeds, Woodhouse Lane, Leeds, West Yorkshire, United Kingdom, LS2 9JT; email: r.yang1@leeds.ac.uk; Y. Li (corresponding author), National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data, 11 Shuangyuan Road, Beijing, Beijing, China; email: liyangyang@cetc.com.cn; Z. Yang and Y. Wang, Beihang University, 37 Xueyuan Road, Beijing, Beijing, China, 100191; emails: {yangzqccc, zb2039111}@buaa.edu.cn; J. Xu, University of Leeds,Woodhouse Lane, Leeds,West Yorkshire, United Kingdom, LS2 9JT, and Beihang University, 37 Xueyuan Road, Beijing, China; email: j.xu@leeds.ac.uk.

1559-1131/2023/05-ART15 $15.00
https://doi.org/10.1145/3572403

content features. RoSGAS uses a multi-agent deep reinforcement learning (RL) mechanism for navigating the search of optimal neighborhood and network layers to learn individually the subgraph embedding for each target user. A nearest neighbor mechanism is developed for accelerating the RL training process, and RoSGAS can learn more discriminative subgraph embedding with the aid of self-supervised learning. Experiments on five Twitter datasets show that RoSGAS outperforms the state-of-the-art approaches in terms of accuracy, training efficiency, and stability and has better generalization when handling unseen samples.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; **Machine learning approaches;**

Additional Key Words and Phrases: Graph neural network, architecture search, reinforcement learning

ACM Reference format:
Yingguang Yang, Renyu Yang, Yangyang Li, Kai Cui, Zhiqin Yang, Yue Wang, Jie Xu, and Haiyong Xie. 2023. RoSGAS: Adaptive Social Bot Detection with Reinforced Self-supervised GNN Architecture Search. *ACM Trans. Web* 17, 3, Article 15 (May 2023), 31 pages.
https://doi.org/10.1145/3572403

1 INTRODUCTION

Social bots—the accounts that are controlled by automated software and mock human behaviors [1]—widely exist on online social platforms such as Twitter, Facebook, Instagram, and Weibo and normally have intentions attempts. For example, interest groups or individuals can use social bots to influence the politics and economy, e.g., swaying public opinions at scale, through disseminating disinformation and on-purpose propaganda, and to steal personal privacy through malicious websites or phishing messages [47]. Such deception and fraud can reach out to a huge community and lead to cascading and devastating consequences.

Social bots have been long studied but not yet well resolved due to fast bot evolution [7]. The third generation of bots since 2016 with deepened mixture of human operations and automated bot behaviors managed to disguise themselves and survived platform-level detection using traditional classifiers [7, 9]. The cat-and-mouse game continues—while new work-around, camouflage, and adversarial techniques evolve to maintain threats and escape from perception, a huge body of detection approaches has emerged to differentiate the hidden behaviors of the emerging socials bots from legitimate users. The recent advancements in **Graph Neural Networks (GNNs)** [40] can help to better understand the implicit relationships between abnormal and legitimate users and thus improve the detection efficacy. GNN-based approaches [6, 14, 15, 17, 31, 37, 50] formulate the detection procedure as a node or graph classification problem. Heterogeneous graphs are constructed by extracting the accounts' metadata and content information from social networks before calculating numerical embedding for nodes and graphs. However, there are still several interrelated problems to be addressed:

GNN architecture design has a strong dependence upon domain knowledge and manual intervention. In most of the existing works, the embedding results are inherently flat because the neighbor aggregation ignores the difference between the graph structure pertaining to the target node and the structures of other nodes. This will result in a lack of deterministic discrimination among the final embeddings when the scale of the formed graph structure grows to be tremendous. To address this issue, subgraphs are leveraged to explore the local substructures that merely involve partial nodes, which can obtain non-trivial characteristics and patterns [45]. However, the subgraph-neural-network-based approaches heavily rely on experiences or domain knowledge in the design of rules for extracting subgraphs and of model architectures for message aggregation [3, 29, 35]. This manual intervention substantially impedes the elaborated design of a neural network model that can adapt to the evolving changes of the newer social bots. Using fixed and fine-grained subgraph extraction rules is not sufficiently effective [9, 18].

Over-assimilated embedding when aggregating a huge number of neighbors. The most intractable and demanding task is to effectively perceive and pinpoint the camouflages of the new-generation social bots. Camouflage technology mainly comprises two distinct categories—feature camouflage and relation camouflage. Feature camouflage is referred to as the procedure where bots steal the metadata of the benign accounts and transform it into their own metadata. They also employ advanced-generation technology to create content information [7]. Apart from mimicking features of legitimate users, relation camouflage techniques further hide the malicious behaviors by indiscriminately interacting with legitimate users and establishing friendships with active benign users [57]. The interactions, particularly with the influencers, can considerably shield the bots from being detected. It is thus critical to include sufficient heterogeneous nodes in the neighborhood when extracting the subgraph for the target user so that camouflaged bots can be picked up. Meanwhile, it is also important to differentiate the subgraph embeddings of different target users while similarizing the embeddings within the same target user. However, the over-smoothing representation problem will manifest [30, 53] as the involvement of a huge number of nodes in the GNNs tends to over-assimilate the node numerical embedding when aggregating its neighbor information.

Inadequate labeled samples. A large number of labeled samples are presumably acquired and massively used in the supervised model training. However, this assumption can be hardly achieved in real-world social bot detection. In fact, there are always very limited users with annotated labels or limited access to adequate and labeled samples [21]. This will hamper the precision of supervised deep learning models and particularly lead to poor performance in identifying out-of-sample entities, i.e., the new types of social bots out of the existing datasets or established models.

To address these issues, the state-of-the-art works [20, 27, 52, 62, 63] adopt **reinforcement learning (RL)** to search the GNN architecture. However, such approaches sometimes lack generalizability; the effectiveness of determining the optimal GNN architecture is tightly bound to specific datasets and usually have huge search space, and hence low efficiency. They are not suited for social networks where the graph structure follows a power-law distribution [4, 34]. In this scenario, dense and sparse local graph structures co-exist and huge disparities manifest among different users. Additionally, as only taking a small fraction of the labelled users as the environment state, the existing RL agents can hardly learn the state space in an accurate manner and will lead to a slow convergence in the RL agents. Hence, it is highly imperative to personalize the selection of subgraphs and GNN architecture of the model for each target user, without prior knowledge and manual rule extraction, and to devise automated and adaptive subgraph embedding to fit the ever-changing bot detection requirements.

In this article, we propose RoSGAS, a subgraph-based scalable Reinforced and Self-supervised GNN Architecture Search approach to adaptively extract the subgraph width and search the model architecture for better subgraph embedding, and to speed up the RL model convergence through self-supervised learning. Specifically, we use the **Heterogeneous Information Network (HIN)** to model the entities and relationships in the social networks and use meta-schema and meta-path to define the required relationship and type constraints of nodes and edges in the HIN, on the basis of real-world observations in social network platforms. We formulate the social bot detection problem as a subgraph classification problem. We first propose a multi-agent RL mechanism for improving the subgraph embedding for target users. The RL agent can start to learn the local structure of the initial 1-order neighbor subgraph of a given target user and help to select the most appropriate number of neighbor hops as the optimal width of the subgraph. The RL agent is also elaborately devised to select the optimal number of model layers such that the neural networks are well suited for encoding the dedicated subgraphs with sufficient precision, without introducing oversized architecture and computation overhead. We then exploit the residual structure to retain the characteristics of a target user as much as possible, thereby overcoming the over-smooth problem on the

occasion of message aggregations from a huge number of neighbor nodes. While using RL to automate the neighbor selection and model construction, we additionally develop a nearest neighbor mechanism in the early stage of RL for accelerating the training process of the proposed RL approach. A self-supervised learning mechanism is further investigated and integrated with the RoSGAS framework to overcome the deficiency of over-assimilated embedding. The self-supervised module can facilitate more discriminative representation vectors and help to enhance the capability of expressing discrepancies among different target users. Experimental results show that RoSGAS outperforms the state-of-the-art approaches over five Twitter datasets and can achieve competitive effectiveness when compared with hand-made design of the model architecture.

Particularly, the main contribution of this work are summarized as follows:

- Proposed for the first time a user-centric social bot detection framework based on the Heterogeneous Information Network and GNN without prior knowledge
- Developed an adaptive framework that leverages Deep Q-learning to optimize the width of the subgraph and the GNN architecture layers for the subgraph corresponding to each target user
- Investigated a nearest neighbor mechanism for accelerating the convergence of the training process in RL agents
- Proposed a self-supervised learning approach to enable homologous subgraphs to have closer representation vectors while increasing the disparities of representation vectors among non-homologous subgraphs after information aggregation
- Presented an explicit explanation for the model stability

Organization. This article is structured as follows: Section 2 outlines the problem formulation, and Section 3 describes the technical details involved in RoSGAS. The experimental setup is described in Section 4, and the results of the experiment are discussed in Section 5. More discussions are given in Section 6. Section 7 presents the related work before we conclude the article in Section 8.

2 PROBLEM FORMULATION

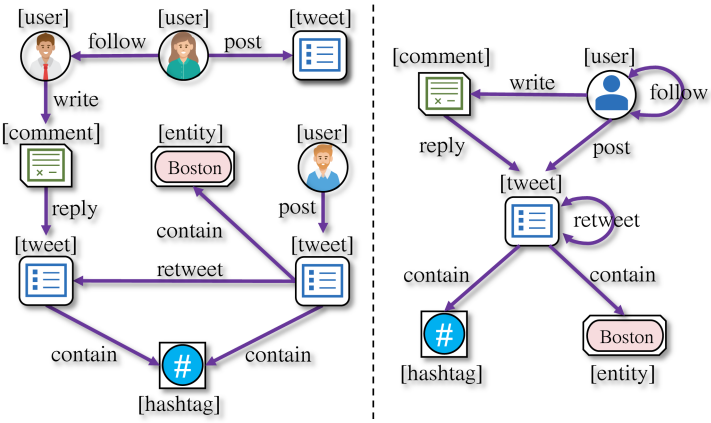
In this section, we introduce HINs and information network representation before discussing the scope of this work and formulating the target problem.

2.1 Preliminaries

In this work, we follow the terminologies used in the work of [13, 24, 36, 42] to define HIN embedding. The aim is to project different nodes in the HIN into a low-dimensional space while preserving the heterogeneous structure and relationships between heterogeneous nodes.

Definition 1 (Heterogeneous Information Network). An HIN denoted as $\mathcal{G} = \mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{R}, \varphi, \phi)$, where $\mathcal{V}$ denotes the nodes set, $\mathcal{E}$ denotes the edges set, $\mathcal{F}$ denotes the node types set, and $\mathcal{R}$ denotes the edge types set. In real-world settings, there may be multiple types of nodes or edges, i.e., $|\mathcal{F}| + |\mathcal{R}| > 2$. Each individual node $i \in \mathcal{V}$ is associated with a node type mapping function $\varphi : \mathcal{V} \rightarrow \mathcal{F}$; similarly, each individual edge $e \in \mathcal{E}$ has an edge type mapping function $\phi : \mathcal{E} \rightarrow \mathcal{R}$.

In a nutshell, real-life information networks have different structures consisting of multi-typed entities and relationships. A relationship is referred to as the link between entities in a network or graph system. For example, Figure 1(a) shows an example of a social network HIN that we construct for Twitter. It comprises five types of nodes (user, tweet, comment, entity, and hashtag) and six types of relationships (write, follow, post, reply, retweet, and contain).



(a) HIN for social network (b) HIN schema

Fig. 1. (a) An example of HIN for social network. (b) Network schema of HIN for social network.

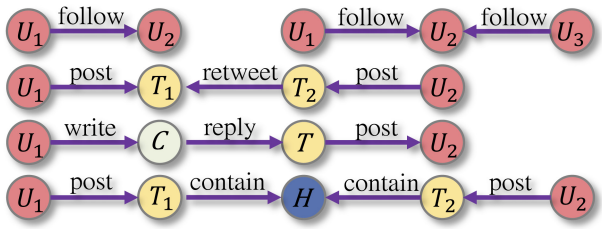


Fig. 2. The extracted meta-paths.

Definition 2 (Network Schema). Given an HIN $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{R}, \varphi, \phi)$, the network schema for network $\mathcal{G}$ can be denoted as $\mathcal{T}(\mathcal{F}, \mathcal{R})$, a directed graph with the node type set $\mathcal{F}$ and edge types set $\mathcal{R}$. In simple words, the HIN schema comprehensively depicts the node types and their relations in an HIN and provides a meta template to guide the exploration of node relationships and extract subgraphs from the HIN. Figure 1(b) exemplifies the network schema that can reflect entities and their interactions in a generic social network.

Definition 3 (Meta-path). Given a Meta Schema $\mathcal{T}(\mathcal{F}, \mathcal{R})$, a Meta-Path $\mathcal{MP}$, denoted as $\mathcal{F}_1 \xrightarrow{\mathcal{R}_1} \mathcal{F}_2 \xrightarrow{\mathcal{R}_2} \ldots \xrightarrow{\mathcal{R}_{l-1}} \mathcal{F}_l$, is a path on $\mathcal{T}$ that connects a pair of network nodes and defines the composite relation $\mathcal{R}$ that contains multiple types of relationships.

In reality, a meta-path describes the semantic relationship between nodes. Mining such a semantic relationship is the cornerstone of subsequent tasks such as classification, clustering, and so forth. As shown in Figure 2, we extracted the five most useful meta-paths from our defined meta-schema, based on observations in social network platforms.

2.2 Problem Statement

We consider the social bot detection problem as a subgraph classification task, instead of a node classification task, in a semi-supervised learning manner.

Definition 4 (Semi-supervised Subgraph Classification). Given a collection of target users, the subgraph pertaining to the ith target user can be defined as $\mathcal{G}_i = \{\mathcal{V}_i, \mathcal{X}_i, \{\mathcal{E}_r\}|_{r=1}^{R}, y_i\}$. $\mathcal{V}_i$ is the collection of nodes in the subgraph, including v_{i_0}, the target user itself, and the neighbors $\{v_{i_1}, \ldots, v_{i_n}\}$

within a few hops from the target user. These nodes are extracted from the entire graph $\mathcal{G}$ and consist of different types. Each node initially has a d-dimensional feature vector and $\mathcal{X}_i$ represents the vector set of all node embeddings, i.e., $\mathcal{X}_i = \{x_{i_0}, \ldots, x_{i_n}\}$, where each element $x \in \mathbb{R}$. The edge in the subgraph can be represented as $e^r_{i_m,i_n} = (v_{i_m}, v_{i_n}) \in \mathcal{E}_r$, where v_{i_m} and v_{i_n} are connected through a certain relationship $r \in \{1, \ldots, R\}$. $y_i \in \{0, 1\}$ represents a binary label of the target user v_{i_0}; 0 indicates benign account, while 1 represents a social bot. Once a dedicated subgraph G_i is extracted, the subsequent task is to conduct a subgraph binary classification. At the core of the adaptive architecture search is to pinpoint the subgraph width (k) and the optimal value of model layers (l) that constitute the whole bot detection model.

2.3 Research Questions

For achieving discriminative, cost-effective, and explainable subgraph embedding, there are three main research challenges facing the RL-based social bot detection:

[Q1] How to determine the right size of the subgraph for an individual target user in a personalized and cost-effective manner? The major issue with the DNN construction is the selection of neighbor hops and model layers. In fact, two interrelated yet opposite factors may affect the choice of a detection model. On the one hand, a larger number of hops and model layers can involve more nodes, including both benign and malicious nodes, in the neighbor aggregation. This is beneficial for the detection quality since the hidden camouflages of social bots could be identified by the higher-order semantic embedding enabled by the continuum of HIN-based data engineering and DNN model training. However, excessive involvement will bring performance issues in terms of time and computation efficiency and, more severely, lead to the over-smooth problem commonly manifested in graphs at scale [30]. On the other hand, a small portion of the neighbors would overlook node information and lead to less informative node embeddings. To resolve this dilemma—balancing competitive accuracy and high computation efficiency—while addressing the assimilation within the neighborhood, it is critical to automatically pick up appropriate hops of neighbors and to stack up *just enough* neural network layers to be assembled in the detection model. This requires the reinforcement learning process to properly define dedicated policies and optimize the setting of environment states and actions.

[Q2] How to accelerate the convergence of reinforcement learning? It is observable that in the initial stage of training, the learning curve substantially fluctuates, and this phenomenon will slow down the training process and model convergence. This is because in the starting phase the noisy data may take up a high portion of the limited memory buffer and thus misdirect to the wrong optimization objective. To ensure the training efficiency, it is necessary to boost the action exploration in RL agent and speed up the training stabilization.

[Q3] How to more efficiently optimize the reinforcement learning agents in the face of limited annotated labels? Data annotation is expensive and sometimes difficult in practical problem solving. If only a small portion of the labeled users are used as the input of the RL agent as environment state, the state space cannot be accurately and efficiently learned within a required time frame. This will consequently delay the optimization of an RL agent and further have a cascading impact on the multi-agent training. This issue therefore necessitates a self-supervised learning mechanism for optimizing the training effectiveness and efficiency.

3 METHODOLOGY

In this section, we will introduce how we design the social bot detection framework through adaptive GNN architecture search with reinforcement learning. We first introduce the basic process of subgraph embedding (Section 3.1). In response to **[Q1]**, we introduce a reinforcement learning enhanced architecture search mechanism (Section 3.2). To address **[Q2]**, we propose a nearest

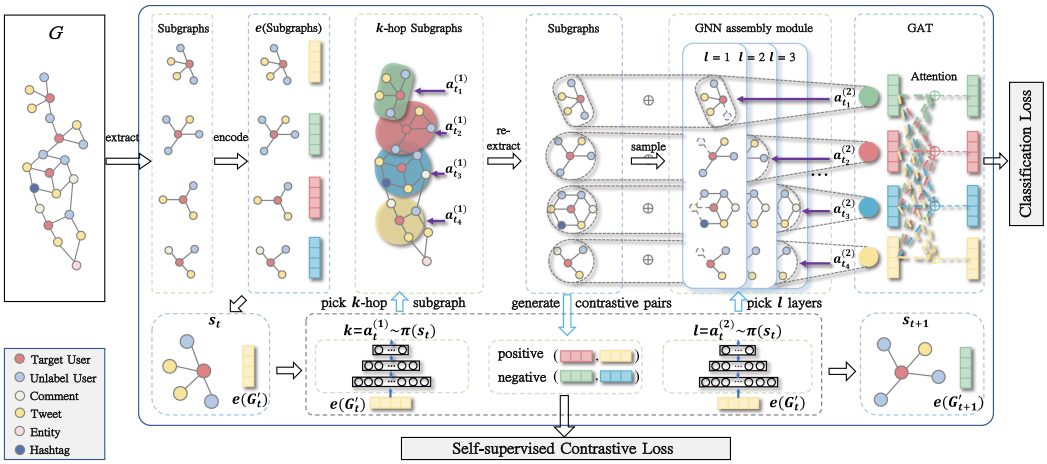


Fig. 3. The proposed RoSGAS framework.

neighbor mechanism (Section 3.3) for accelerating the convergence process of reinforcement learning. The self-supervised learning approach is discussed (Section 3.4) to tackle **[Q3]** before we present how to tackle parameter explosion and outline the holistic algorithm workflow (Section 3.5 and Section 3.6).

3.1 Overview

Figure 3 depicts the overall architecture of RoSGAS to perform the subgraph embedding. The workflow mainly consists of three parts: graph preprocessing and construction, RL-based subgraph embedding, and the final-stage attention aggregation mechanism among graph nodes before feeding into the final classifier for determining the existence of social bots. To aid discussion, Table 1 depicts the notations used throughout the article.

3.1.1 Graph Construction. Initially, the feature extraction module transforms the original information into a heterogeneous graph. The edges between nodes in the heterogeneous graph are established based on the account's friend relationships and interactions in the social network platform. We retrieve the meta features and description features of each account as its initial node feature in a similar way as [56]. Extra tweet features and entities are extracted by using NLPtool[1] from the original tweets. The composition of features for each type of node may vary. For *user* node, features such as *status*, *favorites*, *list*, and so forth are extracted from user metadata. For *tweet* node, we normally extract the number of retweets and the number of replies while embedding the tweet content into a 300-dimension vector and concatenating them together as their original features. Similarly, we embed the content of *hashtag* and *entity* to 300-dimension vectors. To simplify the feature extraction and processing, the feature vector of each node type is set to be 326 dimensions. Those with insufficient dimensions are filled with zero. More details will be given in Section 4.3.

To refine the heterogeneous graph under the given semantics, we further conduct a graph preprocessing by enforcing meta-paths upon the original graph, and only the entities and edges conforming to the given-meta paths will be retained in the graph structure. As shown in Figure 2, we extracted five meta-paths that are widely recognized in social graphs and represent most of

[1]https://github.com/explosion/spaCy.

Table 1. Notations

Symbol	Definition
$U_i; T_i; C_i; H_i$	User; tweet; comment; hashtag
$\mathcal{G}; \mathcal{G}_i; \mathcal{V}; \mathcal{E}$	Graph; ith subgraph; node set; edge set
$\mathcal{F}; \mathcal{R}$	Node type; relation type
$\varphi; \phi$	Mapping a node to the type $\mathcal{F}$; mapping an edge to the type $\mathcal{R}$
$\mathcal{V}_i; \mathcal{X}_i$	Node set of subgraph $\mathcal{G}_i$; node feature set of subgraph $\mathcal{G}_i$
$v_{i_0}; v_{i_n}; y_i$	ith target user; nth node in Subgraph $\mathcal{G}_i$; label of ith target user
$e^r_{v_{i_m}, v_{i_n}}; r$	An edge connecting v_{i_m}, v_{i_n} through a relationship $r \in \mathcal{R}$
$k; l$	The number of neighbor hops and the number of model layers
$\mathcal{D}; h_j; m_i$	The target user set; the original feature of v_j; the model for target user v_{i_0}
L; L	The last layer number and the error correction parameter
$\alpha^k_{ij}; W^k$	The kth attention coefficient and the kth weight matrix
$z_i; \pi$	The embedding of the ith subgraph and the policy network
$s_t; a_t; r_t$	The state, chose action, and reward at timestamp t
$\mathcal{R}, b$	The advantage function measurement of (s_t, a_t) and the history window size
Q, γ	The value of state-action pair and the future cumulative rewards discount parameter
$B; \tau$	The observation experience set and the state-action pair
d_τ, Θ	The state distance measurement function and the parameter of RL agent
α_0, β	The initial weight and decay rate of α in the nearest neighbor mechanism
λ	The weight parameter of the GNN loss function
$\mathcal{G}^k_i$	A subgraph $\mathcal{G}$ for ith target user with k hop neighbors
$\mathcal{L}_{pretext_i}, g_i$	The loss of the ith pretext task, the ith stacked GNN encoder
$y_{pretext_i}$	The $y_{pretext_i}$ stands for the ground truth of subgraph $\mathcal{G}_i$ acquired by pretext task
$\mathcal{G}'_i, \bar{\mathcal{G}}_i$	$\mathcal{G}'_i$ is a positive sample for $\mathcal{G}_i$; $\bar{\mathcal{G}}_i$ is a negative sample for $\mathcal{G}_i$.
$z'_i, \bar{z}_i$	z'_i is the representation of $\mathcal{G}'_i$; $\bar{z}_i$ is the representation of $\bar{\mathcal{G}}_i$

the typical behaviors in the meta-schema defined in Section 2. The main purpose is to cut down the information redundancies in the graph at scale and thus substantially improve the computational efficiency. The transformed graph will be further used to extract subgraphs for the target users before feeding the subgraphs into the downstreaming tasks including the numerical embedding and classification.

3.1.2 RL-based Parameter Searching. The primary goal of the parameter selection is to determine the appropriate subgraph width (k) and the model layers (l) for each target user in the target user collection $\mathcal{D}$ with n users. In a nutshell, for each target user, we take it as the center node and first extract a fixed width (e.g., 1 hop) subgraph $\mathcal{G}_i$ as the initial subgraph. Afterward, we encode the subgraph into the embedding space and regard the embedding as the environment state before feeding it into the reinforcement learning agent. To be more specific, the embedding representation of the ith target user can be obtained by using an encoder (e.g., average encoding operation, sum encoding operation, etc.):

$$e(\mathcal{G}_i) = \{h_j | v_j \in \mathcal{V}_i(\mathcal{G}_i)\}. \tag{1}$$

At the core of the embedding improvement is the RL agent. The preliminary encoding result will be fed into the policy $\widetilde{\pi}_1$ and $\widetilde{\pi}_2$ in the RL agent, successively. $\widetilde{\pi}_1$ is responsible for selecting the appropriate width of subgraph $\mathcal{G}'_i$, while $\widetilde{\pi}_2$ is in charge of pinpointing the most suitable number of layers for constructing model m_i for the target user v_{i_0}. The types of specific model layer can be selected from the most popular models such as GCN [26], GAT [48], and GraphSAGE [23]. Generally speaking, the goal of reinforcement learning is to maximize the expected

accuracy: $\mathbb{E}[\mathcal{R}_{\mathcal{D}}(\{m_i, \mathcal{G}_i\}|_{i=1}^{n})]$ on $\mathcal{D}$:

$$\widetilde{\pi}_1^*, \widetilde{\pi}_2^* = \underset{\theta_1, \theta_2}{\operatorname{argmax}} \mathbb{E}[\mathcal{R}_{\mathcal{D}}(\{\pi_1(e(\mathcal{G}_i); \theta_1), \pi_2(e(\mathcal{G}_i); \theta_2)\}|_{i=1}^{n})]. \tag{2}$$

More details about the learning procedure will be discussed in Section 3.2.

To calibrate the subgraph embeddings, we can then aggregate the k-hop neighbors and stack models with various l layers. However, the aggregation from the stacked GNN models would blur the original detectable features of the target user in the subgraph embedding. To mitigate this issue, we apply the residual network to aggregate the target node's input features and its corresponding embedding delivered by the last layer of the model:

$$h_j^{(L)} = ADD\left(x_{i_j}, h_j^{(L)}\right), \tag{3}$$

where L is the last layer of the stacked GNN model. Then we can apply a pooling operation (e.g., average, sum, etc.) to integrate the subgraph $\mathcal{G}_i$ into a super node:

$$z_i^{(L)} = READOUT\left(\left\{h_j^{(L)}\right\}_{j=1}^{n'}\right). \tag{4}$$

3.1.3 Attention Aggregation and Classification. We adopt an attention mechanism for integrating the influence of subgraphs belonging to the relevant neighbors into the final embedding:

$$z_i = \frac{1}{K}\sum_{k=1}^{K}\sum_{\mathcal{G}_j \in \mathcal{G}} \alpha_{ij}^k \mathrm{W}^{\mathrm{k}} z_j^{(L)}, \tag{5}$$

where α_{ij} is the attention coefficient, $\mathrm{W} \in \mathbb{R}^{\mathrm{d_L} \times \mathrm{d_l}}$ is the weight matrix, and K is the number of independent attention. z_i is the final embedding for detecting if the target user is a social bot. Eventually, the bot classifier digests the learned vector embeddings to learn a classification model and determines if a given social user behaves normally or maliciously. General-purpose techniques including Random Forest, Logistic Regression, and SVM can be adopted for implementing the classifier.

3.2 Reinforced Searching Mechanism

In this subsection, we will introduce how to obtain the optimal policies $\tilde{\pi}_1^*$ and $\tilde{\pi}_2^*$ through the searching mechanism. The learning procedure of the optimal $\tilde{\pi}_1^*$ and $\tilde{\pi}_2^*$ can be formulated as a **Markov Decision Process (MDP)**. An RL agent episodically interacts with the environment where each episode lasts for T steps. The MDP includes state space, action space, reward, and the transition probability that maps the current state and action into the next state.

State Space. In each timestamp t, the state s_t is defined as the embedding of the subgraph extracted from $\mathcal{G}$.

Action Space. Since we need two policies to pinpoint the optimal width of the subgraph and the optimal number of model layers, respectively, the action at timestep t consists of dual sub-actions $(a_t^{(1)}, a_t^{(2)})$. The RL agent integrated in our proposed framework RoSGAS performs an action $a_t^{(1)}$ to get the value of k and performs an action $a_t^{(2)}$ to get the value of l. For instance, $a_t^{(1)}$ is chosen at the timestep t to re-extract the subgraph of the target user v_{i_0}. We then calculate the number of reachable paths from target user v_{i_0} to the other target users in this subgraph as the connection strength. For those target users that are included in the collection $\mathcal{D}$ yet excluded from this subgraph, the connection value will be set to 0. $L1$ normalization is performed upon these values as the reachability probabilities from the target user to the other target users. After selecting certain actions $(a_t^{(1)}, a_t^{(2)})$ at the timestep t, the RL environment forms a probability distribution P_i.

Transition. The probability P_i serves as the state transition probability of the reinforcement learning environment. The subgraph embedding of any other target user is used as the next state s_{t+1}. The whole trajectory of the proposed MDP can be described as $(s_0, (a_0^{(1)}, a_0^{(2)}), r_0, \ldots, s_{T-1}, (a_{T-1}^{(1)}, a_{T-1}^{(2)}), r_{T-1}, s_T)$.

Reward. We need to evaluate whether the search mechanism is good enough at the current timestep t. In other words, it reflects if the parameters in the current RL agent can achieve better accuracy than the parameters at the previous timestep $t-1$. To do so, we first define a measure to flag the improvement of model accuracy when compared with the previous timesteps:

$$\mathcal{R}(s_t, a_t) = \left(\mathcal{ACC}(s_t, a_t) - \frac{\sum_{i=t-b}^{t-1} \mathcal{ACC}(s_i, a_i)}{b-1} \right), \tag{6}$$

where b is a hyperparameter that indicates the window size of historical results involved in the comparison and $\mathcal{ACC}(s_i, a_i)$ is the accuracy of subgraph classification on the validation set at timestep i. $\frac{\sum_{i=t-b}^{t-1} \mathcal{ACC}(s_i, a_i)}{b-1}$ reflects the average accuracy in the most recent b timestep windows. The training RL agent continuously optimizes the parameters to enable a rising accuracy and, accordingly, positive rewards. This will give rise to the cumulative rewards in finite timesteps and eventually achieve the optimal policies.

We use a binary reward r_t combined with Equation (6) to navigate the training direction as follows:

$$r(s_t, a_t) = \begin{cases} 1, & \text{if } \mathcal{R}(s_t, a_t) > \mathcal{R}(s_{t-1}, a_{t-1}) \\ -1, & \text{otherwise.} \end{cases} \tag{7}$$

The value is set to be 1 if a_t can increase the $\mathcal{R}$ compared with that of the previous timestep $t-1$; otherwise it will be set -1.

Termination. State-action values can be approximated by the Bellman optimal equation:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \arg\max_{a'} Q^*(s_{t+1}, a'). \tag{8}$$

Nevertheless, to improve both training speed and stability, we will introduce an enhanced approximation approach in Section 3.3. We exploit the ϵ-greedy policy to select the action a_t with respect to Q^* and obtain the policy function π:

$$\pi(a_t | s_t; Q^*) = \begin{cases} \text{random action}, & w.p.\epsilon \\ \underset{a}{\text{argmax}}\, Q^*(s_t, a), & \text{otherwise.} \end{cases} \tag{9}$$

3.3 Nearest Neighbor Mechanism for Accelerating Model Stabilization

Conventionally, at each time step t, the RL agent employs its prediction network to determine the value of state-action pairs for choosing the best action to maximize the future cumulative rewards. Inspired by [41], we applied the nearest neighbor mechanism for assisting and accelerating the training process of the RL agent. The intuition behind the scheme is that when the RL agent observes similar or the same state-action pairs, the environment is highly likely to produce a similar reward value. In other words, the distance between state-action pairs can indicate their relative reward values. Therefore, we aim to find out the similar state-action pairs and determine the reward of the current state-action pair by combining the reward estimated by the Q-network (i.e., the prediction network) and the reward of the existing similar pairs. This means that the model training benefits from both the RL environment and the prediction network, which can boost the action exploration and accelerate the training stabilization.

To look into and record the historical actions, we set up an observation experience set $B = \{\tau_1, \ldots, \tau_n\}$; each element τ_i in B represents a pair of explored state s_i and the corresponding

selected action a_i, namely, (s_i, a_i). While recording the action-state pairs, we also record the corresponding value labels $\{Q(\tau_i)\} \subseteq \mathbb{R}$. We employ the distance function d_τ—for example, using cosine to calculate the similarity—to measure the distance between the explored state-action pairs and the incoming state-action pairs. We use the distance to ascertain the nearest neighbor of the state-action pair to be estimated from the recorded state-action pairs. Subsequently, the value label of the nearest neighbor can be used to estimate the value of the state-action pair:

$$\hat{Q}(\tau) = \min\{\mathrm{Q}(\tau_\mathrm{i}) + \mathrm{L} \cdot \mathrm{d}_\tau(\tau, \tau_\mathrm{i})\}|_{\mathrm{i=1}}^{\mathrm{n}}, \tag{10}$$

where $\hat{Q}(\tau)$ is the estimated value of τ, and L is a parameter to correct the estimated error.

We combine $\hat{Q}(\tau)$ estimated by the nearest neighbor mechanism and $Q(s_t, a_t; \Theta^{target})$ estimated by the target network into a new estimated value $\hat{Q}(s_t, a_t)$:

$$\hat{Q}(s_t, a_t) = \alpha \cdot \hat{Q}(\tau_t) + (1 - \alpha) \cdot (r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \Theta^{target})), \tag{11}$$

where α is an exponentially decaying weight parameter and $\hat{Q}(\tau_t)$ is the estimated value of $\tau_t = (s_t, a_t)$ by using the nearest neighbor search mechanism. In fact, α is used to assist the RL optimization in the early stage and gradually reduce the effect of the proposed nearest neighbor mechanism when the training procedure moves forward. To achieve this, we set $\alpha = \alpha_0 \cdot (1 - \beta)^k$, where $\alpha_0 \in (0, 1]$ is an initial weight, $\beta \in (0, 1)$ is a decay rate, and k is the episode number. We then define the RL loss function as

$$L(\Theta) = (\hat{Q}(s_t, a_t) - Q(s_t, a_t; \Theta^{pred}))^2, \tag{12}$$

where Θ^{pred} is the parameter of the DQN agent's prediction network, and the Θ^{target} is the parameter of the target network.

3.4 Self-supervised Learning

To better differentiate the subgraph representations among graphs, we propose a contrastive self-supervised learning approach to maximize the difference between two distinct patches, without relying on the human-annotated data samples. The task of self-supervised learning (also known as pretext task) is to minimize the distance between positive samples while maximizing the distance between negative samples. In the context of this work, all substructures (e.g., subgraphs with 1-hop neighbors or 2-hop neighbors) pertaining to the same user should have similar representation vectors. Subgraphs belonging to two distinct target users should have discriminative embeddings.

In general, given a subgraph $\mathcal{G}_i = \{\mathcal{V}_i, \mathcal{X}_i, \{\mathcal{E}_r\}|_{r=1}^R, y_i\}$, the loss $\mathcal{L}_{pretext}$ for a self-supervised learning task can be defined as follows:

$$\mathcal{L}_{pretext}(\mathcal{A}_i, \mathcal{X}_i, g_i) = \phi(g_i(\mathcal{G}_i), y_{pretext_i}), \tag{13}$$

where g_i is the stacked GNN encoder for the extracted subgraph $\mathcal{G}_i$ and the $y_{pretext_i}$ stands for the ground truth of subgraph $\mathcal{G}_i$ acquired by a specific self-supervised pretext task.

Practically, the key step is to generate positive samples in our self-supervised pretext task. After the RL algorithm outputs the customized width value k for the target node v_i, we randomly select a $\bar{k} \in [1, K]! = k$ as the width of a new subgraph $\mathcal{G}_i'$ to serve a new positive sample of the original subgraph $\mathcal{G}_i$. Meanwhile, to provide the negative sampled counterparts, we randomly select the target user v_j from the target user set $\mathcal{D}$ and directly use the learned value k. We use the stacked GNN encoder g_i and the proposed RL pipelines to perform feature extraction and summary and obtain the final subgraph embedding z_i, z_i', $\bar{z}_i$ for the subgraphs $\mathcal{G}_i$, $\mathcal{G}_i'$, $\bar{\mathcal{G}}_i$, respectively. Then we use the margin Triplet loss for model optimization to obtain high-quality representations that can well distinguish the positive and negative samples. The loss function is defined

as follows:

$$\mathcal{L}_{pretext_i} = -max(\sigma(z_i z_i') - \sigma(z_i \bar{z}_i) + \epsilon, 0), \tag{14}$$

where ϵ is the margin value. The loss function of the pretext task will be incorporated into the holistic loss function as the optimization objective of RoSGAS. Since there may exist many overlapping nodes between different subgraphs, especially in a large-scale social graph, the adoption of the loss function can effectively avoid excessive differentiation between positive and negative samples and prevent any performance degradation of representation.

3.5 Parameter Sharing and Embedding Buffer Mechanism

The customized model construction for each individual target user will lead to a substantial number of training parameters. We use the following two schemes to alleviate this problem:

- *Parameter Sharing*: We first determine a maximum base layer number k to initialize the model and then repeatedly stack the whole or part of these layers according to a_t output from the RL agent in each timestep t in the initialization order. This can avoid training a large number of model parameters.
- *Embedding Buffer*: We buffer the embeddings of the relevant subgraphs as a batch to carry out $a_t^{(2)}$ in each timestep to reduce unnecessary operations for embedding passing. Once the buffer space approaches the specified batch size, the model re-construction will be triggered by leveraging the obtained number of layers from $a_t^{(2)}$ and adopting the buffered embeddings. We cleanse the buffer space once the GNN model training terminates to ensure the buffer can be refilled in the later stage.

3.6 Put Them Together

Algorithm 1 summarizes the overall training process of the proposed RoSGAS including the initialization of the subgraph embedding and the follow-up architecture search via the proposed RL process. Specifically, we first construct the social graph according to our definition in Section 1 before initializing the GNN model with the max layers L and the parameters of the two RL agents (Line 1). At the training stage, we randomly sample a target user and embed its k_{init}-hop subgraph as the initial state s_0 (Lines 2–4). Afterward, at each time step, an action pair was chosen to indicate the width k of the subgraph and the number l of GNN layers for the target user (Line 6). Then we re-extract subgraph $\mathcal{G}_t^{'}$ and store $\mathcal{G}_t^{'}$ and the value l represented by $a_t^{(2)}$ into the buffer $\mathcal{B}$ (Lines 7–9). Once the number of $\mathcal{G}_t^{'}$ reaches a threshold value B_D, we stack the GNN model with $a_t^{(2)}$ layers and generate the positive and negative pair for the bth target user in $\mathcal{B}$. Then we train the model together with the self-supervised learning mechanism described in Section 3.4 (Lines 11–15). After the stacked GNN model training, we validate it on the validation dataset to get the reward r_t (Line 18) and store the corresponding transition into memory $\mathcal{M}^{(1)}$ and $\mathcal{M}^{(2)}$ (Lines 20–21).

To optimize the π_1, we fetch batches of transitions from $\mathcal{M}^{(1)}$. For a specific transition $T_t^{(1)} = (s_t, a_t^{(1)}, s_{t+1}, r_t)$, we use the Q-network to select the best action $a_{t+1}^{(1)}$ for the state s_{t+1} and use the target network to estimate the target value $Q^{target} = r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \Theta^{target})$. Then we use the nearest neighbor mechanism to search the nearest neighbor of state-action pair $(s_t, a_t^{(1)})$ and add its reward value upon the value of Q^{target} to obtain a new target value $\hat{Q}(s_t, a_t)$. Then we can optimize the π_1 through Equation (12). This method is also applied to optimize π_2 (Lines 22–24). Eventually, we retrain the GNN with the help of the trained policies π_1 and π_2 (Line 26). The final embedding z_i of each targeted user will be produced by the last attention layer and used for the downstream classification task.

ALGORITHM 1: The overall process of RoSGAS

Input: The max neighbor hop number, layer number: K and L; initial neighbor hop number k_{init}, the batch size of GNN and DQN: B_G, B_D, DQN training step S, the total training epoch T, epsilon value ϵ, the window size b, the error correction parameters L, the initial decay parameter α_0, the decay rate β, the full graph $\mathcal{G}$, targeted node set V.

Initialize L GNN layers, RL agent networks π_1, π_2;
Initialize the memory buffer $\mathcal{M}^{(1)}$, $\mathcal{M}^{(2)}$, and the GNN buffer $\mathcal{B}$;
Sample a target node, extract k_{init}-hop subgraph $\mathcal{G}_0$;
$s_0 = e(\mathcal{G}_0)$ via Equation (1);
for $t = 0, 1, \ldots, T$ **do**
    $a_t = (a_t^{(1)}, a_t^{(2)})$ via Equation (9);
    Re-extract subgraph $\mathcal{G}_t^{'}$ and $e(\mathcal{G}_t^{'})$ via Equation (1);
    Sample meta-path instances to get new $\mathcal{G}_t^{'}$;
    Store $\mathcal{G}_t^{'}$ and $a_t^{(2)}$ into buffer $\mathcal{B}$;
    if size($\mathcal{B}$) > B_D **then**
        Stack $a_t^{(2)}$ layers GNN ;
        for $b = 1, \ldots, B_G$ **do**
            Generate positive and negative pair for the bth target user in $\mathcal{B}$;
            Train the stacked model on the buffer of action $a_t^{(2)}$ via Equation (15);
        end
        Clear the buffer for $a_t^{(2)}$ in $\mathcal{B}$;
    end
    Obtain r_t on validation dataset via Equation (6);
    Jump to the next subgraph $\mathcal{G}_{t+1}$ and $e(\mathcal{G}_{t+1})$;
    Store the $T_t^{(1)} = (s_t, a_t^{(1)}, s_{t+1}, r_t)$ into $\mathcal{M}^{(1)}$;
    Store the $T_t^{(2)} = (s_t, a_t^{(2)}, s_{t+1}, r_t)$ into $\mathcal{M}^{(2)}$;
    for step=*1,...,S* **do**
        Optimize π_1 and π_2 via Equation (12) ;
    end
end
Re-init GNN to train via Equation (15) with $\widetilde{\pi}_1^*, \widetilde{\pi}_2^*$;

We combine the loss function of GNN and the self-supervised loss $\mathcal{L}_{pretext}$ in Equation (13). The loss $\mathcal{L}$ of RoSGAS is defined as follows:

$$\mathcal{L} = \sum_{i=1}^{n}(-\log\left(y_i \cdot \sigma\left(MLP\left(z_i\right)\right)\right) + \mathcal{L}_{pretext_i}) + \lambda \parallel \Theta \parallel_2, \tag{15}$$

where the first term represents the cross-entropy loss function and $\parallel \Theta \parallel_2$ is the $L2$-norm of GNN model parameters; λ is a weighting parameter. MLP reduces the embedding dimension of z_i to the number of categories.

4 EXPERIMENTAL SETUP

4.1 Software and Hardware

We implement RoSGAS with Pytorch 1.8.0, Python 3.8.10, and PyTorch Geometric [19] with sparse matrix multiplication. All experiments are executed on a sever with an NVIDIA Tesla V100 GPU, 2.20GHz Intel Xeon Gold 5220 CPU with 64GB RAM. The operating system is Ubuntu 16.04.6. To

Table 2. Statistics of Datasets

Dataset	Nodes	Edges	Benign	Bots	Labels	Un-Labels
Cresci-15	2,263,472	10,782,235	1,950	3,339	5,289	99.77%
Varol-17	1,978,967	4,916,116	1,244	639	1,883	99.90%
Vendor-19	3,208,255	11,479,317	1,893	569	2,462	99.92%
Cresci-19	669,616	3,341,084	269	297	566	99.92%
Botometer-Feedback	468,536	1,333,762	276	82	358	99.92%

improve the training efficiency and avoid overfitting, we employ the mini-batch technique to train RoSGAS and other baselines.

4.2 Datasets

We build a heterogeneous graph for experiments based upon five public datasets. The detailed descriptions of these datasets are as follows:

- Cresci-15 [8] encompasses two types of benign accounts, including (1) TFP, a mixture of account sets from researchers and social media experts and journalists, and (2) E13, an account set consisting of particularly active Italian Twitter users. Three types of social bots were collected from three different Twitter online markets, called FSF, INT, and TWT.
- Varol-17 [47] collects 14 million accounts of Twitter during 3 months in 2015; 3,000 accounts are sampled and selected according to some given rules. These accounts are then manually annotated into benign accounts and bot accounts.
- Vendor-19 [55] includes a collection of fake followers deriving from several companies. To create a mixture of benign and bot accounts, we mix Vendor-19 with Verified [56] that contains benign accounts only.
- Cresci-19 [32] contains the accounts that are associated with Italian retweet, collected between June 17 and 30, 2018.
- Botometer-Feedback [55] stems from social bot detector Botometer. The dataset contains manually annotated accounts based on the feedback from Botometer.

The statistics of these datasets are outlined in Table 2. The number of each class of labeled nodes in each dataset is basically balanced.

4.3 Feature Extraction

The original datasets above merely include the metadata such as age, nickname, and so forth and the posted tweet of the social account. This information, however, is insufficient to construct the heterogeneous graph, required for the effective subgraph embedding. These publicly released datasets originally included the social accounts' metadata (e.g., accounts' age, nickname) and the account's posted tweet data. Since these public data are not enough to construct the heterogeneous graph we designed, we use Twitter APIs to further crawl and obtain the metadata and tweet data of the friends and followers of the original accounts. We form these nodes into a huge heterogeneous social graph via the multiple relationships aforementioned in Figure 2. We then use the NLP toolkit spaCy to extract name entities and treat them as a type of node in the heterogeneous graph.

In addition, we extracted the original feature vector for each type of node in the heterogeneous graph and further explored the following information as additional features:

- *Account nodes:* We embed the *description* field of the account metadata into a 300-dimension vector through the pre-trained language model Word2Vec. We also extracted some features

such as *status*, *favorites*, and *list* field that would be helpful for bot detection according to [56]. We count the number of followers and the number of friends as the key account features, because the number of followers and friends are the most representative of a Twitter user, and using them could more efficiently and accurately describe a Twitter user. In addition, we divide the numbers by the user account lifetime (i.e., the number of years since the account was created) to reflect the changes during the whole life-cycle of an account. We also use Boolean value to flag the fields including *default_profile*, *verified*, and *profile_use_background_image* and count the length of *screen_name*, *name* and *description* fields and the number of digits in *screen_name* and *name* fields. We combine all these values as the original node features.

- *Tweet nodes*: We embed the text of the original tweet by the pre-trained language model Word2Vec into a 300-dimension vector. Apart from the original node features, we further combine additional information—the number of retweets, the number of replies, the number of favorites, the number of mentioning of the original tweet, and the number of hashtags and the number of URLs involved in the tweet.
- *Hashtag and entity nodes:* We also embed the text of hashtag and entity into a 300-dimension vector. We use zero to fill the blank holes if the number of dimension is less than 300.

The graph constructed by each dataset contains millions of edges and nodes, which greatly increases the difficulty of the social bot detection task. Noticeably, most samples in the datasets are unlabeled; i.e., more than 99% samples are not annotated.

4.4 Baselines and Variations

4.4.1 Baselines. To verify the effectiveness of our proposed RoSGAS, we compare with various semi-supervised learning baselines. Because these baselines will run on very large-scale graphs, to ensure training and inference on limited computing resources, we use the PyG [19] calculation on sparse matrix multiplication to implement these baselines. Detail about these baselines are described as follows:

- **Node2Vec** [22] is built upon DeepWalk and introduces two extra biased random walk methods, BFS and DFS. Compared with the random walk without any guidance, Node2Vec sets different biases to guide the procedure of the random walk between BFS and DFS, representing the structural equivalence and homophily at the same time.
- **GCN** [26] is a representative of the spectral graph convolution method. It uses the first-order approximation of the Chebyshev polynomial to fulfill an efficient graph convolution architecture. GCN can perform convolutional operation directly on graphs.
- **GAT** [48] is a semi-supervised homogeneous graph model that utilizes an attention mechanism for aggregating neighborhood information of graph nodes. GAT uses self-attention layers to calculate the importance of edges and assign different weights to different nodes in the neighborhood. GAT also employs a multi-head attention to stabilize the learning process of self-attention.
- **GraphSAGE** [23] is a representative non-spectrogram method. For each node, it samples neighbors in different hops for the node and aggregates the features of these neighbors to learn the representation for the node. GraphSAGE improves the scalability and flexibility of GNNs.
- **GraphSAINT** [59] is an inductive learning method based on graph sampling. It samples subgraphs and performs GCN on them to overcome the neighbor explosion problem while ensuring unbiasedness and minimal variance.

- **SGC** [51] is a simplified graph convolutional neural network. It reduces the excessive complexity of GCNs by repeatedly removing the non-linearity between GCN layers and collapsing the resulting function into a single linear transformation. This can ensure competitive performance when compared with GCN and significantly reduce the size of parameters.
- **ARMA** [5] is a non-linear and trainable graph filter that generalizes the convolutional layers based on polynomial filters. ARMA can provide GNNs with enhanced modeling capability.
- **Policy-GNN** [27] is a meta-policy framework that adaptively learns an aggregation policy to sample diverse iterations of aggregations for different nodes. It also leverages a buffer mechanism for batch training and a parameter sharing mechanism for diminishing the training cost.

4.4.2 Variants. We generate several variants of the full RoSGAS model to more comprehensively understand how each module works in the overall learning framework and better evaluate how each module individually contributes to the performance improvement. RoSGAS mainly comprises three modules: Reinforced Searching Mechanism, Nearest Neighbor Mechanism, and self-supervised learning. We selectively enable or disable some parts of them to carry out the ablation study.

The details of these variations are described as follows:

- **RoSGAS-*K*:** This variant only enables the reinforced searching, without the aid of any other modules, to find out the width (k) of the subgraph for every target user v_i. Due to the huge scale of the constructed social graph, the search range will be limited to $[1, 2]$ to prevent the explosion of neighbors. Nevertheless, such search range can be flexibly customized to adapt to any other scenarios and datasets. In the context of this model variant, the number of layers is fixed to be $l = 3$.
- **RoSGAS-*L*:** This variant only switches on the reinforced searching mechanism for pinpointing the number of the layers (l) to stack the GNN model for every target user v_i. The search range will be limited to $[1, 3]$ to save computing resources. The width k of the subgraph is fixed at $k = 2$.
- **RoSGAS-*KL*:** This variant enables both the subgraph width search and the layer search in the reinforced searching mechanism for each target user v_i. In this model variant, the width is set to be $k \in [1, 2]$, while the number of layers of the GNN model is set to be $l \in [1, 3]$.
- **RoSGAS-*KL-NN***: This variant utilizes the reinforced search mechanism, together with the nearest neighbor mechanism. In the optimization process of the RL agent, the nearest neighbor module can stabilize the learning in the early stages of RL as soon as possible and accelerate the model convergence.
- **RoSGAS** contains all modules in the learning framework. The self-supervised learning mechanism is additionally supplemented upon RoSGAS-*KL-NN*.

4.5 Model Training

We use the following setting: embedding size (64), batch size (64), base layer of RoSGAS (GCN), learning rate (0.05), optimizer (Adam), L2 regularization weight ($\lambda_2 = 0.01$), and training epochs (30). As aforementioned in Section 4.4.2, we set the action (range) of searching GNN layers from 1 to 3 and the action (range) of subgraph width searching from 1 to 2 to prevent neighbors from exploding for the DQN [33]. The agent training episodes are set to be 20, and we construct five layers of MLP with 64, 128, 256, 128, and 64 hidden units. We use the accuracy obtained from the validation set to select the best RL agent and compare the performance with the other models in the test set. As for the nearest neighbor mechanism, we set the (L = 7), the initial $\alpha_0 = 0.5$.

4.6 Evaluation Metrics

As the number of labeled benign accounts and malicious accounts in the several datasets used is well balanced, we utilize Accuracy to indicate the overall performance of classifiers:

$$Acurracy = \frac{TP + TN}{TP + FP + FN + TN}, \tag{16}$$

where TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative.

5 EXPERIMENT RESULTS

In this section, we conduct several experiments to evaluate RoSGAS. We mainly answer the following questions:

- **Q1:** How different algorithms perform in different scenarios, i.e., algorithm effectiveness (Section 5.1)
- **Q2:** How each individual module of RoSGAS contributes to the overall effectiveness (Section 5.2)
- **Q3:** How the RL search mechanism work in terms of effectiveness and explainability (Section 5.3)
- **Q4:** How fast different algorithms can results, i.e., efficiency (Section 5.4), and how the RL algorithms can converge effectively (Section 5.5)
- **Q5:** How different algorithms perform when dealing with previously unseen data, i.e., generalization (Section 5.6)
- **Q6:** How to explore the detection result and visualize the high-dimensional data (Section 5.7)

5.1 Overall Effectiveness

In this section, we conduct experiments to evaluate the accuracy of the social bot detection task on the five public social bot detection datasets. We report the best test results of baselines, RoSGAS, and the variants. We performed 10-fold cross-validation on each dataset. As shown in Table 3, RoSGAS outperforms other baselines and different variants under all circumstances. This indicates the feasibility and applicability of RoSGAS in wider ranges of social bot detection scenarios. Compared with the best results among all the state of the art, our method can achieve 3.38%, 9.55%, 5.35%, 4.86%, and 1.73% accuracy improvement on the datasets of Cresci-15, Varol-17, Vendor-19, Cresci-19, and Botometer-Feedback, respectively.

In the baselines, Node2Vec is always among the worst performers in the majority of datasets. This is because Node2Vec controls the random walk process by setting a probability p to switch between the BFS and the DFS strategy. Node2Vec sometimes fails to obtain the similarity of adjacent nodes in large-scale social graphs with extremely complex structures and does not make good use of node features. GraphSAGE samples the information of neighbors for the aggregation for each node. This design can not only reduce information redundancy but also increase the generalization ability of the model through randomness. However, the proposed random sampling is not suited for super large-scale graphs, and the inability to adapt to the change of the receptive field drastically limits its performance. GCN multiplies the normalized adjacency matrix with the feature matrix and then multiplies it with the trainable parameter matrix to achieve the convolution operation on the whole graph data. However, obtaining the global representation vector for a node through full-graph convolution operations would massively reduce the generalization performance of the model. Meanwhile, the increases of receptive field will lead to a soaring number of neighbors and thus weaken the ability of feature representation. GAT shares a weight matrix for node-wise feature transformation and then calculates the importance of a node to another neighbor node before

Table 3. Comparison of the Average Accuracy of Different Methods for Social Bot Detection (Unit: %)

Method	Cresci-15	Varol-17	Vendor-19	Cresci-19	Botometer-Feedback
Node2Vec [22]	73.02 ± 0.91	61.43 ± 0.8	76.13 ± 2.61	76.65 ± 4.97	75.68 ± 0.81
GraphSAGE [23]	91.94 ± 1.12	65.71 ± 1.62	81.65 ± 1.19	70.43 ± 0.31	76.16 ± 0.43
GCN [26]	94.22 ± 0.57	65.15 ± 1.85	85.84 ± 0.81	72.75 ± 2.65	76.02 ± 0.34
GAT [48]	94.05 ± 0.31	64.63 ± 1.45	78.04 ± 0.78	71.74 ± 1.63	75.21 ± 1.46
SGC [51]	89.32 ± 0.65	63.32 ± 7.24	78.71 ± 0.75	64.34 ± 1.67	74.51 ± 0.35
GraphSAINT [59]	80.93 ± 0.92	64.89 ± 1.26	77.60 ± 2.51	78.55 ± 1.14	75.51 ± 0.59
Policy-GNN [27]	93.44 ± 0.13	66.20 ± 4.21	82.01 ± 0.23	80.19 ± 7.35	76.61 ± 1.17
ARMA [5]	94.71 ± 0.43	65.43 ± 3.24	83.44 ± 1.94	78.46 ± 1.52	75.77 ± 0.71
RoSGAS-*K*	93.13 ± 0.09	66.06 ± 0.19	76.88 ± 0.78	80.38 ± 0.29	77.09 ± 0.42
RoSGAS-*L*	97.82 ± 0.10	67.42 ± 0.37	77.40 ± 0.45	82.80 ± 0.16	77.25 ± 0.90
RoSGAS-*KL*	97.82 ± 0.14	68.01 ± 2.72	87.84 ± 1.22	82.95 ± 1.13	77.09 ± 0.14
RoSGAS-*KL-NN*	97.84 ± 0.05	75.11 ± 0.07	90.52 ± 0.03	84.88 ± 0.01	77.93 ± 0.14
RoSGAS	**98.09 ± 0.36**	**75.75 ± 0.31**	**91.19 ± 0.35**	**85.05 ± 0.21**	**78.34 ± 0.25**
Gain	**3.38** ↑	**9.55** ↑	**5.35** ↑	**4.86** ↑	**1.73** ↑

aggregating the output feature vector of the central node through the weighted product and summation. GAT also experiences the explosion of receptive field and the consequent increase of the neighbor number. Unlike GraphSAGE, GraphSAINT sets up a sampler to sample subgraphs from the original graph. It uses GCN for convolution on the subgraph to resolve neighbor explosion, and sampling probability is set to ensure unbiasedness and minimum variance. However, extracting subgraphs in GraphSAINT is random and thus limits the precision of subgraph embedding. SGC simplifies the conventional GCNs by repeatedly removing the non-linearities between GCN layers and collapsing the resulting function into a single linear transformation. Namely, the nonlinear activation function between each layer is removed to obtain a linear model. Compared with GCN, SGC can achieve similar performance, with a slightly reduced accuracy among all datasets. In addition, RoSGAS also outperforms Policy-GNN and ARMA since these counterparts merely make exclusive improvement on convolution layers. By comparison, our approach takes advantage of subgraph search and GNN architecture search, while leveraging self-supervised learning to overcome the limitation of limited labeled samples. These functionalities can fulfill better performance even only based upon the basic GCN layer.

5.2 Ablation Study

The second half of Table 3 also compares the performance of multiple variants to demonstrate how to break down the performance gain into different optimization modules.

While RoSGAS-*K* only uses the RL agent for subgraph width search, the achieved effectiveness is competitive against the state of the art in some datasets, such as Varol-17, Cresci-19, and Botometer-Feedback. RoSGAS-*L*, which only enables the architecture search, can even achieve better accuracy than other baselines in most datasets, except the Vendor-19 dataset. These observations are in line with the enhancement provided by the RL capability. Intrinsically, putting the searching mechanisms together will bring synergetic benefits to the effectiveness. Particularly, for dataset Vendor-19, the combination of width search and layer search can significantly make RoSGAS superior to the other counterparts, reaching 87.84% accuracy on average. We can also observe that the contribution of layer search to the performance gain appears to be more significant than the width search. This is because the embedding effectiveness is less insensitive to the neighbor selection, and the number of GNN layers can more effectively affect the overall effectiveness. The

increased layer can make the nodes closer to the target node more frequently aggregated to form the embedding of the target node. Likewise, nodes far from the target node will be less involved in the embedding. Hence, an enhanced embedding effectiveness can be gained from enabling GNN layer search. These findings indicate the necessity of jointly searching the width of subgraphs and the number of GNN layers to unlock the full potential of performance optimization.

As shown in the result of RoSGAS-KL-NN, integrating the nearest neighbor mechanism with the backbone searching mechanism can further enhance the performance gain stemming from the learning process in the RL agent. Most noticeably, the accuracy can be substantially augmented from 68% to 75% by the nearest neighbor design when tackling the Varol-17 dataset. Even if in some cases the improvement is not as significant as that in Varol-17, the similarity-driven nearest neighbor scheme can boost the action exploration in the RL agent and thus help with accuracy promotion. Furthermore, we demonstrate an incremental performance gain from adopting the proposed self-supervised learning mechanism. Across all different datasets, up to 0.7% improvement can be further achieved despite marginal increment observed in some datasets. Further investigation would be required to better leverage large quantities of unlabeled data and enhance the generalization ability of the self-supervised classifiers in different scenarios. This is beyond the core scope of this article and will be left for future study.

5.3 Effectiveness of the RL Mechanism

5.3.1 Parameter Searching Result by the RL Agent. To conduct an in-depth investigation into the effectiveness of the RL search mechanism, we randomly select 20 labeled nodes from the Cresci-15 dataset as the target users and examine the accuracy when adopting some given circumstances of GNN layers. For example, we create three types of GNN models by manually stacking one layer of GCN, two layers of GCN, and three layers of GCN, respectively. We feed the same graph data used in the previous testing into the three models and train 100 times. The trained models are used for validating if the RL can obtain the most proper number of layers for the selected nodes. For each node in the 20 target users, we count the ratio of being correctly classified out of the 100 runs. The main purpose of this investigation is to examine if the RL-based mechanism can pick up the layer with the highest classification ratio, to automatically enable the best model performance.

Figure 4 shows the ratio obtained for each target node when using different layers. Observably, different GCN layers have a varying impact on the correct prediction of a certain node. For example, the prediction effectiveness of other nodes (e.g., node index 2, 4, 5, 11, 12, 13, and 18) will be drastically affected by the number of GNN layers. We reason this phenomenon is because the range of GNNs' receptive field will gradually increase when the layer number ramps up; meanwhile, higher-order neighbor information will be involved and aggregated, thereby having a direct impact on the detection accuracy. By contrast, some nodes (e.g., node index 6, 8, 9, 15, 16, and 19) can be better predicted no matter what information is aggregated from neighbors with different orders. It is therefore necessary to elaborately select the number of GNN layers for these nodes to increase the probability of correct prediction of these nodes.

We use underline, e.g., **0.92**, to mark the final decision made by the RL agent when searching the model layer. The proposed RL agent can select the optimal layer that can deliver the highest prediction ratio. There is only 10% (2 of 20 classification tasks) mismatch between the best layer option and the choice made by the RL agent. This indicates the proposed approach can effectively reduce the manual tuning while reaching the best effectiveness.

5.3.2 Impact of Different Search Ranges on the Effectiveness. We dive into the impact of parameter selection on the overall effectiveness and demonstrate the sensitivity to such parameter changes. To do so, we first fix the maximum searching bound of the number of search layers of

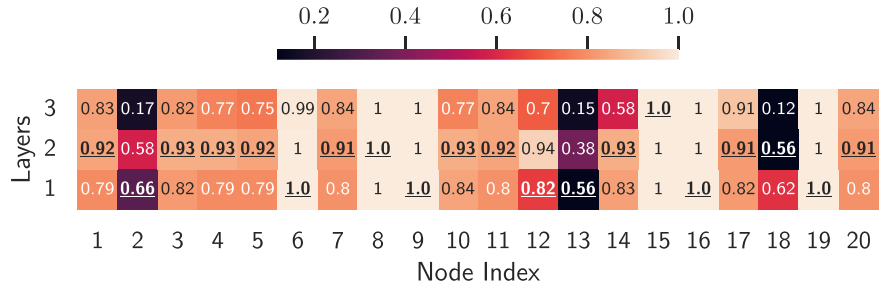
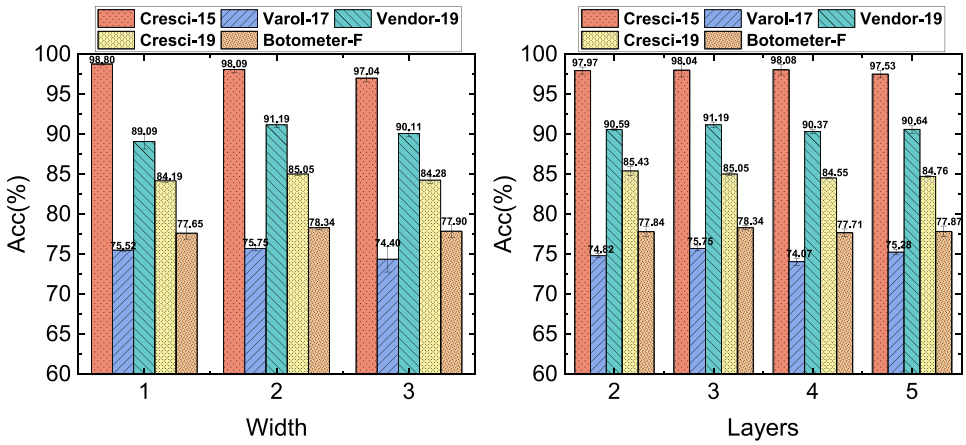


Fig. 4. Per-node prediction effectiveness when conducting architecture search. The underline represents the selected layer number by the RL-agent in the RoSGAS for each node.



(a) The upper bound of width of subgraph (b) The upper bound of the number of GNN layers.

Fig. 5. Impact of search upper bound of subgraph width and layer number on the effectiveness.

the graph neural network to be 3, while gradually increasing the searching bound for the width of the subgraph from 1 to 3. As shown in Figure 5(a), for all datasets, without exception, all the model instances experience a rise of accuracy when the searching range of width grows to 2 but a slight drop when the range is further extended to 3. This is because the increase of width range will lead to growing numbers of neighbors involved in the extracted subgraph. Particularly, the scale of the constructed graphs in the large-scale datasets is normally large and will lead to the explosion of neighbors, which in turn gives rise to the reduced quality of graph embedding and lower accuracy. This observation indicates that the search range for extracting subgraphs needs to be carefully modified and adaptive to the scale of a given scenario.

Given that the best result can be stably obtained by adopting $[1, 2]$ as the width range, we then fix this setting and vary the search range of the number of model layers. We gradually ramp up the upper bound of the range from 2 to 5. As shown in Figure 5(b), there is no significant disparities in the effectiveness among different options. The effectiveness is insensitive to the change of model layers despite some noticeable variations. For example, in the Cresci-15 dataset, the model accuracy will reach the peak when choosing 4 as the upper bound of the searching range, while for Vendor-19 the accuracy peak will come when searching up to three layers. Nevertheless, the discrepancy in accuracy is negligible, and the proposed RL-based searching mechanism can more

Table 4. The Average Time Consumption (Unit: Second) of Running Each Method 10 Times on the Datasets Cresci-15, Varol-17 (Varol), Vendor-19 (Vendor), Cresci-19, and Botometer-Feedback (Botometer-F)

Method	Cresci-15	Varol-17	Vendor-19	Cresci-19	Botometer-Feedback
GCN	572.53	115.04	323.36	18.78	10.21
GAT	576.72	118.51	331.66	21.75	11.64
SGC	296.38	198.62	331.43	108.10	39.97
GraphSAINT	604.13	167.11	460.13	55.98	12.56
Policy-GNN	2076.15	578.21	1734.41	120.34	109.21
ARMA	59.79	51.37	84.89	18.29	12.61
RoSGAS-*KL*	587.53	174.12	354.11	55.87	26.54
RoSGAS-*KL-NN*	612.32	212.13	382.11	356.42	29.22
RoSGAS	809.88	245.62	469.20	429.6	32.27

flexibly and adaptively make the best decision in ensuring the best model performance without incurring excessive cost in exploring additional GNN layers.

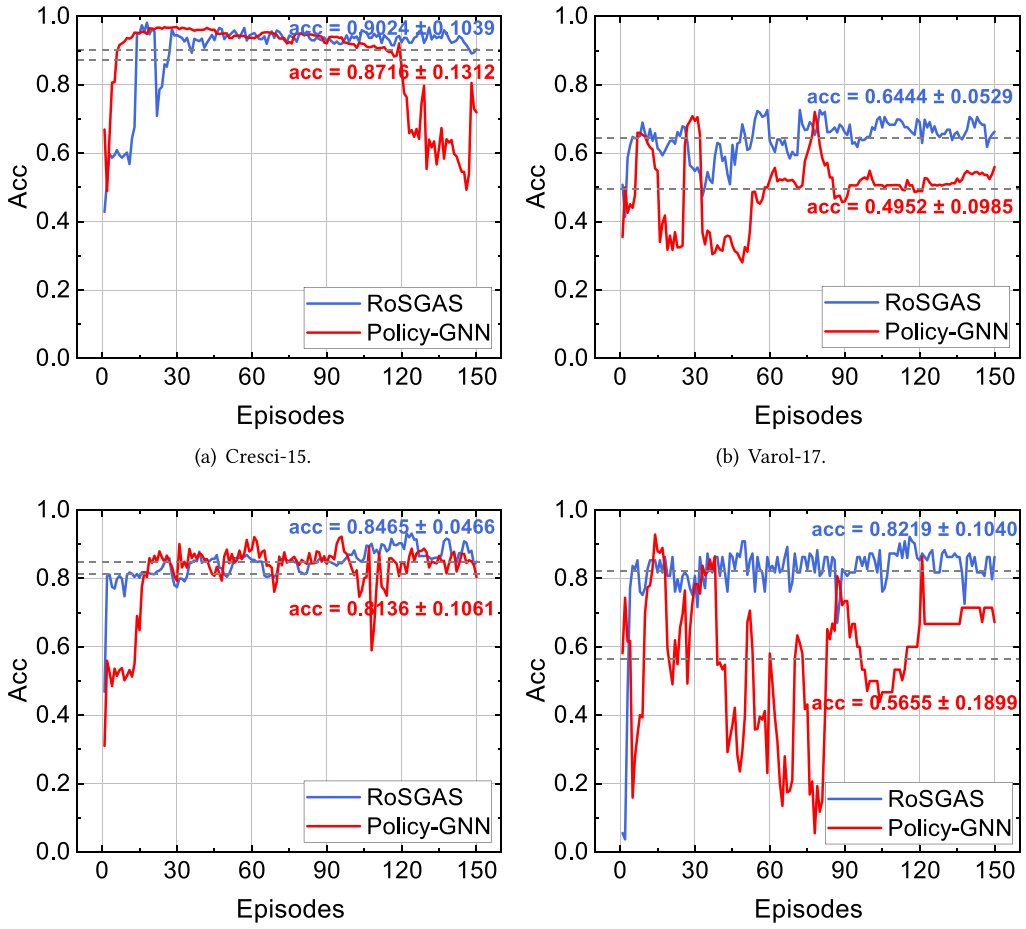
5.4 Efficiency

We primarily evaluate the efficiency by measuring the training time. As the reward signal needs to be obtained from the validation dataset to train the RL agent before constructing the GNN stack, we break down the time consumption into two parts—RL and GNN model training. It is worth noting that sparse matrix multiplication used in PyTorch Geometric can enable GNNs to be applied in very large graphs and accelerate the model training.

Table 4 presents the average time consumption of running each model 10 times. Overall, RoSGAS can strike a balance between the training time and the effectiveness. Although ARMA and SGC take less time to train their model due to the simplified GCN model through removing nonlinearity and collapsing weight matrices between consecutive layers, the achieved accuracy is far lower than RoSGAS across all datasets, particularly on Cresci-19, where the labels are scarcer. Policy-GNN takes the longest time for model training simply because the GNN stacking and convolution operation will be carried out for all nodes in the entire graph.

Most notably, the variant RoSGAS-*KL*, which does not include the nearest neighbor mechanism and the self-supervised learning mechanism, can achieve competitive training efficiency compared with GCN, GAT, and GraphSAINT, only with a marginal time increase. The slight difference is negligible when considering that RoSGAS-*KL* needs to train both the RL agent and the GNN model separately. Compared with RoSGAS-*KL*, RoSGAS-*KL-NN* intrinsically needs extra time to search the set of state-action pairs that have been explored and ascertain the nearest neighbor to the current state-action pair before modifying the expectation of the reward for optimized network parameters. RoSGAS additionally involves the self-supervised learning based on the target user batch to extract the homologous subgraphs for additional forward propagation.

In addition, the time consumption for Cresci-15 is far larger than that for Cresci-19. There is a linearly increased time consumption of GCN and GAT when the graph scale soars. In fact, the GNN training is more efficient since we only need to perform convolution operations on the sampled subgraph; by contrast, each iteration of GCN and GAT will have to perform a convolution operation on all nodes of an entire graph. RoSGAS is solely relevant to the extracted number of subgraphs for detecting the target users, and thus independent from the scale of the entire graph. This clearly showcases the inherent scalability and robustness of our sample and subgraph-based mechanism adopted in RoSGAS.



(a) Cresci-15. (b) Varol-17.

(c) Vendor-19. (d) Botometer-Feedback.

Fig. 6. The RL-agent training process of RoSGAS.

5.5 Stability

We also compare the stability of the RL-based architecture search between RoSGAS and Policy-GNN. Figure 6 demonstrates the detailed training process with the RL agent on Cresci-15, Varol-17, Vendor-19, and Botometer-Feedback, respectively. The dotted line represents the accuracy obtained in the validation set during the 150 episodes. Obviously, RoSGAS can promptly achieve high accuracy under all the datasets, and the mean accuracy can be achieved only after 15 RL agent training episodes. Meanwhile, once reaching this point, a stable state of Nash equilibrium can be steadily maintained without much turbulence.

By contrast, the accuracy of Policy-GNN is noticeably lower than RoSGAS and lacks stability; i.e., very obvious fluctuations manifest. The disparity mainly stems from the design of the state transition and the nearest neighbor mechanism. RoSGAS uses the embedding of the initial subgraph as the state input to the RL agent and jumps between the initial subgraphs as the state transition, while Policy-GNN uses the node embedding as the state input and jumps between the nodes as the state transition. Undoubtedly, the embedding of the initial subgraph as a state can better reflect the local structure of a targeted node, resulting in a stronger representation ability,

and hence enhanced stability. At the same time, during the RL agent training episodes, for a specific state-action pair of one transition, the nearest neighbor mechanism explores the existing pairs and exploits the reward pertaining to this nearest neighbor from the environment. The reward is used as a part of the label to optimize the Q-network, which greatly eliminates the difference between the target network prediction and the actual reward in the initial stage. As a result, the RL agents can achieve higher accuracy with only a few training episodes.

Interestingly, RoSGAS has different volatility across different datasets, and the magnitude of volatility is positively correlated to the average correct rate. This indicates that the node feature distribution and graph structure pertaining to each individual dataset have a non-trivial impact on the model training. The in-depth study will be left for future work.

5.6 Generalization

In the field of social bot detection, the continuous evolution of social bots' camouflage technology has brought generalization challenges to the model design. To identify the emerging attack methods of social bots and mine diverse user information, a robust detection model should have a high level of generalization. In this subsection, we examine the generalization of RoSGAS and compare with other baselines.

5.6.1 Out-of-sample Validation Accuracy. Apart from the in-sample validation in previous subsections, the most effective way to demonstrate the strong generalization is to perform out-of-sample validation, i.e., retaining some of the sample data for model training and then using the model to make predictions for unseen data, and examine the accuracy. To do so, we select one dataset as the training dataset and then use other datasets as the test dataset. We divide the training set into 10 equal parts, take one part of the training set for training each time, and train each baseline for 30 epochs. We use the trained models to predict the labeled accounts in the test datasets to calculate the accuracy.

A series of figures including Figures 7, 8, 9, 10, and 11 show the prediction results when the training is based upon Cresci-15, Varol-17, Vendor-19, Cresci-19, and Botometer-Feedback, respectively. It is obviously observable that RoSGAS outperforms all other baselines in the vast majority of scenarios. The only exceptions are when the model is trained based on Vendor-19 and validated on the Cresci-15 dataset and when the model is trained based on Cresci-15 while validating upon Vendor-19. In these two cases, the accuracy of RoSGAS is slightly lower than GCN. We reason this phenomenon is possibly because GCN is less sensitive to the disparity between two datasets. Unsurprisingly, all models have a reduced accuracy when conducting the out-of-sample prediction as opposed to its in-sample accuracy, simply because of the potential overfitting in the in-sample evaluations. The experiments carried out in this subsection generically showcase the robustness and generalization of our approach when handling new data where different noise manifests.

5.6.2 Stability. We can also observe the minimum performance fluctuation of RoSGAS, compared with other baselines, when different datasets are used as test sets. On the contrary, many other baselines such as GraphSAGE, GCN, and GraphSAINT have a noticeable fluctuation. For example, as demonstrated in Figure 7, when training on Cresci-15, the accuracy of GraphSAGE is only 23.21% and 26.63% when the trained model is tested based on Vendor-19 and Botometer-F. However, the accuracy can climb to 47.8% if tesing on Cresci-19. Likewise, as shown in Figure 8, while the accuracy of the GraphSAINT model trained upon Varol-17 and tested upon Cresci-15 is merely 41.34%, a competitive accuracy can be obtained when the model is tested on Vendor-19 (75.23%) or Botometer-F (74.12%).

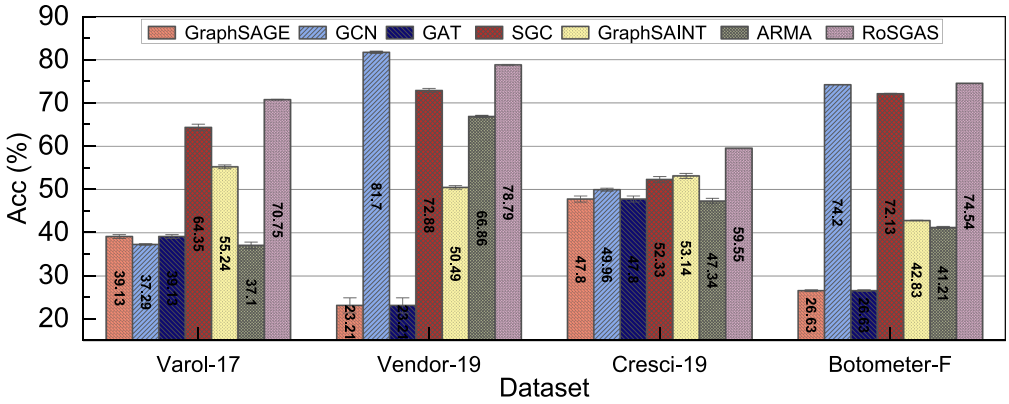


Fig. 7. Training on the Cresci-15 and testing on different datasets.

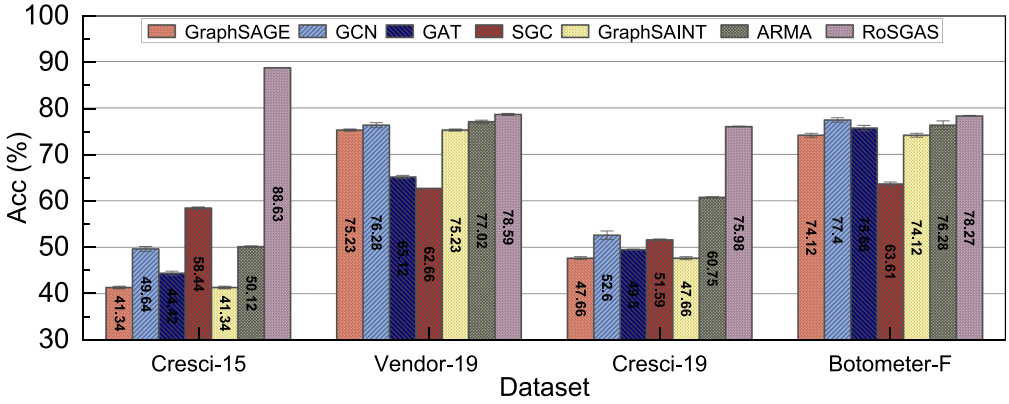


Fig. 8. Training on the Varol-17 and testing on different datasets.

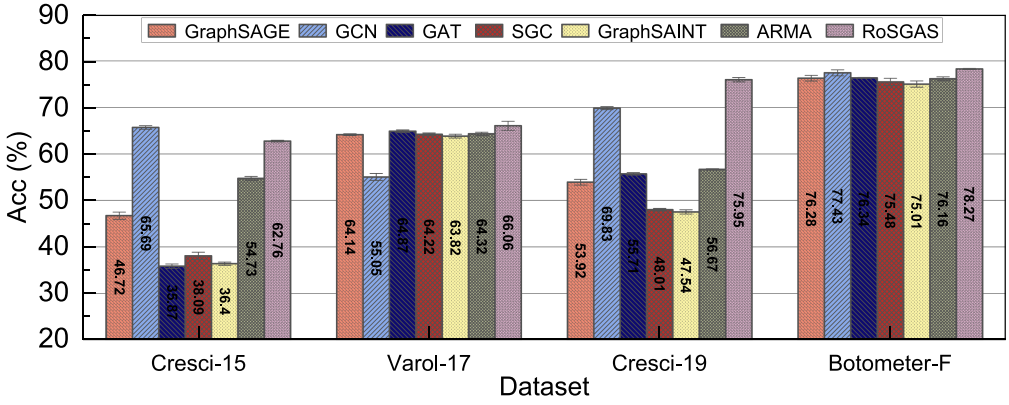


Fig. 9. Training on the Vendor-19 and testing on different datasets

The stable generalization stems from the adaptability and robustness of RoSGAS. In fact, our method only exploits some common features for the task of detecting social bots, without tightly coupling with or depending upon exclusive features or numerical characteristics. This will help maintain an outstanding quality of detection even when the testset varies.

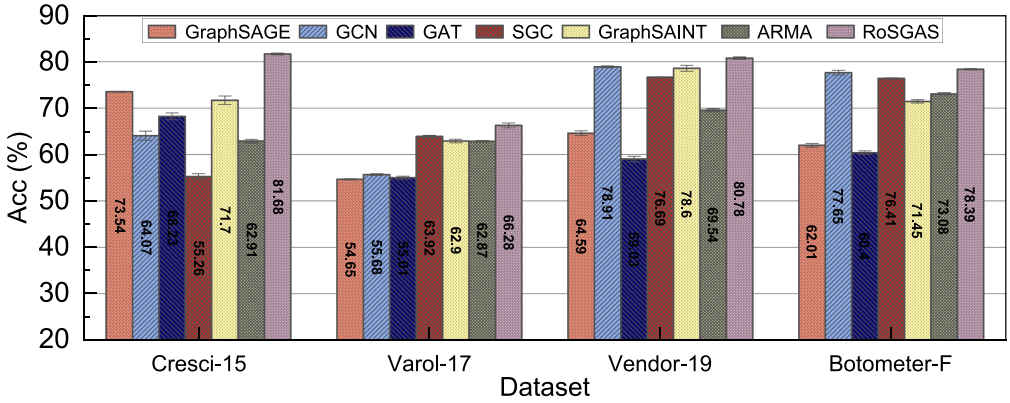


Fig. 10. Training on the Cresci-19 and testing on different datasets

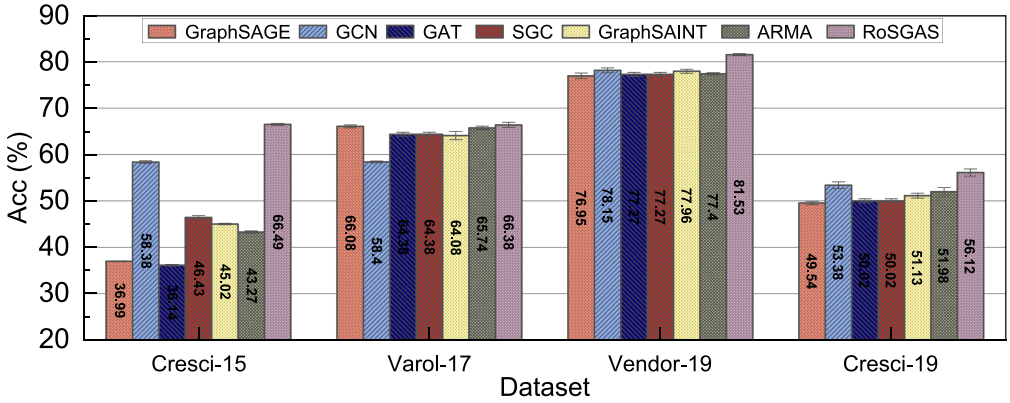
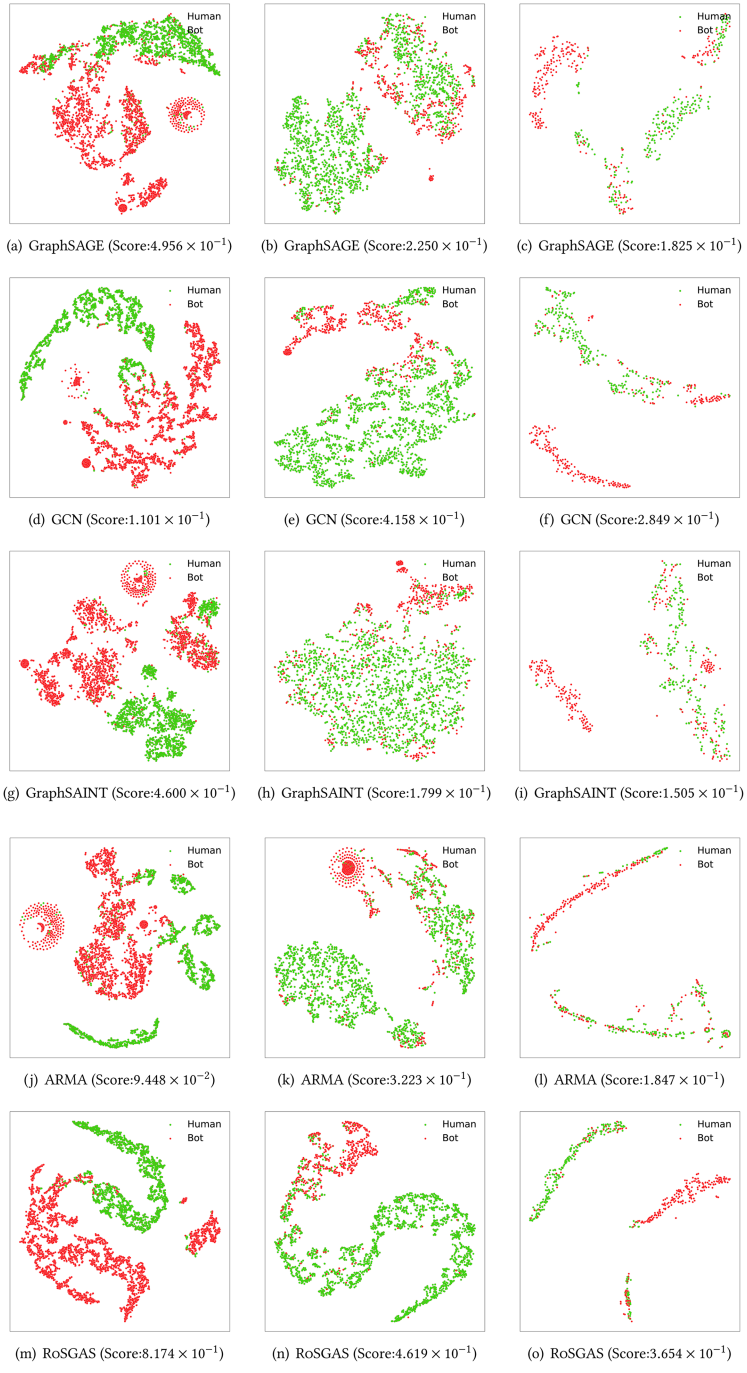


Fig. 11. Training on the Botometer-Feedback and testing on different datasets

5.7 Case Study: Effectiveness of Representation Learning

To further understand the quality of vector representation, we compare the representations of RoSGAS with the baselines that can achieve good results in the accuracy evaluation, i.e., GraphSAGE, GCN, GraphSAINT, and ARMA. For each individual model, we cluster the representation results by using k-means with $k = 2$ and then calculate the homogeneity score—a the-higher-the-better indicator that measures how much the samples in a cluster are similar. The homogeneity is satisfied—the value equals 1—if all of its clusters contain only data points that are members of a single class.

Figure 12 visualizes the result of t-SNE dimensionality reduction on the representation vectors of users. We evaluate each model based on three given datasets, Cresci-15, Vendor-19, and Cresci-19, respectively. For each baseline model, the results on the three datasets are placed in a row. Numerically, the homogeneity score of RoSGAS is higher than other baselines when adopting all datasets. For example, the score of RoSGAS is over 2 times higher than that of GraphSAGE on average, indicating that most distinguishable represent vectors can be obtained by our approach. The visualization completely aligns with the measurement. Observably, the bots in RoSGAS can be far more easily differentiated from the benign users.



(a) GraphSAGE (Score:4.956×10^{-1}) (b) GraphSAGE (Score:2.250×10^{-1}) (c) GraphSAGE (Score:1.825×10^{-1})

(d) GCN (Score:1.101×10^{-1}) (e) GCN (Score:4.158×10^{-1}) (f) GCN (Score:2.849×10^{-1})

(g) GraphSAINT (Score:4.600×10^{-1}) (h) GraphSAINT (Score:1.799×10^{-1}) (i) GraphSAINT (Score:1.505×10^{-1})

(j) ARMA (Score:9.448×10^{-2}) (k) ARMA (Score:3.223×10^{-1}) (l) ARMA (Score:1.847×10^{-1})

(m) RoSGAS (Score:8.174×10^{-1}) (n) RoSGAS (Score:4.619×10^{-1}) (o) RoSGAS (Score:3.654×10^{-1})

Fig. 12. 2D t-SNE plot of representation vectors of users produced by GraphSAGE, GCN, GraphSAINT ARMA, and RoSGAS on the dataset of Cresci-15 (left column), Vendor-19 (middle column), and Cresci-19 (right column), respectively. The corresponding homogeneity score is given in the bracket.

6 DISCUSSION

Significance of subgraph-based and RL-guided solution. This work is to advance the development and application of GNNs in the field of social bot detection. The performance of features-based, statistics-based, and deep-learning-based methods fades in the face of evolving bot technologies. We applied the GNN to leverage the information of the neighborhood and relationship to counter the development of bots. This work is non-trivial when tackling massive datasets with hundreds of thousands or even millions of nodes. The existing solutions to complex model architecture search are not well suited for the problem of social bot detection at scale under study—the data distribution becomes more non-IID due to the evolution of the bots, which complicates the model design with good generalization in practice. The extremely large scale of the social network graph also leads to tremendously different user structures and necessitates adaptive detection of such structures with reasonable computational costs. To this end, we proposed to search subgraphs to realize the reduction of graph scale, the simplification of model architectures, and the improvements of the detection performance.

Necessity of using RL. The width (k) of the subgraph cannot be a common hyperparameter shared by all nodes and requires node-by-node personalization. In fact, social bots tend to randomly follow benign accounts as a disguise. The selection of k primarily derives from our behavioral studies: There are noticeably enough bots in the subgraph within second-order subgraph, while benign nodes could be well recognized in the first-order subgraph. Increasing the value would not incur additional performance gain but involve an overwhelming number of neighbors. Nevertheless, the range can be flexibly configured to adapt to any other scenarios and datasets. This necessitates the adoption of RL to facilitate the customized parameter search for each individual subgraph.

Dealing with real-time streaming scenarios. Critical challenges for on-the-fly bot detection encompass the need for message delivery, distributed event store, and capability of handling the revolution and uncertainty of events and entities in the social network platform over time. This is particularly intricate due to the presence of new bots. The accuracy of offline learning models highly relies on the quality and quantity of the dataset that is fed into the models. However, this will be time-consuming and costly in real-time scenarios, and the model update is required to maintain the high standard of model accuracy. In practice, to implement the streaming pipeline, a distributed crawler needs to be developed to continuously fetch social network information. The collected data is then forwarded to processing modules through distributed event streaming such as Kafka. More online and incremental designs are desired to underpin the on-the-fly version of the current bot detection framework.

7 RELATED WORK

In this section, we summarize the related literature and state-of-the-art approaches. The existing literature can be roughly classified into three categories: GNN-based approaches, subgraph- and RL-based approaches, and self-supervised enhanced approaches.

7.1 GNN-based Social Bot Detection

Early social bot detection mainly focuses on manually analyzing the collected data to find discriminative features that can be used to detect the social bot. However, the detection features are easy to imitate and escaped by social bots, which are constantly evolving, and eventually become invalid. The recent boom of GNNs has promoted the latest progress in social bot detection. The first attempt to use GNNs to detect social bots [2] is mainly by combining the graph convolutional neural network with multilayer perception and belief propagation. The heterogeneous graph is constructed

and original node features are extracted by the pre-trained language model to get the final node embedding after aggregating by the R-GCN model [17], and the BotRGCN successfully surpasses the performance of traditional detection methods on the newly released dataset called TwiBot-20 [16]. The heterogeneous graph is also applied in [15]; it also proposes a relational graph transformer inspired from natural language processing to model the influence between users and learn node representation for better social bot detection, and its performance exceeds the BotRGCN. However, these node-embedding-based and node-classification-based methods perform convolution at the level of the entire graph; stacking multiple layers of GNN on the tremendous scale of social graphs will cause the over-smoothing problem [30, 53]. Subgraphs provide a new perspective to solve this issue; [31] constructs a heterogeneous graph and reconstructs subgraphs based on manually defined heuristic rules for detecting malicious accounts in online platforms. However, this kind of manual-based method not only consumes energy in setting the extraction rules but also cannot be easily generalized to the field of social bot detection.

7.2 Subgraph and RL-based Approaches

The scale of the graph constructed from the social network is tremendous. Performing convolution operation on the entire graph will not only consume computing resources but also lead to performance degradation due to problems such as over-smoothing. The extraction of subgraphs either requires domain-specific expert knowledge to set the heuristic rules [31, 58] or needs to design motifs for matching subgraphs [35, 54], which drastically limits the flexibility and capability of generalization. To address this issue, we leverage RL to adaptively extract subgraphs. There have been a few attempts to marry RL and GNNs. DeepPath [52] establishes a knowledge graph embedding and reasoning framework that utilizes the RL agent to ascertain the reasoning paths in the knowledge base. RL-HGNN [62] devises different meta-paths for any node in an HIN for adaptively selecting meta-path to learn its effective representations. CARE-GNN [14], RioGNN [37], RTGNN [61], and FinEvent [38] all marry the RL and GNNs for dynamically optimizing the similarity threshold to achieve the purpose of selecting more valuable neighbor nodes for the aggregated nodes, to obtain more effective representation vectors for fraud detection or event detection. Policy-GNN [27] utilizes RL to select the number of GNN architecture layers for aggregating the node embedding vectors to classify nodes. GraphNAS [20] is among the first attempts to combine the recurrent neural network with GNNs. It continuously generates descriptions of GNN architecture to find the optimal network architecture based on RL by maximizing the expected accuracy. Similarly to the architecture search in GraphNAS, Auto-GNN [63] additionally proposes a parameter sharing mechanism for sharing the parameters in homogeneous architecture for reducing the computation cost. However, these methods are not combined with the subgraph method and their optimization is tightly coupled with specific datasets. They are not suited for detecting graphs that follow a power-law distribution with huge disparities among different users [4, 34]. By contrast, our approach relies on subgraph embedding to achieve high detection effectiveness without compromising time efficiency and has strong generalization across multiple datasets.

7.3 Self-supervised Learning Approaches on Graphs

In recent years, self-supervised learning has advanced as a promising approach to overcome the limited data annotation and to enable a target objective to be achieved without supervision. The technology has been investigated in a wide range of domains, such as natural language processing [11, 60], computer vision [28, 46], and graph analysis [22, 25]. At the core of self-supervised learning is to define an annotation-free pretext task to train an encoder for representation learning. Particularly for graph analysis, there are a few works of literature about designing self-supervised tasks based on either edge attributes [10, 44] or node attributes [12]. However, the

inherent dependencies among different nodes in the topology hinder the appropriate design of the pretext tasks. DGI [49] trains a node encoder to maximize the mutual information between the node representations and the global graph representation. GMI [39] defines a pretext task that maximizes the mutual information between the hidden representation of each node and the original features of its one-hop neighbors. InfoGraph [43] maximizes the mutual information between the graph embeddings and the substructure embeddings at different scales to more effectively learn graph embeddings. However, the aforementioned self-supervised learning approaches need to take the holistic graph as the input, which is time- and resource-consuming and thus restricts the scalability on large-scale graphs. Our approach aims to obtain a subgraph-level representation to ensure non-homologous subgraphs are discriminative while homologous subgraphs have similar representation vectors.

8 CONCLUSION

This article studies an RL-enabled framework for GNN architecture search. The proposed RoSGAS framework can adaptively ascertain the most suitable multi-hop neighborhood and the number of layers in the GNN architecture when performing the subgraph embedding for the social bot detection task. We exploit HIN to represent the user connectivity and use a multi-agent deep RL mechanism for steering the key parameter search. The subgraph embedding for a targeted user can be more effectively learned and used for the downstream classification with competitive accuracy while maintaining high computation efficiency. Experiments show that RoSGAS outperforms the state-of-the-art GNN models in terms of accuracy, training efficiency, and stability. RoSGAS can more quickly achieve, and carry on with, high accuracy during the RL training and has strong generalization and explainability. We believe the data-centric solution guided by behavioral characterization and reinforcement learning—instead of heavily complicating the network architecture itself—would be a promising and innovative direction, which is both scientifically and engineering-wise challenging in the field of bot detection. In the future, we plan to examine the impact of feature distribution and graph structure on model training and extend RoSGAS to underpin the streaming scenarios.

ACKNOWLEDGMENT

We thank anonymous reviewers for the provided helpful comments on earlier drafts of the manuscript.

REFERENCES

[1] Norah Abokhodair, Daisy Yoo, and David W. McDonald. 2015. Dissecting a social botnet: Growth, content and influence in Twitter. *CSCW*. 839–851.
[2] Seyed Ali Alhosseini, Raad Bin Tareaf, Pejman Najafi, and Christoph Meinel. 2019. Detect me if you can: Spam bot detection using inductive representation learning. *WWW*. 148–153.
[3] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. *NIPS* 33 (2020), 8017–8029.
[4] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.
[5] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 2021. Graph neural networks with convolutional ARMAfilters. *TPAMI* (2021).
[6] Adam Breuer, Roee Eilat, and Udi Weinsberg. 2020. Friend or faux: Graph-based early detection of fake accounts on social networks. *WWW*. 1287–1297.
[7] Stefano Cresci. 2020. A decade of social bot detection. *Commun. ACM* 63, 10 (2020), 72–83.
[8] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. 2015. Fame for sale: Efficient detection of fake Twitter followers. *Decis. Support Syst.* 80 (2015), 56–71.

[9] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. 2017. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. *WWW*. 963–972.
[10] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2018. Adversarial network embedding. In *AAAI*, Vol. 32.
[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL*. 4171–4186.
[12] Ming Ding, Jie Tang, and Jie Zhang. 2018. Semi-supervised learning on graphs with generative adversarial nets. *CIKM*. 913–922.
[13] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. *KDD*. 135–144.
[14] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. 2020. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. *CIKM*. 315–324.
[15] Shangbin Feng, Zhaoxuan Tan, Rui Li, and Minnan Luo. 2022. Heterogeneity-aware Twitter bot detection with relational graph transformers. *AAAI*.
[16] Shangbin Feng, Herun Wan, Ningnan Wang, Jundong Li, and Minnan Luo. 2021. TwiBot-20: A comprehensive Twitter bot detection benchmark. *CIKM*. 4485–4494.
[17] Shangbin Feng, Herun Wan, Ningnan Wang, and Minnan Luo. 2021. 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 (ASONAM'21)*. 236–239.
[18] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. 2016. The rise of social bots. *Commun. ACM* 59, 7 (2016), 96–104.
[19] Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch geometric. *ICLR Workshop*.
[20] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph neural architecture search. *IJCAI*, Vol. 20. 1403–1409.
[21] Zafar Gilani, Ekaterina Kochmar, and Jon Crowcroft. 2017. Classification of Twitter accounts into automated agents and human users. *ASONAM*. 489–496.
[22] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. *KDD*. 855–864.
[23] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NIPS*. 1025–1035.
[24] Yiming Hei, Renyu Yang, Hao Peng, Lihong Wang, Xiaolin Xu, Jianwei Liu, Hong Liu, Jie Xu, and Lichao Sun. 2021. Hawk: Rapid android malware detection through heterogeneous graph attention networks. *TNNLS* (2021), 1–15.
[25] Thomas N. Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[26] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR*.
[27] Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu. 2020. Policy-GNN: Aggregation optimization for graph neural networks. *KDD*. 461–471.
[28] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. 2016. Learning representations for automatic colorization. *ECCV*. Springer, 577–593.
[29] John Boaz Lee, Ryan A. Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. 2019. Graph convolutional networks with motif-based attention. *CIKM*. 499–508.
[30] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. *32nd AAAI Conference on Artificial Intelligence*.
[31] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. *CIKM*. 2077–2085.
[32] Michele Mazza, Stefano Cresci, Marco Avvenuti, Walter Quattrociocchi, and Maurizio Tesconi. 2019. Rtbust: Exploiting temporal patterns for botnet detection on Twitter. *WebSci*. 183–192.
[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[34] Lev Muchnik, Sen Pei, Lucas C. Parra, Saulo D. S. Reis, José S. Andrade Jr., Shlomo Havlin, and Hernán A. Makse. 2013. Origins of power-law degree distribution in the heterogeneity of human activity in social networks. *Scientific Reports* 3, 1 (2013), 1–8.
[35] Hao Peng, Jianxin Li, Qiran Gong, Yuanxin Ning, Senzhang Wang, and Lifang He. 2020. Motif-matching based subgraph-level attentional convolutional network for graph classification. *AAAI*, Vol. 34. 5387–5394.
[36] Hao Peng, Renyu Yang, Zheng Wang, Jianxin Li, Lifang He, Philip Yu, Albert Zomaya, and Raj Ranjan. 2021. Lime: Low-cost incremental learning for dynamic heterogeneous information networks. *IEEE Trans. Comput.* (2021), 628–642.
[37] Hao Peng, Ruitong Zhang, Yingtong Dou, Renyu Yang, Jingyi Zhang, and Philip S. Yu. 2021. Reinforced neighborhood selection guided multi-relational graph neural networks. *ACM Trans. Inf. Syst.* 40, 4, Article 69 (Dec. 2021), 46 pages.

[38] Hao Peng, Ruitong Zhang, Shaoning Li, Yuwei Cao, Shirui Pan, and Philip S. Yu. 2023. Reinforced, incremental and cross-lingual event detection from social messages. *TPAMI* 45, 1 (2023), 980–998.
[39] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. *WWW*. 259–270.
[40] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.
[41] Junhong Shen and Lin F. Yang. 2021. Theoretically principled deep RL acceleration via nearest neighbor function approximation. *AAAI*, Vol. 35. 9558–9566.
[42] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S. Yu Philip. 2016. A survey of heterogeneous information network analysis. *TKDE* 29, 1 (2016), 17–37.
[43] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2019. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. arXiv preprint arXiv:1908.01000 (2019).
[44] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. *WWW*. 1067–1077.
[45] Julian R. Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.
[46] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. 2016. Conditional image generation with PixelCNN decoders. *NIPS*. 4790–4798.
[47] Onur Varol, Emilio Ferrara, Clayton Davis, Filippo Menczer, and Alessandro Flammini. 2017. Online human-bot interactions: Detection, estimation, and characterization. In *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM'17)*, Vol. 11.
[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR*.
[49] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep graph infomax. *ICLR (Poster)* 2, 3 (2019), 4.
[50] Jianyu Wang, Rui Wen, Chunming Wu, Yu Huang, and Jian Xion. 2019. Fdgars: Fraudster detection via graph convolutional networks in online app review system. *WWW*. 310–316.
[51] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. *ICML*. 6861–6871.
[52] Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *EMNLP*. 564–573.
[53] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *ICML*. PMLR, 5453–5462.
[54] Carl Yang, Mengxiong Liu, Vincent W. Zheng, and Jiawei Han. 2018. Node, motif and subgraph: Leveraging network functional blocks through structural convolution. *ASONAM*. IEEE, 47–52.
[55] Kai-Cheng Yang, Onur Varol, Clayton A. Davis, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. 2019. Arming the public with artificial intelligence to counter social bots. *Comput. Hum. Behav.* 1, 1 (2019), 48–61.
[56] Kai-Cheng Yang, Onur Varol, Pik-Mai Hui, and Filippo Menczer. 2020. Scalable and generalizable social bot detection through data selection. *AAAI*, Vol. 34. 1096–1103.
[57] Xiaoyu Yang, Yuefei Lyu, Tian Tian, Yifei Liu, Yudong Liu, and Xi Zhang. 2020. Rumor detection on social media with graph structured adversarial learning. *IJCAI*. 1417–1423.
[58] Zihao Yuan, Qi Yuan, and Jiajing Wu. 2020. Phishing detection on ethereum via learning representation of transaction subgraphs. *BlockSys*. Springer, 178–191.
[59] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. Graphsaint: Graph sampling based inductive learning method. *ICLR*.
[60] Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization. *ACL*. 5059–5069.
[61] Xusheng Zhao, Qiong Dai, Jia Wu, Hao Peng, Mingsheng Liu, Xu Bai, Jianlong Tang, and Philip S. Yu. 2022. Multi-view tensor graph neural networks through reinforced aggregation. *TKDE* (2022), 1–14.
[62] Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. 2020. Reinforcement learning enhanced heterogeneous graph neural network. *arXiv preprint arXiv:2010.13735* (2020).
[63] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. 2019. Auto-GNN: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184* (2019).

Received 25 January 2022; revised 14 June 2022; accepted 20 October 2022