

Article

Dual Graph Neural Networks for Social Bot Detection: Social Relation with an Auxiliary Behavior Similarity Graph

Guohui Song ^{1,2,*}, Shirun Wang ¹, Qi Wang ^{1,2}, Lin Wu ^{1,2} and Fan Liu ³¹ School of Computer and Cyber Sciences, Communication University of China, Beijing 100024, China² State Key Laboratory of Media Convergence and Communication, Communication University of China, Beijing 100024, China³ School of Journalism, Communication University of China, Beijing 100024, China

* Correspondence: sgh3352952@cuc.edu.cn

Highlights

Please indicate how your work links to systems science via your contributions to systems practice, theory, and or methodology.

- A multi-perspective graph model is designed for social bot detection that fuses relation and behavioral views of social networks, and the resulting detector can prospectively serve as a discriminative component in agent social network generation and multi-agent propagation analysis.
- The dual-graph design observes a social platform through two complementary descriptions: explicit social ties and latent behavioral affinity. It then examines how much each description contributes to identifying automated accounts in the system.

What are the main findings and/or the implications of the main findings?

- A dual-branch graph neural network architecture is proposed for social bot detection, explicitly coupling a heterogeneous social relation graph (RGCN) with a separately constructed behavior similarity graph (GAT); the behavior branch is positioned as a sparse, conditionally active auxiliary view rather than a co-equal branch.
- The model reaches $89.48 \pm 0.07\%$ accuracy and $78.11 \pm 0.14\%$ macro-F1 over five seeds at per-seed validation-calibrated thresholds. Ablations show that the text embedding contributes most, the relation branch supplies most graph signal, and the behavior branch's global gain is non-significant—though it is active on covered users ($2.9\times$ more prediction changes, $4.0\times$ larger probability shifts), where a validation-only subgroup threshold lifts macro-F1 from 51.85 to 67.17 without retraining.

Abstract

Social bots threaten the integrity of online social platforms by spreading misinformation, manipulating public opinion, and carrying out fraudulent activities. Detecting social bots is therefore not just a classification problem: looking ahead, a detector can also act as a discriminative component in agent-based social network generation, supplying realism-assessment signals for a “Generate–Detect–Explain–Refine” loop. Existing graph-based detection methods rely mainly on social relation graphs and do not explicitly treat behavioral similarity as a separate parallel graph branch. This paper proposes a dual-branch graph neural network for Twitter bot detection in which a heterogeneous social relation graph is accompanied by a sparse, conditionally active behavior-similarity auxiliary view; the paper's main contributions are a competitive detector and a diagnostic protocol for attributing and repairing the value of that auxiliary view. The relation branch uses a

Academic Editor: Vladimír Bureš

Received: 11 August 2026

Revised: 9 October 2026

Accepted: 19 October 2026

Published: 20 October 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\) license](https://creativecommons.org/licenses/by/4.0/).

Relational Graph Convolution Network to process a heterogeneous social relation graph, while the auxiliary branch applies a Graph Attention Network to a behavior similarity graph that links users with similar follower–following ratios, tweet type distributions, and post influence metrics. Experiments on TwiBot-22 show that the proposed model reaches $89.48 \pm 0.07\%$ accuracy and $78.11 \pm 0.14\%$ macro-F1 over five random seeds at a per-seed validation-calibrated decision threshold ($89.92 \pm 0.07\%$ accuracy and $77.28 \pm 0.31\%$ macro-F1 at the default 0.5 threshold). Per-class and threshold analyses further show that the default operating point is precision-oriented on the minority bot class (bot recall $54.83 \pm 1.60\%$), while operating near the validation-calibrated threshold ($\tau = 0.40$) raises bot recall to $64.12 \pm 1.31\%$. Ablation studies show that the semantic text embedding provides the largest single performance contribution and that the relation branch supplies the bulk of the graph signal. A degree-stratified analysis further shows that the behavior branch is functionally active on its $\approx 1\%$ covered users— $2.9\times$ more prediction changes and $4.0\times$ larger probability shifts than on uncovered nodes—where a validation-only subgroup threshold lifts macro-F1 from 51.85 to 67.17 without retraining, so that it can be turned into a deployable calibrated component.

Keywords: social bot; social network; neural network; disinformation

1. Introduction

Online social networks such as Twitter (now X), Facebook, and Weibo have become indispensable platforms for information dissemination, public discourse, and social interaction. However, the openness and scalability of these platforms also make them vulnerable to abuse by automated accounts, commonly known as social bots. Social bots are software-driven accounts programmed to perform various tasks, ranging from benign customer service to malicious activities such as spreading fake news, manipulating trending topics, astroturfing public opinion, and executing phishing attacks [1,2]. Studies have shown that social bots played significant roles in political events and public health crises [3,4], highlighting the urgency of effective bot detection mechanisms.

Traditional bot detection methods have primarily relied on hand-crafted features derived from user profiles, temporal activity patterns, and content analysis [5]. With the rise of deep learning, researchers have increasingly turned to neural network-based approaches, including graph neural networks (GNNs) for leveraging social network structures [6,7].

From a systems perspective, social platforms are complex systems in which users interact through visible social ties such as friendships and follow relationships, as well as through hidden behavioral connections. The same logic applies to malicious bots. A single bot account may be indistinguishable from a genuine user, yet a group of bots with similar behavioral patterns can exhibit obvious collective patterns when viewed together. This suggests that bot detection should go beyond simply asking whether one account is a bot. It should be treated as a multi-level observation task that looks at both the visible network structure and the hidden behavioral patterns behind campaigns of bots.

From a broader perspective, the present work is not only a static node-level classifier: it is also designed to serve as a component of a dynamic generation loop. Recent work has repurposed social bot detectors as active discriminators within a “Generate–Detect–Explain–Refine” paradigm for agentic social network generation, where a detector evaluates the realism of networks produced by LLM-driven agents and feeds structured evidence back to the generator [8]. The proposed dual-graph model is a candidate component for this role: the RGCN branch is sensitive to relational anomalies in a generated social graph,

and the GAT branch responds to anomalous behavioral affinity among accounts that are not socially connected. Evaluating whether the detector's outputs provide useful, structured feedback for refining multi-agent-generated social networks is a prospective application beyond the scope of this paper; what this paper demonstrates is the detector's classification performance and a mechanistic analysis of its branches.

Despite these advances, most existing graph-based approaches rely on a single observable topology—typically the social relation graph—and therefore lack the second observational lens needed to detect the latent behavioral couplings that underpin emergent coordination. They may overlook important behavioral similarities among bot accounts. Social bots often exhibit similar behavioral patterns, such as comparable follow–follower ratios, consistent tweet type distributions, and analogous influence metrics, even when they are not directly connected in the social network [9]. Capturing these behavioral affinities through a dedicated behavior similarity graph can provide complementary information that is not available in the social relation graph alone.

Motivated by these observations, we propose a dual graph neural network framework for social bot detection. Beyond classification, the proposed detector is further intended to provide discriminative signals for agentic social network generation and information manipulation analysis. The key contributions of this work are as follows:

1. A controlled dual-graph testbed. A dual-branch parallel graph neural network is proposed for social bot detection, coupling a heterogeneous social relation graph (RGCN) with a separately constructed auxiliary behavior similarity graph (GAT); the architecture serves as a controlled testbed for attributing graph signal to each view.
2. Multi-modal feature integration. We combine multiple numerical and categorical features with 1536-dimensional semantic embeddings extracted from user descriptions and tweets, enabling the model to exploit both structured profile information and unstructured textual content.
3. An attribution result with a deployable remedy. Under this protocol we show that the semantic text embedding is the largest single contributor, the relation branch supplies the bulk of the graph signal, and the auxiliary behavior branch's global marginal contribution is statistically undetectable at $\approx 1\%$ coverage. At the same time, the branch is functionally active exactly on covered users, where a validation-only subgroup threshold lifts macro-F1 from 51.85 to 67.17 without retraining—converting an operating-point artifact into a calibrated, deployable component.

In summary, the contribution of this paper is twofold: (i) a competitive detector whose main gains come from multi-modal node features and the relation branch; (ii) a diagnostic protocol that localizes and repairs the conditional value of an auxiliary graph view.

The remainder of this paper is organized as follows. Section 2 reviews related work on social bot detection. Section 3 details the proposed methodology, including data pre-processing, feature engineering, graph construction, model architecture, and training procedures. Section 4 presents the experimental setup and results. Section 5 discusses the implications and limitations of our findings. Section 6 concludes the paper.

2. Related Works

2.1. Feature-Based Bot Detection

Early social bot detection methods relied heavily on hand-crafted features extracted from user profiles, temporal activity patterns, and tweet content. Yang et al. proposed a scalable and generalizable framework that uses minimal account metadata, enabling real-time analysis of large Twitter streams. Their work highlighted the importance of data selection for improving model generalization across different datasets [5]. Zhang et al. explored behavioral feature analysis for social bot identification, demonstrating that

carefully selected behavioral indicators can significantly improve model accuracy and stability [2].

These feature-based methods are typically interpretable and efficient, making them suitable for real-time detection systems. However, they often struggle to generalize across datasets and time periods, as bot operators can easily manipulate profile features to evade detection. The adversarial nature of bot detection has motivated researchers to incorporate more robust signals, such as network structure and textual content.

2.2. Text-Based and Deep Learning Approaches

Text-based methods leverage the content generated by users to distinguish bots from humans. Dukić et al. introduced a BERT-based model for Twitter bot detection, demonstrating the value of contextualized text representations [10]. Arin and Kutlu proposed a deep-learning-based approach for Twitter bot detection, demonstrating the effectiveness of neural text modeling for capturing account behavioral patterns [11].

The emergence of LLMs has further transformed this landscape. Modern bots can generate human-like text, making content-based detection more challenging [4]. At the same time, pretrained language models such as BERT [12] and BERTweet [13] provide powerful tools for encoding user descriptions and tweets into dense semantic representations that can be fused with other feature modalities.

Despite their representational power, text-based and deep learning approaches have notable limitations: they model accounts in isolation without exploiting social network structure, rely on a single modality and discard topology or behavioral coordination signals that are hard to fake, and are increasingly vulnerable to LLM-generated human-like content.

2.3. Graph-Based Bot Detection

Graph neural networks have emerged as a powerful paradigm for social bot detection by leveraging the relational structure of social networks. Tan et al. proposed BotPercent, a community-level bot detection framework that uses graph-based models with confidence calibration to estimate bot populations within specific Twitter communities [7]. Wu et al. proposed BotSCL, a heterophily aware contrastive learning framework that differentiates between representations of heterophilous neighbors while aligning homophilous ones, addressing the feature blending problem in graph-based bot detection [14].

Zeng et al. proposed MRLBot, an unsupervised representation learning framework that uses Transformer and CNN encoder–decoder to extract global and local behavioral features, combined with community-aware random walks for network representation learning [15]. Guyan et al. proposed PEGNN, a peripheral-enhanced graph neural network that captures distant topological features beyond immediate neighbors to improve bot detection [16]. Wan et al. learn a homophilous context graph that captures implicit interactions beyond friend relations, further confirming that a richer graph topology improves bot detection [17].

Graph-based methods are particularly effective at detecting bots that are embedded in social networks, as they can propagate information through follower/following relationships. However, they share a key limitation: most rely on a single social relation graph and therefore overlook behavioral affinities among coordinated accounts that are not directly connected.

2.4. Multi-modal Fusion Methods

The integration of multiple modalities has become increasingly important in social bot detection. Arranz-Escudero et al. proposed a multi-modal framework for bot detection on Twitter that combines user profile features, text analysis, and graph-based techniques [18]. Zhou et al. proposed LGB, a framework that combines language models and GNNs for social bot detection, showing improvements in detecting sparsely linked nodes [19].

Wang et al. proposed HHG-Bot, a hyper-heterogeneous graph-based framework that integrates heterogeneous graph convolutional networks with a trainable hypergraph aggregation model to capture high-order interactions among users and tweets, employing a meta-learning paradigm to enhance generalization across different bot types [20]. Yang et al. [4] introduced BotUmc, an uncertainty-aware framework that uses multi-view causal inference to improve detection confidence. These works highlight the importance of combining complementary information sources for robust bot detection. Yet most multi-modal approaches fuse only profile attributes and textual content while still propagating information over a single social relation graph; they rarely construct an explicit behavioral-affinity view. Consequently, accounts that are individually innocuous but jointly anomalous in their activity patterns remain hard to flag.

2.5. Behavior Analysis and Bot Behavioral Similarity

Behavioral analysis has received significant attention in social bot detection. Social bots often exhibit coordinated behavior patterns, such as synchronized posting, similar content themes, and comparable follower/following strategies [9]. Liu et al. proposed MB-HGTBot, which constructs a multi-type social behavior heterogeneous graph and combines RGCN, attention-based feature fusion, and supervised contrastive learning to model multi-dimensional heterogeneous relations across diverse social behavior contexts, enhancing robustness against camouflaged bots [21]. Meng et al. conducted a systematic review of social bot detection methods in communication research, highlighting persistent ambiguities and opportunities for methodological advancement [22]. Several studies have specifically focused on detecting coordinated bot campaigns. These methods often rely on temporal patterns, content similarity, or network motifs to identify groups of colluded accounts. However, they typically detect coordination post hoc as a separate post-processing step rather than embedding it directly into a learnable graph representation.

It should be noted that prior works such as HCGBot have indeed explored implicit relation graphs [17]. However, they mainly focus on enhancing homophilous neighborhood aggregation through graph structure learning, or serve as standalone graph construction modules. In contrast, the contribution of our work is not to propose a behavioral graph for the first time, but rather to explicitly treat the social relation graph (via RGCN) and the behavior similarity graph (via GAT) as two parallel and independently trainable branches, while fusing multi-modal semantics at the node feature level. This dual-branch parallel architecture is fundamentally different from single-graph improvements or coordination detection.

2.6. Attention Mechanisms in GNNs

Attention mechanisms have been widely adopted in GNNs to improve the interpretability and performance of social bot detection models. Veličković et al. introduced GAT, which enables nodes to attend differently to their neighbors, providing a flexible way to model heterogeneous neighborhood importance [23]. Huang et al. proposed a social bot detection framework based on heterogeneous graph attention and similarity graph features, demonstrating the complementary value of attention-based structural and behavioral modeling [6].

In our framework, GAT is applied to the behavior similarity graph because its attention mechanism can selectively emphasize edges that correspond to strong behavioral affinities, which is particularly useful when the behavior similarity graph contains noisy or weak connections.

In summary, existing methods each leave important gaps: feature- and text-based approaches model accounts in isolation and are vulnerable to LLM-generated content; graph-based detectors rely on a single social relation graph blind to non-adjacent behavioral affinity; multi-modal fusion still propagates over one graph without an explicit behavioral view; and coordination detection remains a post hoc, threshold-sensitive step rather than a learnable representation. Our work targets these gaps by explicitly constructing and fusing two distinct graph views—social relations and behavior similarity—within a dual-graph framework. Together they let structural, behavioral, and textual signals shape node embedding end to end.

3. Materials and Methods

The proposed model adopts a dual-graph architecture for social bot detection, as illustrated in Figure 1.

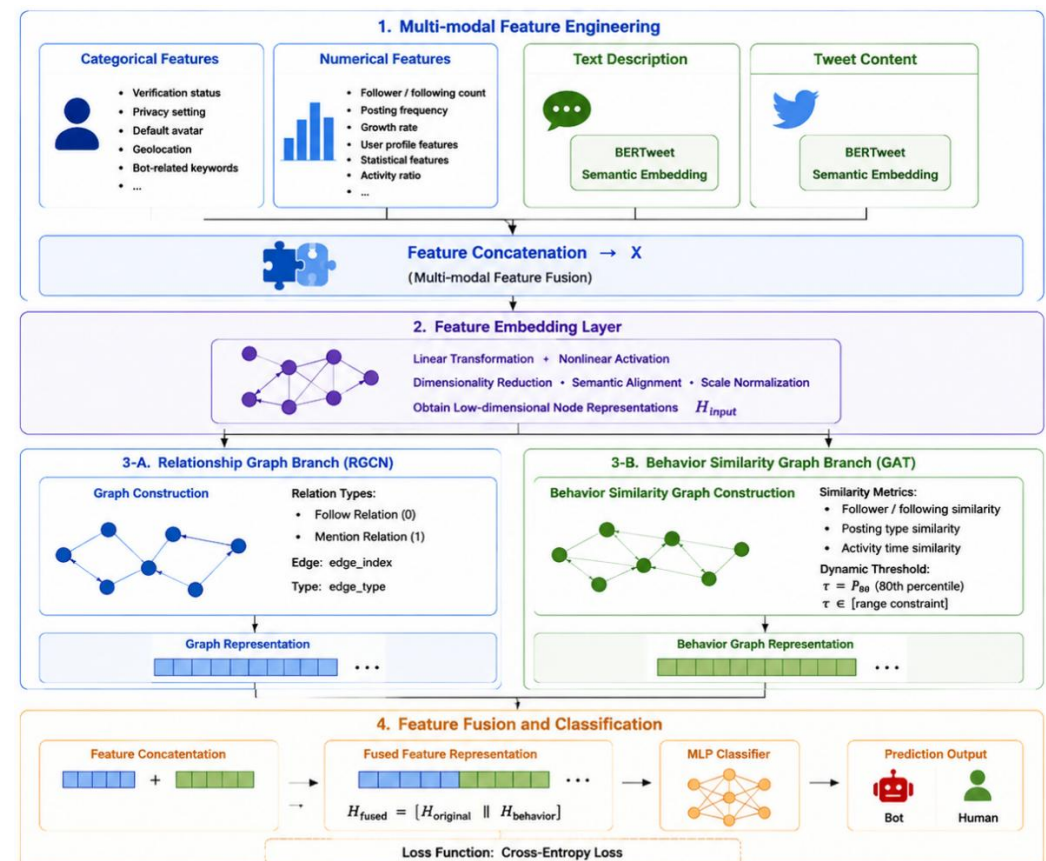


Figure 1. Architecture of the proposed dual-graph model.

The overall framework consists of four components:

1. Multi-modal feature engineering: Four types of user features are extracted and fused: categorical profile attributes, numerical activity statistics, and BERTweet-based semantic embedding from profile descriptions and recent tweets. These features are concatenated into a unified feature vector.

2. Feature embedding layer: The concatenated feature vector is projected into a lower-dimensional hidden space through a linear transformation followed by ReLU

activation and dropout, producing the initial node embedding that serves as the input for both graph branches.

3. Dual-graph learning: Two complementary graph representations are constructed and processed in parallel. A heterogeneous social relation graph capturing follower and following relationships is processed by RGCN, while a behavior similarity graph encoding behavioral affinities is processed by GAT.

4. Feature fusion and classification: The outputs of the two graph branches are concatenated and passed through a two-layer MLP classifier, which produces the final bot/human prediction through a class-weighted training objective and a validation-calibrated decision threshold.

3.1. Dataset

We evaluate our model on the TwiBot-22 dataset [24], a large-scale graph-based Twitter bot detection benchmark. TwiBot-22 contains approximately one million users, 86.7 million tweets, and 170 million social relationships, with 139,943 bot and 860,057 human labels. It models Twitter as a heterogeneous information network with four entity types (user, tweet, list, and hashtag) and fourteen relation types (e.g., follower, following, post, and mention), providing a richer topology than earlier benchmarks. Compared to previous benchmarks such as TwiBot-20 [25], TwiBot-22 provides a more comprehensive graph structure with diversified entity and relation types, making it suitable for evaluating graph-based bot detection methods. The metadata contained in the TwiBot-22 dataset is shown in Table 1 below.

Table 1. Entities and metadata in the TwiBot-22 dataset.

Entity Name	Role in Our Framework	Metadata Used in This Work
User	Categorical features, numerical features, BERTweet description embedding	created_at, description, entities, location, name, profile_image_url, protected_url, username, verified, withheld, followers_count, following_count, tweet_count, listed_count.
Tweet	Tweet-type distribution, post influence, BERTweet tweet embedding	attachments, context_annotations, entities, created_at, geo, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text, withheld, retweet_count, reply_count, like_count, quote_count.
List	Numerical features (listed count, listed-to-followers ratio)	private, created_at, description, name, follower_count, member_count.
Hashtag	Description hashtag count, tweet content preprocessing	hashtag_name

3.2. Data Processing and Feature Engineering

This section describes the data processing pipeline and the construction of multi-modal node features, with the main processing flow illustrated in Figure 2. Building on the multi-modal perspective advocated by Arranz-Escudero et al. [18], we extract and fuse three complementary feature views for each user.

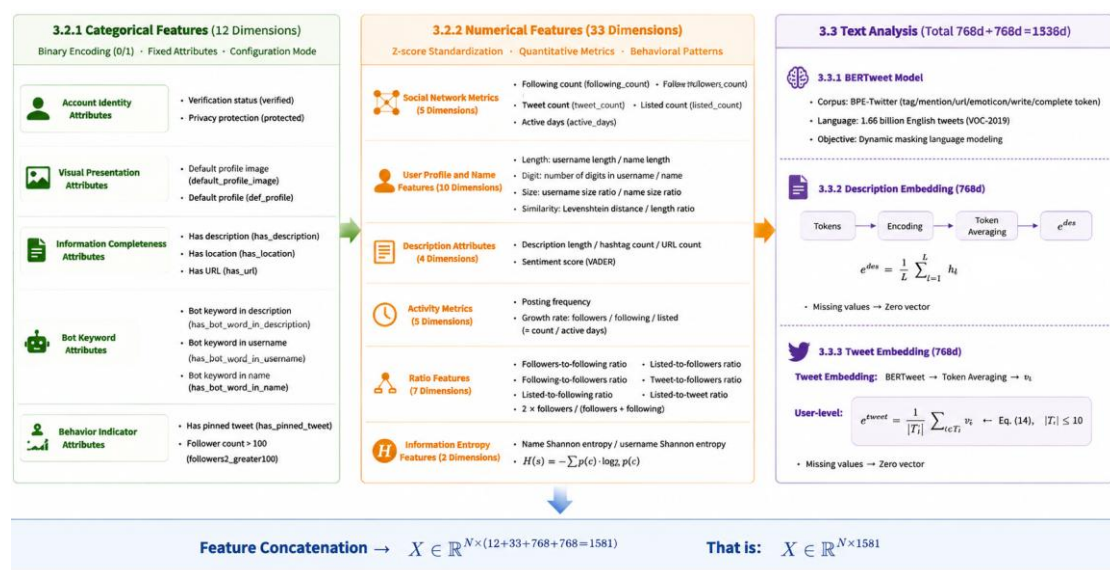


Figure 2. The process of feature engineering.

3.2.1. Categorical Features

We extract 12 categorical features that encode binary or discrete profile configurations. These features are represented as 0/1 indicators or integer values and capture distinct account properties that are difficult for bots to manipulate consistently. The categorical features include:

Account identity attributes: verified status and privacy protection setting.

Visual presentation attributes: default profile image indicator and default profile indicator.

Information completeness attributes: presence of description, location, and URL.

Bot-related keyword attributes: binary indicators for the presence of the word “bot” in the description, username, and name.

Behavior indicator attributes: whether the user has a pinned tweet and whether twice the follower count exceeds 100.

These 12 categorical features are concatenated into the categorical sub-vector $x_u^{\text{cat}} \in 0,1^{12}$.

3.2.2. Numerical Features

We extract 33 numerical features that quantify user activity, profile metadata, and derived behavioral ratios. All numerical features are standardized using z-score normalization to have zero mean and unit variance. The z-score means and standard deviations are computed at the preprocessing stage, before any data split, over the full population of 1,000,000 users (transductive setting: the entire graph is available at training time), without using any label information; the standard deviation uses the sample formula (ddof = 1). This convention is consistent with the transductive evaluation protocol of the benchmark and is stated explicitly here to facilitate independent reproduction. The numerical features can be grouped into the following categories:

Social network metrics: following count, followers count, tweet count, listed count, and account age in days.

User profile and name features: username length, name length, number of digits in username/name, ratio of uppercase to lowercase letters in username/name, total number of digits in username/name, the ratio of username length to name length, and the Levenshtein distance between username and name.

Description attributes: description length, number of hashtags in description, number of URLs in description, and description sentiment score computed with VADER.

Activity metrics: posting frequency, follower growth rate, friend growth rate, listed growth rate, and active-day-normalized activity.

Ratio features: followers-to-following ratio, following-to-followers ratio, following-to-total ratio, listed-to-followers ratio, tweets-to-followers ratio, listed-to-tweets ratio, and the relative followers' score.

Information entropy features: Shannon entropy of the name and username.

These 33 numerical features form the numerical sub-vector $x_u^{num} \in \mathbb{R}^{33}$.

3.2.3. Text Analysis

We use BERTweet [13] to encode user profile descriptions and recent tweets. Each description and each tweet is truncated to a maximum sequence length of 50 tokens. For tweets, we aggregate embeddings from up to 10 historical tweets per user via mean pooling. The final textual representation for each user consists of a 768-dimensional description embedding and a 768-dimensional tweet embedding, yielding a 1536-dimensional semantic feature vector.

BERTweet is chosen over BERT [12] because of its more robust pretraining scheme, which has been shown to produce better representations for short and noisy social media text. The use of pretrained language models allows the model to capture subtle semantic cues in user-generated content without requiring task-specific feature engineering.

Formally, the description embedding $e_u^{des} \in \mathbb{R}^{768}$ is computed by averaging the token-level RoBERTa outputs over the input sequence length L :

$$e_u^{des} = \frac{1}{L} \sum_{l=1}^L h_l^{des} \quad (1)$$

Similarly, the tweet embedding $e_u^{tweet} \in \mathbb{R}^{768}$ is obtained by first averaging the token-level outputs for each tweet and then averaging over up to $|T_i| \leq 10$ tweets:

$$v_t = \frac{1}{L_t} \sum_{l=1}^{L_t} h_l^{(t)}, \quad e_u^{tweet} = \frac{1}{|T_i|} \sum_{t \in T_i} v_t \quad (2)$$

Missing descriptions or empty tweet histories are represented as zero vectors.

3.2.4. Feature Integration and Embedding

The complete node feature vector for each user is formed by concatenating the 12 categorical features, 33 numerical features, a 768-dimensional description embedding, and a 768-dimensional tweet embedding, resulting in a total feature dimension of 1581.

Formally, the node feature vector x_u for user u is defined as:

$$x_u = [x_u^{cat} | x_u^{num} | e_u^{des} | e_u^{tweet}] \quad (3)$$

where $|$ denotes vector concatenation.

After concatenating the heterogeneous features, we obtain a raw node feature matrix $x_u \in \mathbb{R}^{N \times 1581}$, where N is the number of users. The feature embedding layer projects this high-dimensional representation into a lower-dimensional latent space via a linear transformation followed by a non-linear activation:

$$H^{(0)} = \text{ReLU}(Wx_u + b) \quad (4)$$

where the weight matrix $W \in \mathbb{R}^{128 \times 1581}$ and the bias $b \in \mathbb{R}^{128}$ are learnable parameters. The output $H^{(0)} \in \mathbb{R}^{N \times 128}$ serves as the initial node representation for the subsequent graph learning stage.

The feature embedding layer serves two primary purposes:

Feature dimensionality reduction: It compresses the 1581-dimensional raw feature vector into a 128-dimensional dense representation, reducing the model's parameter count and computational cost.

Multi-feature semantic fusion: It fuses categorical features, numerical features, description text, and tweet text into a unified user representation, allowing the model to learn cross-feature correlations through end-to-end training.

3.3. Graph Network Construction

After transforming the multi-modal user features into dense node representations, we construct two complementary graph structures over the user set. The social relation graph encodes explicit following relationships, while the auxiliary behavior similarity graph captures implicit behavioral affinities derived from user activity patterns. Together, these graphs provide relational contexts that help distinguish bots from humans.

3.3.1. Relationship Graph Construction

The social relation graph is constructed from the follower and following relationships provided in TwiBot-22. Formally, let $G_r = (\mathcal{V}, \mathcal{E}_r)$ denote the social relation graph, where $\mathcal{V}$ is the set of users and $\mathcal{E}_r$ is the set of directed edges. We consider two relation types: follower (incoming) and following (outgoing). Each edge $e = (u, v) \in \mathcal{E}_r$ is represented by an edge index $e = [u, v]^T$ and an edge type $r_e \in \{0, 1\}$, where $r_e = 0$ denotes a follower relation (i.e., u is followed by v) and $r_e = 1$ denotes a following relation (i.e., u follows v). In matrix form, the edge index is stored as $E \in \mathbb{N}^{2 \times |\mathcal{E}_r|}$ and the edge type as $r \in \mathbb{N}^{|\mathcal{E}_r|}$. These two tensors are fed into RGCN [26], which applies relation-specific weight matrices during message passing.

This graph captures explicit social connections among users. In the context of bot detection, bots often follow many human accounts to gain visibility, and some bots are connected to each other through follow-back patterns. RGCN is well-suited for this graph because it can model multiple relation types with separate weight matrices, as described in the RGCN forward pass below.

As illustrated in Figure 3, the RGCN module takes the projected node features $H^{(0)} \in \mathbb{R}^{N \times 128}$ together with the relationship graph (E, r) as inputs. For each node A , the computation in the first RGCN layer proceeds in three steps:

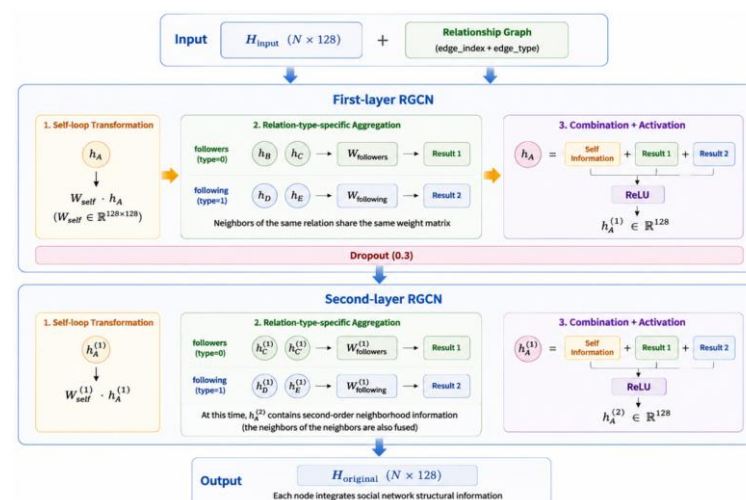


Figure 3. The process of relationship graph construction.

Self-loop transformation. The node representation $h_A \in \mathbb{R}^{128}$ is transformed by the self-loop weight matrix $W_{\text{self}} \in \mathbb{R}^{128 \times 128}$:

$$h_{A,\text{self}} = W_{\text{self}} h_A. \quad (5)$$

Relation-type-specific aggregation. Neighbors of node A are grouped by relation type. For the follower relation (type 0), the embeddings of follower neighbors {B, C} are aggregated with weight matrix $W_{\text{followers}}$ to produce Result 1; for the following relation (type 1), the embeddings of following neighbors {D, E} are aggregated with weight matrix $W_{\text{following}}$ to produce Result 2. Neighbors sharing the same relation type use the same weight matrix.

Combination and activation. The self-loop information and the two relation-specific aggregation results are summed and passed through a non-linear activation function:

$$h_A^{(1)} = \sigma(\text{SelfInformation} + \text{Result1} + \text{Result2}) \in \mathbb{R}^{128} \quad (6)$$

After the first RGCN layer, dropout regularization (with the ratio selected from {0.2, 0.3, 0.4} via hyper-parameter tuning; see Section 4.1 for details) is applied to obtain the layer-1 output. In our initial configuration we set it to 0.3, while the final optimal value is 0.2 as determined in Section 4.1. A second RGCN layer repeats the same three steps over $h_A^{(2)}$, using weight matrices $W_{\text{self}}^{(2)}$, $W_{\text{followers}}^{(2)}$, and $W_{\text{following}}^{(2)}$, yielding the final social-relation-aware representation $H_{\text{original}} \in \mathbb{R}^{N \times 128}$, where each node integrates second-order social network structural information.

3.3.2. Behavior Similarity Graph Construction

It is worth emphasizing that the behavioral signals used in this section serve a fundamentally different role from the node features described in Section 3.2. The node features characterize who a user is and what they express; the three behavioral signals here are designed solely to define how users resemble each other, acting as relational inductive biases that determine the topology and edge weights of the behavior similarity graph. Because these signals are deterministic functions of the same raw metadata used for node features, the behavior branch does not introduce information that is absent from the node feature matrix. Its role is therefore not to add new information in an information-theoretic sense, but to re-route and re-weight existing information through a behavioral-neighborhood aggregation path. Whether this re-routing helps is an empirical question: as shown in Section 4.5, it is functionally active only on the users the behavior graph covers, and its effect there is a systematic shift in the decision geometry rather than a uniform performance gain. This is why the branch is framed as a sparse, conditionally active auxiliary view rather than a co-equal branch.

The behavior similarity graph captures implicit behavioral affinities among users. Its construction consists of five stages: behavior feature extraction, user filtering, similarity computation, threshold determination, and graph network construction. The graph is constructed once over the full user set in a transductive manner and is then fixed: the same graph is used in training, validation, and testing. The active-user selection and the similarity threshold depend only on label-free activity statistics (post counts and interaction metadata) and do not consult the train/validation/test split or any class label, so no test-set information enters the construction.

In the TwiBot-22 graph benchmark evaluation, transductive full-graph inference is a common and accepted setting: the entire graph topology and node features are visible during training, while only the labels of training nodes are used in the loss. Under this setting, using the unlabeled metadata of test nodes (e.g., post counts, follower/following counts) to construct an auxiliary graph does not involve any class label and therefore does not constitute label leakage.

Although some of these signals (e.g., follower/following counts) are derived from the same raw metadata as the node features, their role in the model is distinct: they are used only to construct the graph structure and are not updated during training (except for their influence on the attention coefficients in the GAT layer). Therefore, they do not create redundancy with the node features; instead, they provide a complementary graphical view of behavioral affinity that is propagated through the network, while the node-level semantic and attribute information remains fully preserved in the feature vectors.

Stage 1: Behavior feature extraction. For each user u , we extract three behavioral signals, with the processing flow illustrated in Figure 4.

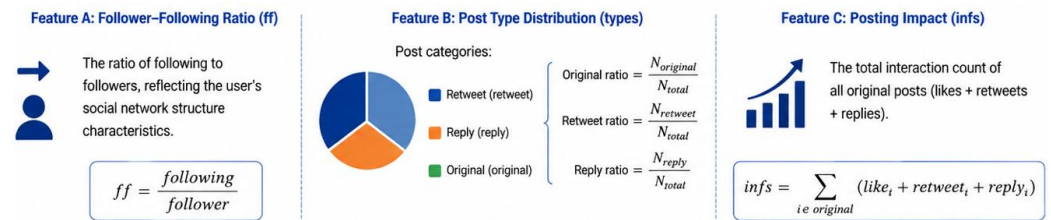


Figure 4. The process of behavior feature extraction.

Feature A—Follower–following ratio $ff(u)$: The ratio reflecting the user’s social network structure characteristics.

$$ff(u) = \frac{\max(\text{following}(u), 1)}{\max(\text{followers}(u), 1)} \quad (7)$$

Feature B—Post type distribution (Types): A three-dimensional probability vector $[p_{\text{original}}, p_{\text{retweet}}, p_{\text{reply}}]^T$, representing the proportions of original tweets, retweets, and replies among the user’s posts.

Feature C—Posting impact (Infs): The signal measures the total interaction count received by the user’s original posts.

$$\text{Infs}(u) = \sum_{t \in \text{original}(u)} (\text{like}_t + \text{retweet}_t + \text{reply}_t) \quad (8)$$

Users with no tweets are assigned a tweet-type distribution of (0, 0, 0) and a post-influence of 0, and such users do not enter the behavior similarity graph; the active-user pool requires a user to have at least 5 tweets in the dataset. All ratio-type hand-crafted features have a zero-denominator guard (the ratio is set to 0 when the denominator is 0), and when the account creation time is missing, the number of active days is counted as 1 day. The three signals are selected on the basis of their discriminative relevance to bot behavioral similarity, which is the phenomenon the behavior similarity graph is intended to model. The follower–following ratio reflects the asymmetric follow patterns commonly associated with follower–farming operations; the post-type distribution captures the extent to which a user acts as a content producer rather than a retweet-based amplifier, the latter being characteristic of coordinated amplification; and the post influence measures the engagement attracted by a user’s original posts, which may diverge markedly from what would be expected given the account’s follower count. None of these signals depends on text comprehension, so their computation is inexpensive and language-agnostic. Moreover, they do not constitute the only behavioral information available to the model: per-user behavioral semantics are additionally provided by the hand-crafted features and the text embedding described in Section 3.2.

Stage 2: User filtering. To control computational cost, we rank all users by their post count in descending order and select the top $K = 10,000$ most active users for subsequent pairwise similarity computation. The choice of an activity-based ranking, rather than a random subset, follows from the observation that content amplification requires content

generation in order to amplify it; accounts in the long tail of the activity distribution, which typically post rarely or not at all, yield degenerate behavioral features (e.g., an undefined post-type distribution and zero influence) and therefore contribute little reliable signal to a pairwise similarity measure. Restricting the graph to the top-K active users thus concentrates the computation on the accounts for which the behavioral features are most informative. It should also be noted that users outside this subset are not excluded from the model: they remain represented by their feature vectors and by the social relation graph (Section 3.3.1), so the behavior similarity graph supplements, rather than replaces, the information available for such accounts.

Stage 3: Similarity computation. For any two users A and B, we compute a similarity score for each behavioral signal separately and then sum them to obtain an overall similarity.

Follower–following similarity: We use relative difference to measure similarity, with larger values indicating higher similarity:

$$w_{ff} = 1 - \frac{|ff_A - ff_B|}{\max(ff_A, ff_B)} \quad (9)$$

Post type distribution similarity: We use the Manhattan distance (L1 norm) to measure the difference between the two post-type probability distributions. The $w_{types} \in [0,1]$.

$$w_{types} = \max(0, 1 - (|p_{A,original} - p_{B,original}| + |p_{A,retweet} - p_{B,retweet}| + |p_{A,reply} - p_{B,reply}|)) \quad (10)$$

Posting impact similarity: We again use relative difference:

$$w_{infs} = \begin{cases} 1 - \frac{|infs_A - infs_B|}{\max(infs_A, infs_B)}, & \text{otherwise} \\ 0, & \text{if } \max(infs_A, infs_B) = 0 \end{cases} \quad (11)$$

To suppress unreliable pairs, each per-signal similarity is gated before summation: the effective contribution is $w_k \cdot G_k$, where the binary gate $G_k \in \{0,1\}$ equals 1 only if the pair lies within a 50% relative difference on that signal— $|ff_A - ff_B| < 0.5 \max(ff_A, ff_B)$ and $\min(ff_A, ff_B) \geq 0.1$ for the follower–following term, $\Delta_{types} < 0.5$ (with Δ_{types} denoting the L1 distance in Equation (10)) for the post-type term, and $|infs_A - infs_B| < 0.5 \max(infs_A, infs_B)$ for the posting-impact term—and equals 0 otherwise. Gating is a per-signal saturation mechanism, not an exclusion rule: no pair is dropped at this stage, and a pair whose post-type term is gated to zero can still be admitted on the strength of the other two signals. The overall behavioral similarity is the sum of the gated contributions:

$$\text{total}(A,B) = w_{ff} + w_{types} + w_{infs} \quad (12)$$

Because each of the three terms lies in $[0, 1]$, the three terms are on a common scale and the combined weight satisfies $\text{total}(A,B) \in [0,3]$.

Stage 4: Similarity threshold determination. The threshold is percentile-based: the percentile q is a hyper-parameter of the pipeline (fixed to 80 in all our experiments), while the threshold value τ itself is derived from the empirical similarity distribution rather than set manually. The choice of the 80th percentile is motivated by the observation that only a small fraction of user pairs exhibit strong behavioral similarity, and these pairs are the most plausible candidates for behaviorally related accounts.

Concretely, we take the first 500 user pairs from the top-K active users, compute their overall similarities, sort the scores, and set the threshold τ to the 80th percentile of these 500 scores, additionally clamped to the range $[0.1, 2.0]$ as an implementation safeguard. This selection is deterministic and label-free: it depends only on the ordering of the dataset, not on the train/validation/test split or on any class label. We then iterate over user pairs

and add an undirected edge (u, v) whenever $\text{total}(u, v) \geq \tau$. Each admitted edge is stored in both directions, together with its overall similarity score as the edge weight.

Density is controlled by two successive mechanisms. First, the percentile threshold τ determines which pairs are admitted: with $K = 10,000$, it admits 9,458,072 directed edges (4,729,036 undirected pairs) covering 9998 users. Second, to bound the memory and computation cost of the GAT branch, the released implementation enforces an edge budget $M = 10^6$ when the graph is loaded for training: edges are ranked by weight, and only the top- M are retained. The behavior similarity graph used in all experiments therefore contains 1,000,000 directed edges (500,000 undirected pairs) covering 9844 of the 1,000,000 users, with retained weights in $[1.839, 2.983]$ (mean 2.124). The reciprocal structure is preserved under this truncation: each undirected similarity pair is emitted as two directed edges carrying bit-identical weight values, so they are either both retained or both discarded in the weight-based truncation. Empirical verification on the final constructed artifact shows that the retained 1,000,000 directed edges decompose exactly into 500,000 reciprocal pairs (the reverse edge of every retained edge is also in the retained set), i.e., the truncation boundary does not split any undirected pair. The pipeline's sensitivity to the remaining hyper-parameters is limited. Because the edge weight is a monotone function of the pair's overall similarity, retaining the top- M edges by weight is equivalent to retaining the top- M by raw similarity, independent of the threshold τ and of the calibration sample. Therefore, the percentile q and the calibration-pair count only determine which low-similarity pairs are excluded from the candidate pool, and as long as the pool exceeds M by a wide margin, none of those excluded pairs could have entered the top- M graph anyway. The final topology is thus governed by K and M together with the similarity ranking, rather than by the precise values of q and the pair count.

Algorithm 1 summarizes the complete behavior similarity graph construction process:

Algorithm 1. Behavior Similarity Graph Construction

Input: User behavioral feature matrix $B \in \mathbb{R}^{N \times d_b}$, maximum users $K = 10,000$, threshold pairs $P = 500$, maximum edges $M = 10^6$.

1. Select top- K most active users based on tweet count.
2. Extract behavioral features ff , types, and infs for each selected user.
3. Compute per-signal similarities w_{ff} , w_{types} , and w_{infs} for all user pairs (50% relative-difference gates; Equations (9)–(11)).
4. Sum per-signal similarities to obtain the overall similarity $\text{total}(u, v)$.
5. Take the first P user pairs, sort their overall similarities, and set the threshold τ to the 80th percentile (clamped to $[0.1, 2.0]$).
6. Add edge (u, v) if $\text{total}(u, v) > \tau$. If total edges exceed M , rank them by weight and retain the top- M .

Output: Edge list $\mathcal{E}_b$ and edge weights w_b for behavior similarity graph $\mathcal{G}_b = (\mathcal{V}, \mathcal{E}_b)$.

The computational complexity of pairwise similarity computation is $O(K^2 d_b)$, which is manageable for $K = 10,000$ but may require approximate methods for larger datasets.

The behavioral signals also serve a precise and limited function: the three behavioral signals determine the topology of the behavior similarity graph—that is, the existence and strength of each edge—whereas the node representations propagated through the GAT branch are 128-dimensional projections of the full 1581-dimensional feature vectors, comprising the hand-crafted features and the BERTweet text embedding. The behavior similarity graph thus encodes pairwise behavioral affinity, while the attribute-level information of each user is preserved in full at the node level. This separation of roles implies that the graph is intended to capture behavioral-affinity links that are not observable from the social relation graph, rather than to provide a complete account of user behavior.

Users that are not covered by the behavior similarity graph are formally treated as zero-degree nodes in the GAT branch: they have no behavioral neighbors from which to aggregate information. To keep the attention normalization numerically stable for such nodes, a self-loop is added to every node, so that each user aggregates at least its own representation. Consequently, for a zero-degree node, the attention coefficients reduce to the self-loop term and the GAT output degenerates to a linear transformation of its own projected embedding. Its behavior-aware representation is therefore determined solely by its multi-modal node features, which remain fully informative, while no information is propagated from other users. This treatment guarantees that the GAT branch is well defined over the entire user population and that the behavioral view contributes evidence exactly where behavioral evidence exists. Because the top-K set is selected over all users, it spans the training, validation, and test splits: a test user inside the top-K set that retains at least one behavior edge aggregates evidence from its behavioral neighbors at inference time; all remaining test users reduce to the self-loop case above and rely on the social relation graph and their node features.

Stage 5: Graph network construction. The GAT branch processes the behavior similarity graph network; each GAT layer performs five operations: linear transformation, attention score computation, softmax normalization and weighted aggregation.

Step 1: Linear transformation. The input node representation $h_i \in \mathbb{R}^{128}$ is first projected by a learnable weight matrix $W \in \mathbb{R}^{128 \times 128}$:

$$h'_i = Wh_i \quad (13)$$

Step 2: Attention score computation. For each edge (j, i) , a raw attention score is computed by concatenating the transformed representations of the center node i and its neighbor j , followed by a dot product with the attention parameter vector $a \in \mathbb{R}^{256}$ and a LeakyReLU non-linearity with negative slope 0.2.

$$e_{ij} = \text{LeakyReLU}_{0.2} \left(a^\top [h'_i \parallel h'_j] \right) \quad (14)$$

This score measures the importance of neighbor j to node i .

Step 3: Softmax normalization. The raw scores are normalized across all neighbors of node i to obtain attention coefficients:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \quad (15)$$

In our implementation, edge weights do not enter the attention mechanism: the coefficients of Equation (15) are computed from node features only. All reported results use the stock GATConv of PyTorch Geometric 2.6.1 (single head, no edge features); the edge weights of the behavior similarity graph act exclusively at the graph-construction stage, where they determine neighbor admission through the percentile threshold and the top-M edge budget (Stage 4).

Step 4: Weighted aggregation. The neighbor representations are aggregated using the adjusted attention weights, followed by a ReLU activation:

$$h_i^{(1)} = \text{ReLU} \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} h'_j \right) \in \mathbb{R}^{128} \quad (16)$$

After the first GAT layer, dropout with a ratio of 0.3 is applied as the initial setting (the ratio is later tuned within $\{0.2, 0.3, 0.4\}$; see Section 4.1 for the optimal choice). A second GAT layer repeats the same four steps over $h_i^{(1)}$, yielding the final behavior-aware representation $H_{\text{behavior}} \in \mathbb{R}^{N \times 128}$, where each node integrates behavior similarity information from its neighborhood.

3.4. Fusion and Classification

The outputs of the relation and auxiliary behavior branches are concatenated and passed through a two-layer MLP with dropout.

$$\begin{cases} z = [H_{\text{behavior}} | H_{\text{original}}] \\ \hat{y} = W_2 \text{ReLU}(W_1 z + b_1) + b_2 \end{cases} \quad (17)$$

To account for the class imbalance of TwiBot-22 (bot prevalence 13.99%), the model is trained with a class-weighted cross-entropy loss, assigning the bot class a weight $w_1 = 1.5$ relative to the human class ($w_0 = 1.0$).

$$\mathcal{L} = -\frac{1}{|\text{train}|} \sum_{u \in \text{train}} [w_0 \cdot y_u \cdot \log p_u + w_1 \cdot (1 - y_u) \cdot \log(1 - p_u)] \quad (18)$$

where $p_u = \text{softmax}(z_u)$ denotes the bot probability of user u . This weight shifts the learned decision boundary toward the minority bot class and was selected on the validation set from $\{1.0, 1.4, 1.5, 1.9, 3.3\}$.

During inference, a user is classified as a bot if its bot probability exceeds a decision threshold, $p_u \geq \tau^*$, rather than by the default argmax rule (equivalent to a fixed threshold of 0.5). The threshold is calibrated per seed on the validation set only. That is, among candidate thresholds scanned in steps of 0.01, we select the one maximizing validation macro-F1 subject to validation accuracy not falling below its default-threshold level. This deployment-time adjustment trades a marginal amount of accuracy for a larger gain in the class-balanced metric and involves no test-set information.

4. Experiments

We evaluate the proposed model on the TwiBot-22 dataset. The dataset is partitioned into training (70%), validation (20%), and testing (10%) sets using a stratified random approach that preserves the class distribution. The experiments are conducted on a server equipped with two NVIDIA A10 GPUs, two AMD EPYC 9554 CPUs, and 256 GB RAM. The implementation is based on PyTorch 2.7.1 (CUDA 11.8). Our unified evaluation protocol combines three components—stratified five-seed splits, class-weighted training, and validation-calibrated decision thresholds—and all headline numbers are reported at the per-seed calibrated operating point defined in this section.

4.1. Training Details

We use the AdamW optimizer [27] with a learning rate selected from $\{0.0001, 0.001, 0.01\}$ and a weight decay of 5×10^{-4} . The hidden dimension is selected from $\{64, 128\}$, dropout from $\{0.2, 0.3, 0.4\}$, and the number of training epochs from $\{100, 150, 200\}$. Model weights are initialized using Xavier uniform initialization [28]. The social relation graph and the behavior similarity graph are constructed once and kept fixed across all phases; during training, message passing in both branches operates over the full graph, where validation and test users participate only as unlabeled graph nodes (their features are inputs), but their labels are never accessed and the training loss is computed only on labeled training users.

To account for the class imbalance of TwiBot-22 (13.99% bot prevalence), the cross-entropy loss assigns the bot class a weight of 1.5 relative to the human class (weight 1.0). The weight was selected on the validation set from $\{1.0, 1.4, 1.5, 1.9, 3.3\}$; this shifts the decision boundary toward the minority class and improves the class-balanced metric at a small cost in overall accuracy.

After training, the decision threshold is calibrated per seed on the validation set: we select τ^* that maximizes validation macro-F1 subject to validation accuracy not falling below its level at the default 0.5 threshold, scanning $\tau \in [0.05, 0.95]$ in steps of 0.01. No

test-set information is involved. All test metrics in this paper are evaluated at this per-seed τ^* ; where relevant, default-threshold (0.5) values are additionally reported for transparency. For the full model, the validation-selected thresholds are $\tau^* = 0.42, 0.45, 0.43, 0.41$, and 0.41 for seeds 0–4 (mean 0.42); across all ablation variants the range is $\tau^* \in [0.38, 0.45]$ (Section 4.4).

To quantify seed-level variability, every configuration in the grid search is used to train five independent single models with random seeds. Each seed independently shuffles the node indices and produces its own 70/20/10 train/validation/test split, and no ensemble is used in any reported result. Within a fixed seed, the ablation variants of Section 4.4 reuse the same split as the corresponding full model, which is what makes the paired significance tests across seeds valid.

Because the behavior similarity graph is restricted to the top 10,000 active users and the full graph fits into GPU memory, we perform full-batch training: in each epoch the entire graph is forwarded through the model and the loss is computed only on the training nodes indicated by the train mask. Both the model weights and the hyper-parameters are selected based on the validation set: we perform a grid search over all candidate configurations and pick the one that yields the highest validation macro-F1. The performance of the selected model is then reported on the held-out test set.

4.2. Evaluation Metrics

We evaluate the model's performance using accuracy and the macro-F1 score, two widely adopted metrics in classification tasks. Accuracy measures the proportion of correctly classified samples among the entire test set:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \quad (19)$$

where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively.

The macro-F1 score computes the F1 score for each class independently and then takes the unweighted mean across all C classes:

$$\text{Macro-F1} = \frac{1}{C} \sum_{c=1}^C \frac{2P_c R_c}{P_c + R_c} \quad (20)$$

where P_c and R_c are the precision and recall for class c , respectively. Unlike accuracy, which can be dominated by the majority class, macro-F1 assigns equal weight to each class and thus provides a more balanced assessment of the model's performance on imbalanced datasets such as TwiBot-22.

4.3. Experiment Results

To evaluate the proposed model, we compare it against representative baselines previously evaluated on the TwiBot-22 benchmark. Because the original papers differ in data partitions, input features, and preprocessing, cross-paper numbers are not strictly comparable; we therefore separate the baselines into two groups. Strictly matched comparisons are rerun by us under the exact protocol of Section 4.1 — identical 70/20/10 stratified splits, identical input features, and the five-seed paired protocol. Literature-reported comparisons quote the numbers of the corresponding publications and serve as context only, situating our model within the broader literature rather than claiming strict superiority. The strongest relevant baseline, TMTM [18], reports a 70/20/10 stratified partition, which belongs to the same protocol family as ours but uses different actual partitions, features, and preprocessing. We therefore treat TMTM as literature-reported context and make no

statistical claim across papers: the significance tests in this paper apply exclusively to within-protocol comparisons, where all variants share identical splits, features, and seeds. Their model structures and literature reports are briefly described in Tables 2 and 3. Among these, since BotRGCN is a commonly used open-source baseline model, we reran it using the same data partition as our model.

Table 2. Descriptions of the relative models.

Authors	Method	Detail
Zeng et al., 2023 [15]	MRLBot	A graph-based framework that leverages following – follower relations and behavioral interactions without text or explicit user features.
Feng et al., 2023 [24]	BotRGCN	BotRGCN is a relational graph convolutional network for Twitter bot detection that models users and their heterogeneous relations.
Wu et al., 2024 [14]	BotSCL	A heterophily aware contrastive learning framework that separates the representations of heterophilous neighbors while aligning homophilous ones, addressing the feature blending problem in graph-based bot detection.
Arranz-Escudero et al., 2025 [18]	TMTM	This method integrates feature, text, and tweets within an RGCN model that captures following, follower, and ownership relations among users.
Guyan et al., 2025 [16]	PEGNN	A graph network that leverages a single social relation graph, enhancing node representations with peripheral information for bot detection.

Table 3. Literature-reported results (original publications; different partitions and preprocessing).

Model	Accuracy (%)	Macro-F1 (%)
MRLBot	80.25	61.45
BotSCL	82.39 ± 0.05	61.53 ± 0.145
PEGNN	88.71	65.65
TMTM	90.06 ± 0.21	76.58 ± 0.117

Table 4 reports the comparison results. Among the strictly matched baselines, our model outperforms BotRGCN on both metrics. Both the rerun BotRGCN and the literature-reported results are evaluated at the default 0.5 decision threshold, whereas our headline numbers use the per-seed calibrated threshold τ from Section 4.1. This threshold is selected on validation data only, maximizing validation macro-F1 under a validation-accuracy constraint; no test information is involved. To make the operating-point difference fully transparent, we report both variants of our model. At the default 0.5 threshold, our model achieves 89.92 ± 0.07 accuracy and 77.28 ± 0.31 macro-F1; at the calibrated τ , it achieves 89.48 ± 0.07 and 78.11 ± 0.14 . The comparison outcome is insensitive to this choice: under either operating point, our macro-F1 exceeds the strongest literature-reported baseline (77.28 or 78.11 vs. 76.58 ± 0.12 for TMTM), and our accuracy remains within 0.58 pp of TMTM’s reported value (0.13 pp at the default threshold). Threshold calibration is a deployment-time adjustment that trades a small amount of accuracy for a larger gain in the class-balanced metric; Section 5.1 analyzes this trade-off in detail.

Table 4. Comparison on TwiBot-22.

Model	Accuracy (%)	Macro-F1 (%)
-------	--------------	--------------

BotRGCN	83.12 ± 0.13	73.48 ± 0.28
Ours	89.48 ± 0.07	78.11 ± 0.14

4.4. Ablation Experiments

To assess the contribution of each graph branch, we conduct a branch-removal ablation under the same unified protocol as the main results (Section 4.1): all variants are trained with weighted cross-entropy (bot-class weight 1.5), model selection is performed on validation macro-F1, and the decision threshold is calibrated per seed on the validation set ($\tau^* \in [0.38, 0.45]$ across variants and seeds). We compare four variants: (i) the full model (dual-graph); (ii) w/o behavior graph, which removes the GAT branch and reduces the model to the RGCN relation branch; (iii) w/ random behavior graph, which replaces the behavior similarity graph with a degree-preserving random graph of identical size and edge count, isolating behavioral similarity from branch capacity; and (iv) w/o relation graph, which removes the RGCN branch and reduces the model to the GAT branch. All variants share the same optimal hyper-parameters (hidden dimension 128, dropout 0.2, learning rate 0.001, 200 epochs) and are averaged over five random seeds; within each seed, all variants reuse the same split, which is what makes the paired significance tests valid. Table 5 reports accuracy and macro-F1 at the per-seed calibrated operating point.

Table 5. The ablation experiment results.

Model	Accuracy (%)	Macro-F1 (%)
w/o relation graph (behavior only)	88.07 ± 0.14	75.89 ± 0.23
w/random behavior graph	89.41 ± 0.13	78.07 ± 0.25
w/o behavior graph (relation only)	89.45 ± 0.10	78.10 ± 0.19
Full Model (Dual-Graph)	89.48 ± 0.07	78.11 ± 0.14

Contribution of the relation graph. Removing the RGCN branch causes the largest degradation: accuracy drops by 1.41 pp and macro-F1 by 2.22 pp (78.11 → 75.89), and the paired differences are positive in all five seeds (both $p < 0.001$). Explicit follower–following structure is thus the dominant graph signal, consistent with the abnormal following patterns that relational message passing is designed to capture.

Contribution of the behavior similarity graph. The marginal effect of the behavior branch on top of the relation branch is not statistically detectable at the global test-set level: the full model and the relation-only variant differ by +0.03 pp in accuracy ($p = 0.48$). This is consistent with the coverage analysis in Section 3.3.2: after density control, the behavior graph covers only 9844 of 1,000,000 users ($\approx 0.98\%$), so the branch can act only on a small high-activity sub-population. Therefore, a broad complementary gain cannot be claimed.

Random-graph control. To test whether the behavioral branch exploits graph structure beyond what is already encoded in node features, we rewire the behavioral graph while holding its degree and weight structure fixed. The behavioral graph is an undirected similarity graph stored symmetrically (each undirected pair appears as two directed edges with identical weight), from which we keep the top 1,000,000 directed edges by weight (9844 participating nodes). We rewire it by keeping the source sequence fixed and applying one uniform random permutation to the destination sequence; the weight array is untouched, so each rewired edge inherits the weight of its original slot (no resampling or reassignment). By construction this preserves, exactly and node-by-node (verified for every realization): the 1,000,000-edge budget, every node’s out-degree, every node’s in-degree, and the edge-weight distribution including each node’s outgoing weight profile. The rewiring intentionally destroys which pairs of nodes are connected: the fraction of edges whose reverse edge is also present drops from 100% (the graph is fully reciprocal by construction) to $\approx 4.0\%$, and the permutation introduces 187–203 self-loops per

realization ($\leq 0.021\%$ of edges; the original graph has none). We evaluate five independent rewiring realizations, one per training seed, each fully retrained under the identical protocol. The rewired-graph variant is statistically indistinguishable from the full model ($F1@t^*$: 78.07 ± 0.25 vs. 78.11 ± 0.14 , paired $p = 0.47$; $Acc@t^*$: 89.41 ± 0.13 vs. 89.48 ± 0.07 , $p = 0.23$), with small cross-realization variance.

Under the unified calibrated protocol, the relation branch supplies the bulk of the graph gain, while the global marginal contribution of the behavior branch lies within noise. The random-graph control further shows that, at the current $\approx 1\%$ coverage and with node-derivable behavioral signals, the behavior branch does not outperform a degree-preserving random graph of identical size and weight structure. We report it as such. The dual-branch architecture is retained not because it improves global averages, but because the behavior view is the only component that can in principle respond to behavioral affinity among socially unconnected accounts. Section 4.5 shows that this response is real and localized: the branch is functionally active exactly on the users it covers. Extending coverage and replacing node-derivable signals with non-node-derivable behavioral evidence are identified as the binding conditions for turning this localized activity into a global gain (Section 5.4).

4.5. Degree-Stratified Analysis of the Behavior Branch

The global ablation of Section 4.4 reports test-set averages, which are dominated by the 99% of users that the behavior graph does not cover. Rather than treating this null as an endpoint, this section turns it into a mechanism-level characterization: we localize where the branch acts (by graph exposure), diagnose whether its within-stratum effect reflects information loss or an operating-point artifact, and repair the artifact with a validation-only subgroup threshold. Following the observation that the retained behavior graph reaches only 9844 of 1,000,000 users ($\approx 0.98\%$) while all other users receive self-loops only, we stratify the test set by behavior degree and evaluate the full model against the relation-only variant within each stratum. Node degree is computed from the retained graph (Section 3.3.2) and is therefore observable at inference time. The stratified analysis, the per-node flip analysis (Table 6), and the subgroup remedy below are computed on the independently trained model set of Section 5.1—five seeds under the same unified training protocol—evaluated at its default-0.5 operating point, for which per-seed predictions and probabilities are retained; accordingly, the size-weighted stratum accuracies in Table 6 reproduce this set’s global accuracy of 89.92. The positive-degree stratum contains 967 test nodes per seed with a bot prevalence of 3.29%, and the zero-degree stratum contains 99,033 nodes per seed with a bot prevalence of 14.10%.

Table 6. Degree-stratified test performance.

Stratum	n/Seed	Bot-Prevalence	Model	Accuracy (%)	Bot-Precision (%)	Bot-Recall (%)	Macro-F1 (%)
degree > 0	967	3.29%	Full	96.77 ± 0.46	$46.67 \pm 44.72^*$	2.86 ± 2.75	51.85 ± 2.45
degree > 0	967	3.29%	Relation-only	96.37 ± 0.51	42.19 ± 12.32	27.08 ± 7.15	65.47 ± 4.48
degree = 0	99,033	14.10%	Full	89.85 ± 0.07	67.24 ± 0.98	54.75 ± 1.76	77.26 ± 0.36
degree = 0	99,033	14.10%	Relation-only	89.84 ± 0.07	66.86 ± 0.84	55.48 ± 1.63	77.39 ± 0.34

* The covered stratum contains only ≈ 32 bot accounts per seed; when a seed predicts no bots in this stratum, precision is zero by convention, which inflates the variance.

As shown in Table 6, on zero-degree nodes, which constitute the overwhelming majority of the test set, the two variants are statistically indistinguishable on all metrics (all $p \geq 0.27$, paired t-test over five seeds). The behavior branch neither helps nor harms the 99% of users it does not cover, which confirms that its effects are attributable to behavioral-

neighbor aggregation rather than to the additional self-feature transformation. Within the covered stratum, by contrast, the full model exhibits a localized degradation in both recall (2.86% vs. 27.08%, $p = 0.0011$) and macro-F1 ($p = 0.0018$), caused not by information loss but by an operating-point shift, as shown in Table 7.

Table 7. Per-node paired prediction changes (5-seed total).

Stratum	Flip 0→1 (/Seed)	Flip 1→0 (/Seed)	Flip Rate	Corrections: Breakages
degree > 0	0.0	19.4	2.01%	58: 39
degree = 0	232.6	449.8	0.69%	1737: 1675

Prediction changes concentrate precisely on covered nodes: the flip rate is 2.9× higher ($p = 0.0003$) and the probability shift 4.0× larger ($p = 0.0003$) than on uncovered nodes. The branch is thus functionally active exactly where behavioral edges exist, and the node-level net effect within the covered stratum is positive. The residual changes on zero-degree nodes are small and essentially symmetric (1737: 1675), i.e., metric-neutral; we attribute this drift to altered optimization through the shared fusion layer rather than to behavioral-neighbor aggregation.

The covered sub-population is 96.7% human (bot prevalence 3.29% vs. 14.10% outside), so behavioral-neighbor smoothing pulls covered bot accounts toward the human majority: their median bot-probability drops from 0.308 (relation-only) to 0.131 (full), while the human median remains low (0.014 vs. 0.008). Ranking quality within the stratum is fully preserved (subgroup ROC-AUC 0.897 ± 0.028 vs. 0.905 ± 0.019 , $p = 0.42$), indicating that the degradation is a threshold artifact rather than a loss of discriminative information.

Because behavior degree is observable at inference, the operating point can be repaired without any test-set information: for covered nodes we select a separate threshold τ_{sub} on the validation set only (maximizing validation macro-F1 within the stratum; $\tau_{\text{sub}} = 0.15 \pm 0.03$), while zero-degree nodes keep the global default threshold (0.5) of the same model set. This raises the covered-stratum macro-F1 from 51.85 to 67.17 ± 2.92 and bot recall to 45.07%. The remedy requires no retraining and is a pure deployment-time adjustment.

In summary, the behavior branch behaves exactly as designed—it propagates behavioral-neighborhood evidence, and does so only where behavioral edges exist—while its global contribution is bounded by the 0.98% coverage ceiling rather than by the formulation. Extending coverage beyond the top-10,000 active users is therefore the principal direction for strengthening this branch (Section 5.4). Beyond this specific branch, the analysis outlines a reusable evaluation pattern for multi-view architectures: when a global ablation is null, stratify test nodes by their exposure to the component, test whether the within-stratum effect reflects information loss (ranking quality preserved: subgroup ROC-AUC 0.897 vs. 0.905, $p = 0.42$) or an operating-point artifact (covered-bot probability median $0.308 \rightarrow 0.131$), and repair by validation-only subgroup calibration (51.85 $\rightarrow$ 67.17) without retraining. This localize–diagnose–repair sequence applies to any auxiliary view that covers a sub-population of the graph and, together with the degree-preserving random-graph control, constitutes a methodological contribution of this paper.

4.6. Effect of Text Embedding

To quantify the contribution of the text embedding, we retrain the full dual-graph model with all text features removed, so that each user is represented by the 45 hand-crafted features only (12 categorical and 33 numerical). The no-text variant is trained and evaluated under the identical unified protocol of Section 4.1—same stratified splits and seeds, class-weighted objective, validation-based model selection, and per-seed threshold calibration—so that the comparison is strictly matched. Table 8 reports the results.

Table 8. The effect of text embedding.

Model	Accuracy (%)	Macro-F1 (%)
Removing text embedding	87.41 ± 0.11	69.48 ± 0.14
Full Model (Dual-Graph)	89.48 ± 0.07	78.11 ± 0.14

Removing the text embeddings decreases accuracy by 2.07 pp and macro-F1 by 8.63 pp, with the paired difference positive in all five seeds (both $p < 10^{-5}$). The macro-F1 drop is roughly four times the accuracy drop, indicating that the minority bot class relies heavily on textual semantics: without descriptions and tweets, the model loses the linguistic evidence that distinguishes automated accounts from human ones, and the class-balanced metric degrades accordingly. Among the components examined in this paper, the text embedding is thus the largest single contributor to macro-F1 (−8.63 pp when removed), exceeding the contribution of the social relation graph.

Notably, although the projection layer maps both input variants to the same 128-dimensional hidden space before graph convolution, the two modalities encode fundamentally different aspects of user behavior—hand-crafted features capture structured profile signals, while the text embedding encapsulates semantic context from user-generated content.

This finding is consistent with the error analysis in Section 5.1: many false-negative cases involve bots with human-like structural profiles but distinctive textual content. The BERTweet embedding captures semantic nuances in user descriptions and tweets that structural and behavioral features alone cannot encode, such as content themes, linguistic patterns, and semantic consistency. These results further validate the necessity of multi-modal fusion: the significant performance drop when removing text embedding indicates that semantic information serves as an important complement to structural features. For well-disguised bot accounts that mimic human topological and behavioral patterns, textual cues often provide the distinguishing evidence, making both modalities indispensable.

5. Discussion

5.1. Error Analysis

To gain deeper insight into the limitations of the proposed model, we conduct a detailed error analysis under the unified protocol of Section 4.1. All quantitative results in this subsection are averaged over the five seeds (mean ± std, as shown in Table 9); the illustrative misclassification cases examined afterward are drawn from the seed-0 model as representative examples. The quantitative decomposition is performed at the default 0.5 threshold.

Table 9. Per-class precision, recall and F1.

Class	Precision (%)	Recall (%)	F1 (%)	Support
Human	92.86 ± 0.21	95.63 ± 0.31	94.22 ± 0.05	86,006
Bot	67.13 ± 0.98	54.83 ± 1.60	60.34 ± 0.64	13,994

The support counts are fixed across seeds by the stratified splits (bot prevalence 13.99%). On average, each seed’s model misclassifies 3762 human accounts as bots (a human false-positive rate of 4.37%) and 6321 bots as humans (a bot false-negative rate of 45.17%). The class-conditional error rates therefore exhibit a marked asymmetry: the bot FNR is roughly ten times the human FPR. At the default threshold the model operates in a precision-oriented regime—two of three accounts flagged as bots are indeed bots—at the cost of missing about half of the bot population. We emphasize that this asymmetry should be read as a property of the chosen operating point rather than of the architecture: the model’s test-set ranking quality is ROC-AUC = 0.912 ± 0.001 and PR-AUC = 0.680 ± 0.004 against a prevalence baseline of 0.140, indicating that the probability ranking separates the two classes well and that the precision-oriented regime is an operating-point choice. A controlled comparison of per-class recall with prior work is not feasible, as the strongest multi-modal baselines (e.g., TMTM) report only accuracy and F1 without per-class results.

As shown in Table 10, lowering the threshold trades accuracy for coverage of the minority class. At $\tau = 0.40$, bot recall rises to 64.12% and macro-F1 to 78.10%, at a cost of 0.69 pp accuracy. We note that the macro-F1-optimal region of this sweep ($\tau \approx 0.40$) coincides with the validation-calibrated operating point used in the main results ($\tau^* \approx 0.42$), cross-validating the calibration protocol on an independently trained model set.

Table 10. Threshold sweep on the test set.

Threshold	Accuracy (%)	Macro-F1 (%)	Bot-Precision (%)	Bot-Recall (%)
0.30	87.50 ± 0.35	77.29 ± 0.35	53.97 ± 1.03	73.03 ± 0.96
0.35	88.55 ± 0.25	77.93 ± 0.23	57.65 ± 1.00	68.57 ± 1.15
0.40	89.23 ± 0.18	78.10 ± 0.21	60.96 ± 0.92	64.12 ± 1.31
0.45	89.67 ± 0.11	77.88 ± 0.20	64.10 ± 0.95	59.57 ± 1.47
0.50	89.92 ± 0.07	77.28 ± 0.31	67.13 ± 0.98	54.83 ± 1.60
0.55	90.05 ± 0.08	76.39 ± 0.44	70.30 ± 1.00	50.04 ± 1.63
0.65	89.91 ± 0.08	73.46 ± 0.59	76.87 ± 1.01	39.97 ± 1.56
0.70	89.65 ± 0.10	71.26 ± 0.67	80.32 ± 0.83	34.52 ± 1.45

We further characterized the seed-0 errors (3337 false positives and 6655 false negatives) by applying explicit screening rules to every error in this well-defined sample (the seed-0 test split at the default 0.5 threshold); the group counts are exact, and the individual accounts quoted below are illustrative members of each group. Among the false positives, that is, human accounts wrongly flagged as bots, three rule-based patterns emerged (the rules may overlap, so shares do not sum exactly). The first is minimal-activity accounts (381 cases, 11.4% of the false positives; followers ≤ 5 and tweets ≤ 15), for example an account with 0 followers that follows 5 accounts, has published 0 tweets, was created about 8 months before data collection, and has no profile description (bot probability 0.96). The second is extreme follow asymmetry (117 cases, 3.5%; following-to-follower ratio ≥ 30 with at least 10 followers), for example an account with 23 followers but 2129 following ($\approx 93:1$) and 146 tweets (probability 0.93), an even stronger asymmetry than is typically observed in legitimate “follow-back” accounts. The third is completely dormant accounts (258 cases, 7.7%; no published tweets). Together, the three rules tag 756 of the 3337 false positives ($\approx 23\%$); the remaining false positives do not match any of the three rules. These accounts exhibit surface-level behavioral traits commonly associated with automated accounts, such as sparse engagement, skewed follow patterns, and incomplete profiles, yet belong to legitimate but passive human users.

For the false negatives, that is, bot accounts misclassified as humans, the most prevalent rule-tagged pattern involves bots that have cultivated mature, human-like digital

identities, consistent with the known presence of highly deceptive bots in the TwiBot-22 benchmark [24]. We identify three rule-based sub-patterns. The first is long-lived high-activity identities (3455 cases, 51.9% of the false negatives; account age ≥ 300 days, description ≥ 20 characters, and followers ≥ 100), for example an account registered for about 10 years with 86,291 followers, 78,693 tweets, a detailed 158-character biography, and 1406 list memberships (probability 0.0008), indistinguishable from a genuine influencer by any single feature. The second is viral-account mimicry (2 cases; followers $\geq 100,000$ with at most 50 tweets), rare but instructive, for example an account with 505,918 followers but only 31 tweets and an 8.5-year history (probability 0.05). The third is identity-mimicking descriptions (543 cases, 8.2%; descriptions containing identity keywords such as “news” or “official”), for example accounts posing as news outlets (“Leading Lebanese portal offering Live & Real Time News from Lebanon and the Middle East”, 162,214 followers, 683,396 tweets) or presenting consistent professional personas (writers, museum curators). The tagged sub-patterns together cover $\approx 60\%$ of the false negatives before de-duplication. These frequencies establish deep camouflage strategies—long account histories, elaborate profiles, sustained high-volume activity, and integration into community structures—as the largest identifiable group among the missed accounts in this sample, while the rule-based tagging remains descriptive rather than a validated error taxonomy; they further suggest that reliably detecting such accounts may require evidence beyond per-account features (e.g., synchronization across accounts).

5.2. Computational Complexity

Table 11 reports the per-component time and space complexity with the following notation: N , number of users; $K = 10,000$, top active users selected for the behavior similarity graph (Section 3.3); $|\mathcal{E}_r|$ and $|\mathcal{E}_b|$, edges of the social relation and behavior similarity graphs; d and d_h , raw and hidden feature dimensions; d_b , behavioral feature dimension; R , number of relation types; L , number of GNN layers.

Table 11. The computational complexity.

Component	Time Complexity	Space Complexity
Feature extraction	$O(Nd)$	$O(Nd)$
Social relation graph	$O(\mathcal{E}_r)$	$O(\mathcal{E}_r + Nd_h)$
Behavior similarity graph	$O(K^2 d_b)$	$O(K^2 + \mathcal{E}_b)$
RGCN forward pass	$O(LR \mathcal{E}_r d_h)$	$O(Nd_h)$
GAT forward pass	$O(L \mathcal{E}_b d_h)$	$O(Nd_h)$
Output MLP	$O(Nd_h)$	$O(Nd_h)$

Per-epoch training time is dominated by the two GNN forward passes; behavior-graph construction is instead a one-time preprocessing cost of $O(K^2 d_b)$. The space figures cover per-layer activations only; training additionally stores intermediate activations for backpropagation, hence the 23.3 GB peak memory.

5.3. Limitations

Beyond the component-level findings quantified in Sections 4.4–4.5, five limitations should be noted.

1. Behavior-graph coverage. To keep the quadratic pairwise similarity computation tractable, the behavior similarity graph is built only over the top-10,000 active users. Under this coverage the behavior branch’s global contribution is statistically undetectable, and real behavioral edges do not outperform degree-preserving random edges in global metrics. This is a genuine limitation, and we do not claim a global gain. What the branch does provide is a localized, diagnosable, and calibratable effect

on covered users (Section 4.5): it changes their decision geometry, and a validation-only subgroup threshold converts this into a substantial covered-stratum macro-F1 improvement without retraining. Extending coverage and introducing non-node-derivable behavioral signals are therefore the binding constraints on turning this conditional value into a global one.

2. Protocol-bound cross-paper comparison. Because no official TMTM implementation is available, a strictly matched comparison is limited to the rerun BotRGCN; all other baselines, including TMTM, are literature-reported under related but non-identical protocols, so cross-paper margins are read as competitiveness rather than established superiority (Section 4.3).
3. Architectural constraints. The GAT branch uses a single attention head due to GPU-memory limits on the 1M-node graph, which may limit its expressive power; and the similarity-threshold quantile ($q = 0.8$) is fixed rather than learned.
4. Static, non-adversarial setting. Our evaluation assumes a static setting, yet recent work shows that GNN-based bot detectors are fragile under realistic, constraint-aware perturbations [29]; the dual-graph model inherits this vulnerability.
5. Coordination semantics. The edges of the behavior similarity graph encode behavioral affinity—similarity in follower/following ratio, post-type distribution, and engagement—which is at most a proxy for, not direct evidence of, coordinated behavior. This paper evaluates bot classification, not coordination detection, and the agentic-network generation integration (Section 5.4) is a prospective application rather than an evaluated capability.

5.4. Future Work

1. Scaling the behavior graph and enriching behavioral evidence. Approximate nearest-neighbor methods such as HNSW [30], LSH, or random-projection trees can reduce construction cost to near-linear while extending coverage from the top-10,000 users to the entire network, and can support periodic reconstruction from streaming activity. Future work can proceed in two parts: first, incorporating temporal extensions (posting-hour entropy, inter-tweet intervals, daily posting rate) into temporal activity sequences; second, since current behavioral features are deterministic functions of the input features, the branch carries no information beyond node features at any coverage. Therefore, achieving global gains requires both higher coverage and behavioral signals that are not derivable from per-node features, such as platform-level interaction timing or cross-account content synchronization.

2. Jointly calibrated multi-group operating points. The degree-aware threshold remedy of Section 4.5 calibrates a subgroup threshold on validation data only. A principled extension is a jointly calibrated multi-group scheme—optimizing per-stratum thresholds simultaneously under global accuracy and fairness constraints—and an end-to-end treatment in which the decision threshold and the graph-construction quantile are learned parameters.

3. Integration with agent social network generation. A further direction is to embed the proposed detector into a “Generate–Detect–Explain–Refine” loop for agent social network generation [8]. Our model can supply two distinct types of evidence: the RGCN branch indicates whether a generated relational structure deviates from known coordination patterns, and the behavior branch indicates whether behavioral synchronization is anomalous. Converting these signals into re-prompts for LLM agents, and evaluating whether the resulting networks become harder to distinguish from real ones, is a natural extension.

4. Adversarial robustness and instance-level explanation. Future work will evaluate the dual-graph model against constraint-aware attacks such as BOCLOAK [29] and

explore defenses including adversarial training, graph structure learning, and randomized smoothing. On the explanation side, the present analysis covers feature groups, individual hand-crafted features, and subgroup-level structural diagnosis; instance-level structural attribution—tracing a single prediction back to the specific social relations, behavioral neighbors, and feature groups that drove it, via methods such as GNNExplainer or SubgraphX [31]—remains future work and would support human-in-the-loop moderation and regulatory compliance.

6. Conclusions

In this paper, we proposed a dual-branch graph neural network for social bot detection on TwiBot-22, in which a heterogeneous social relation graph processed by an RGCN is coupled with a sparse auxiliary behavior-similarity graph processed by a GAT. Under a unified evaluation protocol—five-seed stratified splits, class-weighted training, validation-only model selection, and per-seed threshold calibration—the model achieves 89.48 ± 0.07 accuracy and 78.11 ± 0.14 macro-F1 at the calibrated operating point. The component-level attribution is unambiguous: the BERTweet text embedding is the largest single contributor, the relation branch supplies the bulk of the graph signal, and the behavior branch's global marginal contribution lies within noise at its current coverage. A degree-stratified analysis shows that the behavior branch is nonetheless functionally active exactly on the users its graph covers, where a validation-only subgroup threshold lifts the covered-stratum macro-F1 from 51.85 to 67.17; together with the degree-preserving random-graph control, this locates the null global effect at the coverage ceiling rather than in the formulation. We therefore frame the behavior graph as a sparse auxiliary view rather than a co-equal branch, and identify coverage scaling via approximate nearest-neighbor methods, together with behavioral signals not derivable from node features, as the principal directions for strengthening it (Section 5.4). The subgroup calibration protocol introduced here provides a reusable deployment-time mechanism for realizing the branch's value wherever behavioral coverage exists.

Author Contributions: Conceptualization, G.S. and F.L.; methodology, G.S. and S.W.; software, S.W.; validation, G.S. and L.W.; formal analysis, G.S. and S.W.; investigation, S.W.; resources, Q.W. and L.W.; data curation, G.S.; writing—original draft preparation, G.S. and F.L.; writing—review and editing, G.S.; visualization, S.W.; supervision, L.W.; project administration, Q.W.; funding acquisition, Q.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Fundamental Research Funds for the Central Universities (Grant NO. CUC26TD02, Grant NO. CUC26AI05, CUC24WQ18).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The TwiBot-22 dataset used in this study is publicly available and can be accessed via the official project website <https://twibot22.github.io/> (accessed on 11 August 2026)

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ezzeddine, F.; Ayoub, O.; Giordano, S.; Nogara, G.; Sbeity, I.; Ferrara, E.; Luceri, L. Exposing influence campaigns in the age of LLMs: A behavioral-based AI approach to detecting state-sponsored trolls. *EPJ Data Sci.* **2023**, *12*, 46.
2. Zhang, P.; Du, Y.; Wang, Q.; Zhang, J.; Qin, R. Research on social bot identification through behavioral feature analysis. *PLoS ONE* **2025**, *20*, e0324539.
3. Yang, K.C.; Menczer, F. Anatomy of an AI-powered malicious social botnet. *J. Quant. Descr. Digit. Media* **2024**, *4*, 7.
4. Yang, T.; Hu, Y.; Lu, F.; Zhang, Z.; Sun, Q.; Li, J. BotUmc: An uncertainty-aware Twitter bot detection with multi-view causal inference. *arXiv* **2025**, arXiv:2503.03775.
5. Yang, K.C.; Varol, O.; Hui, P.M.; Menczer, F. Scalable and generalizable social bot detection through data selection. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020.
6. Huang, D.; Song, J.; Zhang, X.; Yang, W. A social bot detection framework based heterogeneous graph attention and similarity graph features. In Proceedings of the 2nd International Conference on Computer, Nanchang, China, 27–29 December 2024.
7. Tan, Z.; Feng, S.; Sclar, M.; Wan, H.; Luo, M.; Choi, Y.; Tsvetkov, Y. BotPercent: Estimating bot populations in Twitter communities. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, 6–10 December 2023; pp. 14295–14312.
8. Yan, H.; Chen, L.; Zhang, B.; Liu, H.; Peng, H.; Li, S.; Bai, Y.; Qu, Z. Agentic LLMs for Social Network Generation using Explainable Adversarial Re-prompting. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD 2026)*, Jeju, Republic of Korea, 9–13 August 2026; ACM: New York, NY, USA, 2026.
9. Luceri, L.; Deb, A.; Badawy, A.; Ferrara, E. Red bots do it better: Comparative analysis of social bot partisan behavior. In Proceedings of the Companion Proceedings of the Web Conference, San Francisco, CA, USA, 13–17 May 2019.
10. Dukić, D.; Keča, D.; Stipić, D. Are you human? Detecting bots on Twitter using BERT. In Proceedings of the IEEE 7th International Conference on Data Science and Advanced Analytics, Sydney, Australia, 6–9 October 2020.
11. Arin, E.; Kutlu, M. Deep learning based social bot detection on Twitter. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 1763–1772.
12. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 2–7 June 2019.
13. Nguyen, D.Q.; Vu, T.; Nguyen, A.T. BERTweet: A pre-trained language model for English Tweets. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*; Association for Computational Linguistics: Kerrville, TX, USA, 2020; pp. 9–14.
14. Wu, Q.; Yang, Y.; He, B.; Liu, H.; Yang, R.; Liao, Y. BotSCL: Heterophily-aware social bot detection with supervised contrastive learning. In *Pattern Recognition: 27th International Conference, ICPR 2024, Kolkata, India, 1–5 December 2024, Proceedings, Part VII*; Springer: Cham, Switzerland, 2024; pp. 53–68.
15. Zeng, F.; Sun, Y.; Li, Y. MRLBot: Multi-dimensional representation learning for social media bot detection. *Electronics* **2023**, *12*, 2298.
16. Guyan, Q.; Liu, Y.; Zhang, P. PEGNN: Peripheral-enhanced graph neural network for social bot detection. *Expert Syst. Appl.* **2025**, *263*, 127294.
17. Wan, H.; Luo, M.; Wang, J.; Chang, X.; Zheng, Q. HCGBot: Learning homophilous context graphs for Twitter bot detection. *IEEE Trans. Knowl. Data Eng.* **2026**, *3*, 1813–1825.
18. Arranz-Escudero, O.; Quijano-Sanchez, L.; Liberatore, F. Enhancing misinformation countermeasures: A multimodal approach to Twitter bot detection. *Soc. Netw. Anal. Min.* **2025**, *15*, 14.
19. Zhou, M.; Zhang, D.; Wang, Y.; Geng, Y.; Dong, Y.; Tang, J. LGB: Language model and graph neural network-driven social bot detection. *IEEE Trans. Knowl. Data Eng.* **2025**, *37*, 4728–4742.
20. Wang, T.; Wang, Z.; Li, H.; Xia, C.; Zhao, C. HHG-Bot: A hyperheterogeneous graph-based Twitter bot detection model. *IEEE Trans. Comput. Soc. Syst.* **2025**, *12*, 3416–3430.
21. Liu, Q.; Zhang, W.; Lu, J. MB-HGTBot: Modeling multitype social behaviors in heterogeneous graphs for robust Twitter bot detection. *IEEE Trans. Comput. Soc. Syst.* **2026**, *13*, 6174–6195.
22. Meng, X.; Wang, X.; Peng, T.Q. Navigating ambiguities: A systematic review and comparative analysis of social bot detection methods in communication research. *Commun. Methods Meas.* **2026**, *20*, 103–132.
23. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph attention networks. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018), Vancouver, BC, Canada, 30 April–3 May 2018.

24. Feng, S.; Tan, Z.; Wan, H.; Wang, N.; Luo, M. TwiBot-22: Towards graph-based Twitter bot detection. In Proceedings of the Advances in Neural Information Processing Systems 35 (NeurIPS 2022), New Orleans, LA, USA, 28 November–9 December 2022; pp. 35254–35269.
25. Feng, S.; Wan, H.; Wang, N.; Li, J.; Luo, M. TwiBot-20: A comprehensive Twitter bot detection benchmark. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM '21), Virtual Event, 1–5 November 2021; pp. 4485–4494.
26. Schlichtkrull, M.; Kipf, T.N.; Bloem, P.; van den Berg, R.; Titov, I.; Welling, M. Modeling relational data with graph convolutional networks. In Proceedings of the Semantic Web: 15th International Conference, ESWC 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 593–607.
27. Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. In Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May 2019.
28. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010), Sardinia, Italy, 13–15 May 2010; pp. 249–256.
29. Mukherjee, K.; Alom, Z.; Ngo, T.G.B.; Akcora, C.G.; Kantarcioglu, M. Optimal transport-guided adversarial attacks on graph neural network-based bot detection. *arXiv* **2026**, arXiv:2602.00318.
30. Johnson, J.; Douze, M.; Jégou, H. Billion-scale similarity search with GPUs. *IEEE Trans. Big Data* **2021**, *7*, 535–547.
31. Baghershahi, P.; Fournier, G.; Nyati, P.; Medya, S. From Nodes to Narratives: Explaining Graph Neural Networks with LLMs and Graph Context. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: San Diego, CA, USA, 2026.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.