BotSpot++: A Hierarchical Deep Ensemble Model for Bots Install Fraud Detection in Mobile Advertising

YADONG ZHU, XILIANG WANG, QING LI, and TIANJUN YAO, Mobvista Inc.
SHANGSONG LIANG, Sun Yat-sen University

Mobile advertising has undoubtedly become one of the fastest-growing industries in the world. The influx of capital attracts increasing fraudsters to defraud money from advertisers. Fraudsters can leverage many techniques, where bots install fraud is the most difficult to detect due to its ability to emulate normal users by implementing sophisticated behavioral patterns to evade from detection rules defined by human experts. Therefore, we proposed BotSpot[1] for bots install fraud detection previously. However, there are some drawbacks in BotSpot, such as the sparsity of the devices' neighbors, weak interactive information of leaf nodes, and noisy labels. In this work, we propose BotSpot++ to improve these drawbacks: (1) for the sparsity of the devices' neighbors, we propose to construct a super device node to enrich the graph structure and information flow utilizing domain knowledge and a clustering algorithm; (2) for the weak interactive information, we propose to incorporate a self-attention mechanism to enhance the interaction of various leaf nodes; and (3) for the noisy labels, we apply a label smoothing mechanism to alleviate it. Comprehensive experimental results show that BotSpot++ yields the best performance compared with six state-of-the-art baselines. Furthermore, we deploy our model to the advertising platform of Mobvista,[2] a leading global mobile advertising company. The online experiments also demonstrate the effectiveness of our proposed method.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Information systems** → **Online advertising;**

Additional Key Words and Phrases: Mobile ad fraud, app install fraud, bots fraud detection, graph neural network

ACM Reference format:
Yadong Zhu, Xiliang Wang, Qing Li, Tianjun Yao, and Shangsong Liang. 2021. BotSpot++: A Hierarchical Deep Ensemble Model for Bots Install Fraud Detection in Mobile Advertising. *ACM Trans. Inf. Syst.* 40, 3, Article 50 (November 2021), 28 pages.
https://doi.org/10.1145/3476107

[1]This work is based on our previous work published at CIKM 2020, which serves the same purpose of detecting bots installs in mobile advertising.
[2]https://mobvista.com/.

Authors' addresses: Y. Zhu, X. Wang, Q. Li, and T. Yao, Mobvista Inc. Beijing, China; emails: {yadong.zhu, xiliang.wang, qing.li, bertrand.yao}@mobvista.com; S. Liang, Sun Yat-sen University, Guangzhou, China; email: liangshangsong@gmail.com.

1046-8188/2021/11-ART50 $15.00
https://doi.org/10.1145/3476107

1 INTRODUCTION

Mobile advertising marketing is being expanded in higher velocity, which leads to continued climbing of mobile app install fraud. Mobile app install fraud implications trickle down to every aspect of mobile advertising marketing initiatives. The most obvious implication is the direct financial loss fraud creates. According to Scalarr.io's latest mobile app install fraud report [39], Scalarr.io's projections for 2020 estimate losses close to the $16.1 billion mark as the result of mobile ad fraud. In addition, mobile app install fraud can ruin the data accuracy and lead advertisers and ad networks to investing and reinvesting in bad publishers (channels) due to the pollution of data being analyzed.

In the context of mobile app install fraud, the advertised product is an app, and the associated ad campaigns for this product are distributed to one or several ad channels. An ad channel (publisher) can claim credit from the advertiser if a user clicks the ads from this ad channel and installs the app, given that the **Mobile Measurement Partner (MMP)** attributes this user install to this ad channel correctly. Furthermore, an ad channel could be a website or a mobile app, or it could be a proxy of several apps. A fraudster, oftentimes cooperating with malicious ad channels, may leverage certain tricks to confuse the attribution system such that the install is falsely attributed and the malicious ad channel collects money from advertisers without subsequent user actions.

As illustrated in Figure 1, a general mobile ad workflow consists of the following steps:

(1) An ad request is sent from the user of some publisher to the ad server of the ad network.
(2) The user clicks on an ad that appears on their mobile device.
(3) The impression and click will be registered with the ad network who is responsible for the ad's placement, and the user will be redirected to the appropriate app store based on their device's **operating system (OS)**.
(4) The impression and click will also be recorded with the MMP, an independent third-party platform connecting between advertisers and publishers), such as Adjust[3] and Appsflyer,[4] for install attribution and further analysis.
(5) The user reaches the app store and downloads the app.
(6) The user launches the app for the first time, and the related metadata is sent to the MMP.
(7) The credit for the app's install will be attributed to the appropriate ad network by the MMP attribution algorithm.

There are many common types of mobile app install fraud, such as click injection, click spamming, device farm, and bots, among which bots install fraud is the most numerous and difficult to detect. According to the latest report of Appsflyer, they have blocked more than 1.6 billion fraud installs over the past 3 years, and the fraud installs resulted by bots accounts for more than 56% of the entire mobile ad install fraud activities [1].

Therefore, we focus on bots install fraud detection in this article. Essentially, bots are malicious codes that run a set of program or action. The bots aim to send clicks, installs, and in-app events for installs that never truly occurred to make a profit. The bots fraud detection in mobile advertising is a challenging task, which is because bots can simulate user behavior with such accuracy that they evade traditional detection methods.

Impression fraud detection and click fraud detection have been extensively studied in the context of both web advertising and mobile advertising, yet very few works solve bots install fraud detection in the context of mobile advertising. BotSpot [54] is proposed to tackle this problem by

[3] https://www.adjust.com/.
[4] https://www.appsflyer.com/.

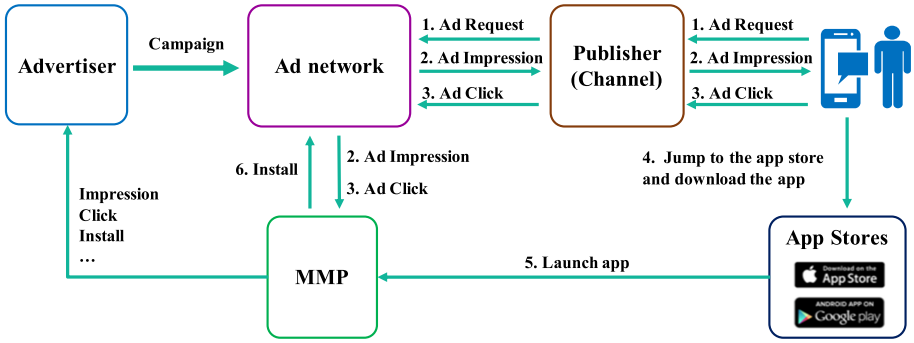

Fig. 1. Mobile ad workflow. In the context of mobile advertising, the main contributors of the mobile advertising market are *advertisers*, *publishers*, *advertising networks (ad networks)*, and the *MMP*. The advertiser is often an app or brand looking to spread a specific message about their product (e.g., new user acquisition and retargeting campaigns). The publisher is the one who provides the capability and inventory that allows advertisers to run ads in their apps or on mobile sites. Ad networks act as a technical and commercial intermediary between advertisers and publishers, such as Mobvista, a leading global mobile advertising company. The MMP is an independent third-party platform connecting between advertisers and publishers.

leveraging local context from a heterogeneous network and global context from the GBM part. But BotSpot suffers from some problems: (1) sparsity of device nodes' neighbors: most device nodes have only one or rare neighbors, which cause the graph convolution layer to not be applied on device nodes' neighbors directly; (2) weak interactive information: the local context is offered from averaging the leaf embeddings without considering the interactive information between different leaf embeddings; and (3) noisy labels: there may be noisy in labels because they are obtained from the third party.

To address the preceding problems, we propose a novel hierarchical deep ensemble model for bots install fraud detection. Our main contributions can be summarized as follows:

(1) We propose to leverage domain knowledge and a clustering algorithm for alleviating the sparsity of the device's neighbors, which benefits the information flow for better representation learning, where similar scenarios may apply. The experiments demonstrate the effectiveness of our proposed method.
(2) We resort to a multi-head self-attention module to enhance the interaction among various leaf nodes, which is more adequate to retrieve knowledge from leaf nodes compared with the simple average operation.
(3) Label smoothing is leveraged to prevent the model from overconfident prediction, as our labels are obtained from the feedback of advertisers and a third party, and the labels may be not always correct.
(4) Extensive offline and online experiments demonstrate that our proposed method can detect more bots installs with the threshold found in the validation dataset when models' precision is fixed to a certain threshold such as 90%, 85%, or 80%.
(5) We have open sourced our codes and released three datasets collected from the advertising platform of Mobvista.[5]

[5]https://github.com/mobvistaresearch/BotSpot-Plus.

The rest of the article is organized as follows. We first introduce the related work in Section 2. Second, we describe our proposed method in Section 3. Then we describe our experimental settings and analyze the results in Section 4. Next, we discuss the model convergence problem first, then we describe several limitations of our proposed model and outline how these might be overcome in future work in Section 5. Finally, we make some conclusions about our work in Section 6.

2 RELATED WORK

In this section, we briefly discuss two lines of related work: fraud detection in mobile advertising and graph-based fraud detection.

2.1 Fraud Detection in Mobile Advertising

To detect mobile advertising fraud behaviors, such as impression fraud and click fraud, some machine learning methods have been successfully applied. Haider et al. [15] proposed an ensemble-based method (e.g., J48 [36], Random Forest [2]) to classify each impression as fraudulent or non-fraudulent. To handle the issue of an imbalanced dataset, they also applied SMOTE [4]. Perera et al. [33] proposed a novel ensemble model that is based on six different learning algorithm approaches for click fraud detection in mobile advertising. Zhang et al. [57] proposed a combination of a cost-sensitive back-propagation neural network (CSBFNN) and the novel artificial bee colony [20] algorithm in their research toward detection of click fraud. They optimized the feature selection process with BPNN connection weights using the artificial bee colony to alter the relation between features and weights. Oentaryo et al. [31] found that the features derived from fine-grained time-series analysis are crucial for accurate fraud detection, and that ensemble methods offer promising solutions to highly imbalanced nonlinear classification tasks with mixed variable types and noisy/missing patterns. Taneja et al. [43] proposed a novel framework for prediction of click fraud in mobile advertising that consists of feature selection using recursive feature elimination and classification through Hellinger distance decision tree. They chose recursive feature elimination for the feature selection, as it has given better results as compared to the wrapper approach when evaluated using different classifiers. To deal with the class imbalance issue present in the dataset, the use the Hellinger distance decision tree as as classifier. Dou et al. [9] investigated the problem of download fraud in the App Market by setting up a honeypot to capture the ground truth of fraudulent activities, and identify the download fraud leveraging feature engineering and XGBoost [5].

For the install fraud detection in mobile advertising, inspired by the work of Li et al. [24], Yao et al. [54] propose an anti-fraud method based on a heterogeneous graph that incorporates both local context and global context via **graph neural networks (GNNs)** and a gradient boosting classifier to detect bots installs at Mobvista.

2.2 Graph-Based Fraud Detection

Inspired by the work of Mikolov et al. [29], various forms of graph embedding methods (e.g., Node2Vec [14], DeepWalk [34], and LINE [44]) have been proposed to extract the node embeddings from a graph without requiring any labels. However, node features are not utilized. The GNN is further presented, in which the limitations of graph embedding are compensated. One of the most prominent progresses is known as the **graph convolutional network (GCN)** [22], which exploits multiple propagation layers to aggregate neighborhood information and enlarge the receptive field for each node in the graph. The GCN achieves significant improvements compared to previous graph-mining methods such as DeepWalk. Hamilton et al. [16] proposed GraphSAGE, an inductive framework that leverages node sampling and feature aggregation techniques to efficiently generate node embeddings for unseen data, which breaks the limitation of applying the GCN in transductive settings. The graph attention network (GAT) [46] further introduces the

masked self-attentional layers [8] into a GCN. By leveraging masked self-attentional layers, the GAT can assign different importance to different nodes within a neighborhood.

Wang et al. [47] presented a graph-based spam detection method to identify suspicious reviewers. To detect the spam reviews at Xianyu (a second-hand goods app), Li et al. [24] proposed GAS, which is based on GCNs. In GAS, a heterogeneous graph and a homogeneous graph are integrated to capture the local context and global context of a comment. Xu et al. [52] proposed FeatNet, which incorporates the device features and device relationships in network representation learning to detect fraud devices. Breuer et al. [3] studied the problem of early detection of fake user accounts on social networks. Nguyen et al. [30] proposed FANG, a graph learning framework that enhances representation quality by capturing the rich social interactions between users, articles, and media, thereby improving both fake news detection and source factuality prediction. For detecting malicious accounts at Alipay, inspired from a connected subgraph approach, Liu et al. [27] proposed GEM. They summarize two fundamental weaknesses of attackers, namely "Device aggregation" and "Activity aggregation," and naturally present a neural network approach based on heterogeneous account-device graphs. They also proposed an attention mechanism to learn the importance of different types of nodes while using the sum operator for modeling the aggregation patterns of nodes in each type. To identify the fraudulent apps in mobile advertising, Hu et al. [18, 19] proposed two models, namely iBGP and GFD, respectively. Liu et al. [28] introduced three inconsistency problems: context inconsistency, feature inconsistency, and relation inconsistency. They proposed the GraphConsis model to tackle these inconsistency problems: (1) for the context inconsistency, they proposed to combine the context embeddings with node features; (2) for the feature inconsistency, they designed a consistency score to filter the inconsistent neighbors and generate corresponding sampling probability; and (3) for the relation inconsistency, they learned the relation attention weights associated with the sampled nodes. Dou et al. [10] proposed CARE-GNN to solve the feature camouflage and relation camouflage so as to avoid the camouflage behavior of fraudsters harming the model's performance. They first devise a label-aware similarity measure to find informative neighboring nodes. Then they leverage reinforcement learning to the optimal amounts of neighbors to be selected. Finally, the select neighbors across different relations are aggregated together. Schlichtkrull et al. [40] introduced **relational graph convolution networks (R-GCNs)** and applied them to standard knowledge base completion tasks. R-GCNs are related to a recent class of neural networks operating on graphs, and are developed specifically to deal with the highly multi-relational data characteristic of realistic knowledge bases. Furthermore, Wang et al. [48] proposed a collaboration based multi-label propagation (CMLP) algorithm to detect fraud users in e-commerce platforms. They introduced a generic version that involves a collaboration technique to exploit label correlations. Then, to accelerate it on large-scale e-commerce data, they also proposed a heterogeneous graph-based variant that detects communities on the user-item graph directly.

3 PROPOSED METHOD

In this section, let us first introduce the main notations in the article. We use bold uppercase letters (e.g., $\mathbf{X}$) and bold lowercase letters (e.g., $\mathbf{x}$) to represent matrices and vectors, respectively. We use non-bold uppercase letters to represent a set. We employ non-bold letters (e.g., x) to represent scalars. We use $func\_name(\bullet)$ to represent a function. We summarize the main notations in Table 1.

Next, we describe the model architecture of our proposed BotSpot++ from the top level. As shown in Figure 2, our BotSpot++ concatenates the heterogeneous GNN information and pretrained GBM model information to classify the samples.

Table 1. Summary of the Main Notations

Notation	Explanation
$G(V, E)$	A graph with the set of vertexes and the set of edges, shortened to G
$BiG((D, C), E)$	A bipartite graph with two sets of nodes (D, C) and a set of edges (E)
C	The set of channel-campaign nodes
S	The data samples
$n^{classes}$	The number of classes
n^{trees}	The number of weak tree learners
α	The hyperparameter that controls the smoothness degree
$\mathbf{y}_i$	The label for sample i
$\bar{\mathbf{y}}_i$	The smoothed label for sample i
l	The l^{th} layer of a neural network
$NS(\bullet)$	A function for sampling neighbors
$NA(\bullet)$	A differentiable function for node aggregation
$SN(\bullet)$	A function to get a set of device nodes given a super device node
D^{origin}	The set of single device nodes
D^{super}	The set of super device nodes
$\mathbf{M}^{embed}$	The set of leaf embeddings (embedding matrix)
$S^{minibatch}$	The samples in a minibatch of S
$\mathbf{H}^{(l+1)}$	The $(l+1)^{th}$ hidden layer in a neural network
$\mathbf{W}^{(l)}$	The weight matrix of the l^{th} hidden layer in a neural network
$\tilde{\mathbf{D}}$	A diagonal matrix in graph G with a self-loop
$\mathbf{x}_d^{origin}$	The input features of origin device node d, $\forall d \in D$
$\mathbf{x}_c$	The input features of channel-campaign c, $\forall c \in C$
$\mathbf{h}_v^{l-1}$	The vector representation for vertex v at the $(l-1)^{th}$ layer
$\mathbf{W}^{bots,l}$	The weight matrix of a linear layer of bots devices at the l^{th} layer
$\mathbf{W}^{normal,l}$	The weight matrix of a linear layer of normal devices at the l^{th} layer
$\mathbf{s}_v^{bots,l}$	The attention scores of bots installs for vertex v at the l^{th} layer
$\mathbf{s}_v^{normal,l}$	The attention scores of normal install for vertex i at the l^{th} layer
$\lambda_v^{t,l}$	The attention score over $\mathbf{s}_v^{normal,l}$ and $\mathbf{s}_v^{bots,l}$, where $t \in \{bots, normal\}$
$\mathbf{h}_i^{super}$	The convolved vector representation for super device node i
$\mathbf{h}_{i,q}^{origin}$	The representation of origin device node i in super device node q
FFN_θ	2-layer feed forward network with parameters θ

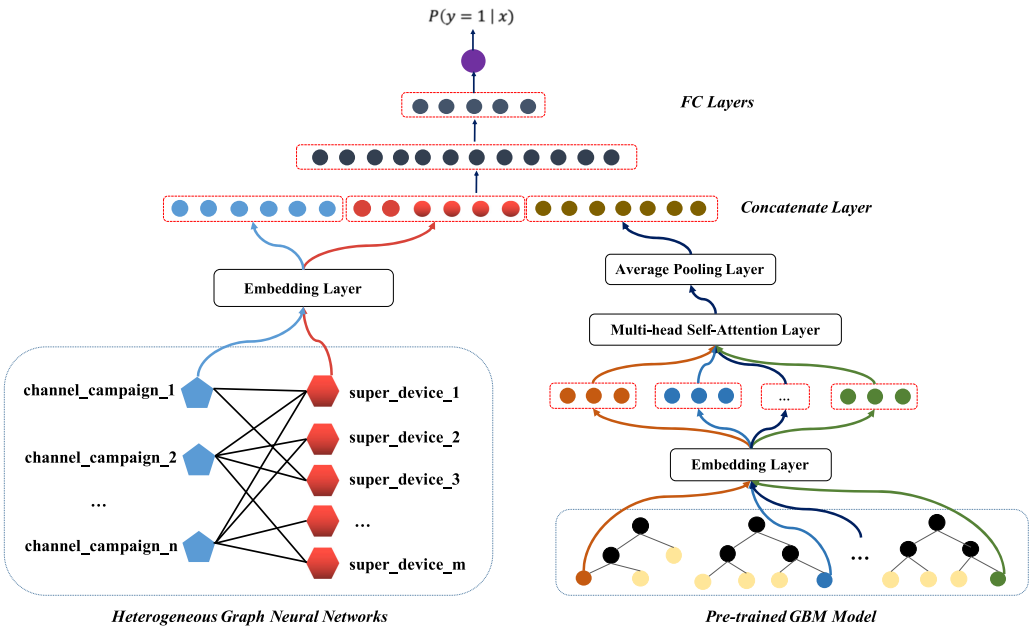

Fig. 2. Model architecture of BotSpot++.

3.1 Preliminary

The GNN is introduced in the work of Gori et al. [13] as a generalization of recursive neural networks that can directly deal with a more general class of graphs (e.g., cyclic), directed and undirected graphs. In the work of Estrach et al. [11], a model based on a spectrum of graph Laplacian is proposed, in which the model directly learns the representation of the localized linear filter. However, the model may suffer from overfitting and is computationally expensive due to its requirement of decomposition on the Laplacian matrix. Defferrard et al. [6] introduced a GNN with polynomial parameterization for the localized filter to reduce the number of model parameters to alleviate overfitting and reduce computational cost. Kipf and Welling [22] further refined the model discussed earlier by proposing a layer-wise linear model where the degree of the Chebyshev polynomial is limited to 1. However, by stacking multiple linear layers with non-linearity, a rich class of convolutional filters can still be recovered. To stabilize the training process in the GCN [22], the authors further proposed to add a self-loop to keep the eigenvalue of the graph matrix less than 1, hence avoiding the issue of gradient explosion. The multi-layers propagation rule in the GCN is shown in Equation (1), where $\mathbf{H}^{l+1}$ is the node representation at the $(l+1)^{th}$ layer, $\tilde{\mathbf{A}}$ is the adjacency matrix with a self-loop, $\tilde{\mathbf{D}}$ is the diagonal matrix in graph G with a self-loop, and $\mathbf{W}^l$ is the graph convolutional filter at the l^{th} layer.

$$\mathbf{H}^{l+1} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{l}\mathbf{W}^{l}\right) \tag{1}$$

Hamilton et al. [16] further proposed a more generalized version of GNNs based on the work of Kipf and Welling [22], where a computational subgraph is built for every vertex in an inductive manner and node representations can be extracted in a bottom-up approach leveraging various kinds of message passing mechanisms such as Pooling or LSTM-based aggregation. The propagation rule for GraphSAGE is shown in Equation (2), where $NS(\bullet)$ is the neighboring function and

$\mathbf{h}_v^k$ is the node representation for vertex v at layer k.

$$\begin{aligned} \mathbf{h}_{NS(v)}^k &\leftarrow NA_k\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in NS(v)\right\}\right), \\ \mathbf{h}_v^k &\leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}_v^{k-1}, \mathbf{h}_{NS(v)}^k\right)\right) \end{aligned} \tag{2}$$

Veličković et al. [46] incorporate the attention mechanism and multi-head attention to improve the expressive power of each vertex and achieve a new SOTA on various datasets. Currently, many new deep learning based graph representational learning algorithms have been proposed [12, 26, 49, 51, 56] and are applied in various fields [25, 35, 53, 55]. In this work, we propose a novel algorithm that can be adapted to a sparse graph for bots install fraud detection. The underlying notion is to leverage both global context and local context to facilitate the downstream classification problem.

3.2 Problem Definition

Since there are devices with both normal installs and bots installs, we formulate our problem as an edge classification problem on a sparse graph just as in the work of Yao et al. [54]. The graph *BiG* is a heterogeneous bipartite graph with two sets of nodes, namely device nodes D^{origin} and channel-campaign nodes C. A device node represents a mobile device where the user clicked an ad and downloaded the corresponding app. A channel-campaign node represents an ad campaign presented on an ad channel. An edge between a device node and a channel-campaign node represents an app install by a given device node $d \in D^{origin}$ that is attributed to an ad channel-campaign $c \in C$. Our main task is to classify every edge $e_{dc}(d \in D^{origin}, c \in C)$ into a normal edge or a bots fraud edge.

3.3 Label Acquisition

In this work, we gather the feedback of whether an app install is a normal install or a bots install from the advertisers or third party to annotate each data sample, and hence our model can be trained in a fully supervised manner.

3.4 Motivation

The proposed graph is sparse and unbalanced—that is, the channel-campaign nodes may connect to many device nodes yet the device nodes connect to a single channel-campaign node in most cases. In the work of Yao et al. [54], BotSpot is proposed to tackle this problem by defining the context into two categories: local context and global context. The local context refers to the representation of a channel-campaign node using a tailored message passing scheme to reflect its preference to bots and normal devices, whereas the global context refers to the information offered by the leaf embeddings of leaf nodes predicted by a pre-trained GBM. It is worth mentioning that the leaf embeddings are obtained from the neural network by feeding the leaf nodes to it. By leveraging the local context and global context, BotSpot can classify each edge in the graph with a more holistic view. However, due to the sparsity of the formulated graph, many device nodes merely connect to a single channel-campaign node. As a result, we only employ an FFN layer for feature extraction of device nodes in BotSpot, without incorporating any relational information. Intuitively, applying the message passing mechanism on device nodes will have better representations in the embedding space by being aware of their neighbors. Therefore, in this work, we ask the following question: how do we apply the message passing for device nodes and how do we enhance the interaction of leaf nodes so as to generate a more expressive embedding for every leaf node? In the following work, we propose several modifications to BotSpot, which are aiming to address the problems mentioned earlier.

3.5 Proposed Method

In this section, we detail how to enhance BotSpot in a variety of algorithmic aspects so as to address the modeling issues in BotSpot. Specifically, BotSpot++ gets the local context obtained from message passing for channel-campaign nodes and device nodes with a new graph structure. In addition, as we all know, ensemble methods may obtain better predictive performance by using multiple learning algorithms [38]. Therefore, we ensemble the leaf nodes predicted by GBM with a deep learning model to obtain a better performance, which is namely global context in our work. Now, let us explain how to get the local context and global context in detail.

3.5.1 Message Passing for the Channel-Campaign Node. For the channel-campaign node, we leverage the same message passing mechanism as in BotSpot. Therefore, the explicit normal neighbors and bots neighbors of a channel-campaign node are leveraged separately together with the attention mechanism, which helps the channel-campaign node learn a biased representation toward bots or normal neighbors. Essentially, the normal neighbors and bots neighbors are all device nodes in our scenario.

Mathematically, the tailored message passing mechanism can be expressed as Equation (3), where $\mathbf{h}_v^{bots,l}$ is the obtained node representations for vertex v at the l^{th} layer for bot devices, and $\mathbf{W}^{bots,l}$ is an affine layer for linear transformation of bots devices. Similarly, $\mathbf{h}_v^{normal,l}$ is the node representation for vertex v at layer l for normal devices, and $\mathbf{W}^{normal,l}$ is another set of model parameters for affine transformation. $\mathbf{s}_v^{normal,l}$ and $\mathbf{s}_v^{bots,l}$ are the attentional scores for bots and normal installs, respectively. $\mathbf{q}_v^{l-1}$ denotes the final output embedding for vertex v at the $(l-1)^{th}$ layer, and $\|$ denotes the concatenation operator to concatenate two vectors. $\lambda_v^{t,l}$ is the attention weights after softmax function over $\mathbf{s}_v^{normal,k}$ and $\mathbf{s}_v^{bots,l}$. By performing a linear combination over $\mathbf{h}_v^l$ and $\mathbf{g}_v^l$, the representation over channel-campaign node v is able to capture its preference toward bots and normal installs.

Practically, we generate the normal neighbors N_i and bots neighbors B_i for every channel-campaign node i using the training dataset. Then we sample a fixed number of normal neighbors and bots neighbors from N_i and B_i respectively in the training phase and the test phase. It should be noted that to avoid data leakage, we should exclude the device nodes connected by the edge to be predicted from the normal neighbors or bots neighbors when sampling.

$$
\begin{aligned}
&\mathbf{h}_v^{bots,l} = \mathbf{W}^{bots,l} \bullet NA^l\left(\left\{\mathbf{h}_b^{bots,l-1}\right\}\right), \forall b \in NS(v),\\
&\mathbf{h}_v^{normal,l} = \mathbf{W}^{normal,l} \bullet NA^l\left(\left\{\mathbf{h}_b^{normal,l-1}\right\}\right), \forall b \in NS(v),\\
&\mathbf{z}_v^l = \lambda_v^{bots,l}\mathbf{h}_v^{bots,l} + \lambda_v^{normal,l}\mathbf{h}_v^{normal,l}, \forall v \in D^{origin},\\
&\mathbf{s}_v^{bots,l} = FFN_\theta\left(\mathbf{q}_v^{l-1}\|\mathbf{h}_v^{bots,l}\right),\\
&\mathbf{s}_v^{normal,l} = FFN_\theta\left(\mathbf{q}_v^{l-1}\|\mathbf{h}_v^{normal,l}\right),\\
&\lambda_v^{t,l} = \frac{\exp\left(\mathbf{s}_v^{t,l}\right)}{\sum_{t\in T}\exp\left(\mathbf{s}_v^{t,l}\right)}, \forall t \in T = \{bots, normal\}
\end{aligned}
\tag{3}
$$

3.5.2 Message Passing for the Device Node. In BotSpot, a device node always connects to a single channel-campaign node in most cases. Therefore, the representation of a device node is extracted from its neighbors' information by using a 2-layer MLP, where graph convolution cannot be employed. Actually, it would be better if some relational information could be leveraged for device nodes to extract better representations that are aware of their "neighboring" ad

channel-campaign nodes. In this work, we propose multiple methods to construct super device nodes so that message passing can be applied on the device nodes.

Let us explain the multiple algorithms for constructing super device nodes in detail. First, we have domain-based algorithms. We aggregate the device nodes with the same *package_name* into a super device node, where the *package_name* refers to the app's name promoted by an advertiser. The reason we choose *package_name* as the criterion for device node aggregation is that ad networks typically distribute the same apps to multiple ad channels (i.e., channel-campaign nodes) with "similar" advertising effects and similar audiences. As a result, the device nodes in the same super device node can be linked to multiple "similar" ad channel-campaign nodes. Second, we have clustering-based algorithms. Practically, we found that there are a large number of device nodes in some super device nodes when we use *package_name* to construct super device nodes. It may cause the device nodes in a larger super device node to connect to many and even all channel-campaign nodes. As a result, we apply clustering algorithms to the device nodes in every larger super device node generated by *package_name*. In this way, we can divide every larger super device node into multiple smaller super device nodes. As for the clustering algorithm, we tried K-Means [17], GMM [37], and Mean-Shift [7] for clustering device nodes in every larger super device node. We also conducted a series of experiments to verify the effects of different clustering algorithms, and the specific experimental results are listed in Section 4.

Now, the device nodes are grouped into multiple super device nodes, and thus we can treat each device node in a super device node as connecting to several or dozens of channel-campaign nodes to which the super device node associates. Therefore, we can apply the message passing mechanism to device nodes with the new graph structure. It should be noted that the super device nodes are used to make every device node have multiple edges connected to channel-campaign nodes, which are actually virtual nodes and do not participate in the graph convolution operation.

Mathematically, the message passing mechanism for super device node $q \in D^{super}$ and the device nodes $u \in q$ follows Equation (4),

$$\begin{aligned} \mathbf{h}_q^{super} &= NA\left(x_i\right), \forall i \in NS(q), q \in D^{super}, \\ \mathbf{h}_{u,q}^{origin} &= \mathbf{h}_q^{super} \| \mathbf{W}^{origin} \mathbf{x}_u, \forall u \in SN(q), q \in D^{super}, \end{aligned} \tag{4}$$

where x_i denotes the feature vector of device node i, $\mathbf{h}_q^{super}$ denotes the convolved feature representation of super device node q, $\mathbf{h}_{u,q}^{origin}$ is the node representation of device node u that is inside super device node q, $SN(i)$ is a function that returns a set of device nodes inside super device node i, and $\mathbf{W}^{origin}$ denotes the weight of affine transformation of origin device node u.

The process of forming super device nodes according to various criteria is shown in Figure 3. What needs to be further explained is that the $\mathbf{h}_q^{super}$ is the result of neighbor sampling operation for every device node in super device node q. As for graph convolution operation, we concatenate the $\mathbf{h}_q^{super}$ and $\mathbf{W}^{origin}\mathbf{x}_u$ to get the new representation for the device node u.

By forming super device nodes, the hierarchical graph is built and each device is able to leverage similar channel-campaign nodes to learn more interactive representations, thus benefiting the downstream classification problem.

3.5.3 Global Context Extraction. As in the work of Yao et al. [54], we also extract the global context by leveraging the fixed decision paths from a pre-trained GBM. This is because a fixed decision path in a decision tree corresponds to a series of decision criteria. Thus, we first get the $leaf\_nodes_s$ extracted from the pre-trained GBM model for every sample s by calling the *LEAF_INDEX* function, $s \in S$. It should be noted that the length of $leaf\_nodes_s$ equals to n^{trees}. Then, we feed all $leaf\_nodes_s$ as category features into an embedding layer to get an embedding vector of every leaf

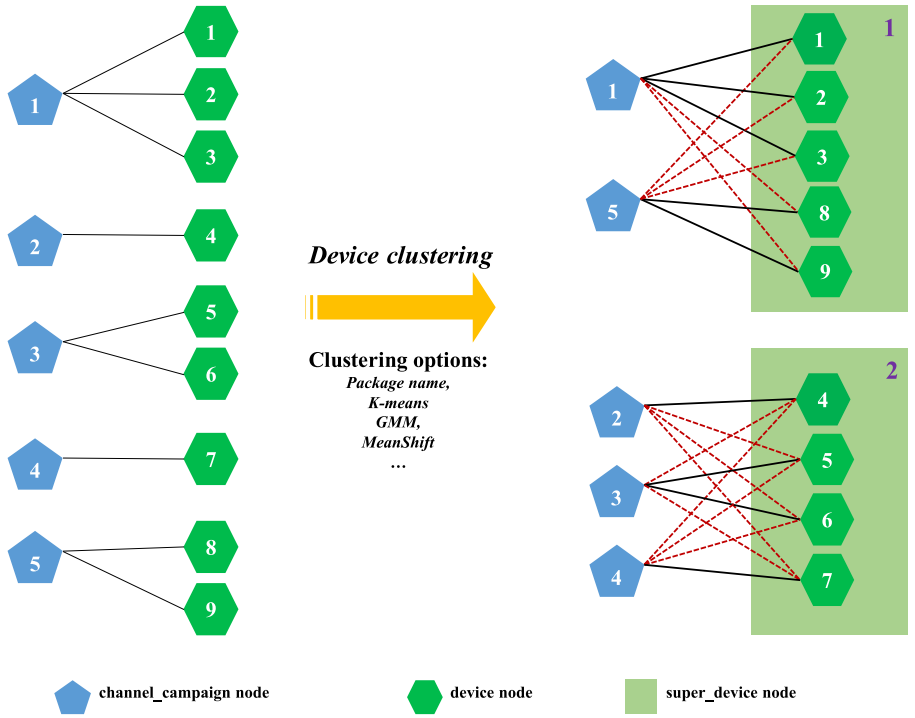

Fig. 3. An illustration of forming super device nodes.

index node. Therefore, we finally get the matrix M^{embed}, where each row represents the vector corresponding to the leaf index. Now, we can get the matrix M_s^{embed} of sample s by concatenating all the vectors of leaf nodes in $leaf\_nodes_s$.

In BotSpot, we apply $Average(\bullet)$ to all leaf embeddings of data sample s directly to get its global context information. However, there are some drawbacks to this method. First, it is a well-known fact that in a gradient boosting machine, the weak learner in the early stage contributes more than those in the later stages, and hence would carry more information, yet a simple mean probably cannot capture this statistical pattern. Second, as leaf embedding for each data sample s corresponds to a fixed decision path in the decision tree, the leaf embeddings M_i^{embed} cannot attend to each other and thus may lead to some sub-optimality. In this work, we employ a self-attention mechanism to obtain a higher-level representation for each leaf node so as to remedy these drawbacks, and hence all the leaf node embeddings attend to the other leaf node embeddings, then the representation after the self-attention module is averaged to get the final aggregated embedding. Equation (5) illustrates the entire process:

$$\begin{aligned} leaf\_nodes_s &= LEAF\_INDEX(s), \forall s \in S, \\ \mathbf{M}_s^{embed} &= \left\{\mathbf{M}_i^{embed}, \forall i \in leaf\_nodes_s\right\}, \\ \mathbf{h}_s^{global} &= AVERAGE(SELF\_ATTENTION(\mathbf{M}_i^{embed}, num\_heads)), \end{aligned} \tag{5}$$

where $leaf\_nodes_s$ denotes the leaf nodes array of the data sample s, $LEAF\_INDEX$ denotes a function that accepts s as input and outputs all its leaf nodes predicted by a pre-trained GBM, M^{embed} is a matrix of all leaf embeddings learned from the neuron network, and $SELF\_ATTENTION$ is a implementation of Devlin et al. [8].

The final representation is obtained by concatenating h_s^{global} and h_s^{local}, which is fed into an FFN module with a dropout mechanism. Finally, the model will output a probability indicating whether it is a bots or normal install as illustrated in Equation (6):

$$P(y = 1 \mid \mathbf{x}_s; \theta) = \sigma\left(\mathbf{W}^2\left(\mathbf{W}^1\left(\mathbf{h}_s^{local} \| \mathbf{h}_s^{global}\right) + b^1\right) + b^2\right), \tag{6}$$

where $\sigma(\bullet)$ is the sigmoid function, θ is the model parameters of our proposed method, and s is a data sample, $s \in S$. $\mathbf{h}_s^{local}$ is the local context information of s, which is the vector derived from the left part of the model architecture shown in Figure 2. Furthermore, $\mathbf{W}^1$, $\mathbf{W}^2$ is the two weight matrix, and b^1, b^2 is the bias variable.

3.5.4 Model Training. We resort to binary cross-entropy as the loss function and optimize the model parameters using Adam [21]. As our label is obtained using the feedback from advertisers and a third party, the label may be noisy. Label smoothing [42] is leveraged to prevent the model from overconfident prediction.

Therefore, we first calculate the smoothed label $\bar{y}$, then calculate the total loss L using the cross-entropy loss function. The mathematical formula is as shown in Equation (7):

$$\begin{aligned} \bar{y}_s &= (1 - \alpha) y_s + \frac{\alpha}{n^{classes}}, \\ L &= -\frac{1}{|S|} \sum_{s \in S} \left(\bar{y}_s \log\left(p_s\right) + \left(1 - \bar{y}_s\right) \log\left(1 - p_s\right)\right), \end{aligned} \tag{7}$$

where y_s is true label of sample s, $\bar{y}_s$ is the smoothed label of sample s, p_s is the probability of the sample s output by the model, α is a hyperparameter that controls the smoothness degree, and $n^{classes}$ is the number of labels in the classification problem (i.e., 2). The pseudo-code of our proposed method is shown in Algorithm 1.

4 EXPERIMENTS

To evaluate the effectiveness of our proposed model, we conducted a series of experimental evaluations on real-world datasets collected from Mobvista's advertising platform. The codes and datasets used in our experiments were publicly accessible,[6] which will be detailed in the next section. As our model is delicately designed for bots install fraud detection, these principle research questions are broken down into the following specific ones to guide our experimental studies.

RQ1. Does our proposed BotSpot++ outperform competitive baseline models?

RQ2. How does the different modules of BotSpot++ affect the overall performance?

RQ3. How does different parameters affect the model performance?

RQ4. How does our proposed BotSpot++ perform compare to state-of-the-art methods in our real online production environment?

RQ5. What is the time complexity of all models?

RQ6. Can we qualitatively and quantitatively evaluate the effect of BotSpot++?

RQ7. May down-sampling the normal installs benefit our model due to unbalanced training data?

4.1 Dataset

To evaluate our proposed model more comprehensively, we built three datasets for different time periods. Each dataset mainly includes device information (device brand, device name, IP, etc.), geographic information (country, city, etc.), campaign information, and channel information, which

[6] https://github.com/mobvistaresearch/BotSpot-Plus.

ALGORITHM 1: BotSpot++ Algorithm

Input: bipartite graph G
the cluster values of device nodes $cluster\_values$
weight matrix for channel-campaign nodes of k^{th} layer W_1^k
weight matrix for device nodes of k^{th} layer W_2^k
weight matrix for channel-campaign node of k^{th} layer W_1^k
Output: the estimated probability of input edge $e_{d,c}$ of being bots install

// construct super device nodes.
$super\_device\_neibrs \leftarrow dict()$

// cluster values are the label generated by domain knowledge based or cluster-based method.
for $edge, v$ in $(G, cluster\_values)$ **do**
  $d, c \leftarrow edge$
  if v not in $super\_device\_neibrs$ **then**
    $super\_device\_neibrs[v] \leftarrow set()$
  end if
  $neibrs \leftarrow GET\_DEVICE\_NEIBRS(d)$
  $super\_device\_neibrs[v].add(neibrs)$
end for
//we only use 1-layer graph convolution layer considering time complexity
$k = 1$
$\mathbf{h}_c^0 \leftarrow \mathbf{x}_c, \forall c \in C$
$\mathbf{h}_d^0 \leftarrow \mathbf{x}_d, \forall d \in D^{origin}$
for $edge, v$ in $G, cluster\_values$ **do**
  $d, c \leftarrow edge$ ▷ d: index of device nodes, c: index of channel-campaign nodes

  //local context information
  $\mathbf{h}_{NS(c)} \leftarrow NA^k(\{\mathbf{h}_u^{k-1}, \forall u \in NS(c)\})$
  $\mathbf{h}_c^k \leftarrow \sigma(CONCAT(\mathbf{W}_1^k \mathbf{x}_c, \mathbf{h}_{NS(c)}))$
  $s \leftarrow super\_device\_neibrs[v]$
  $\mathbf{h}_{NS(s)} \leftarrow NA^k(\{\mathbf{h}_v^{k-1}, \forall v \in NS(s)\})$
  $\mathbf{h}_s^k \leftarrow \sigma(CONCAT(\mathbf{W}_3^k \mathbf{x}_s, \mathbf{h}_{NS(s)}))$
  $\mathbf{h}_d^k \leftarrow CONCAT(\mathbf{W}_2^k \mathbf{x}_d, \mathbf{h}_s^k)$
  $\mathbf{h}_{d,c}^{local,k} \leftarrow CONCAT(\mathbf{h}_d^k, \mathbf{h}_c^k)$

  //global context information
  $leaf\_nodes_{d,c} \leftarrow LEAF\_INDEX(CONCAT(x_d, x_c))$ ▷ the length of leaf nodes = n^{trees}
  //M^{embed} is a matrix of all leaf embeddings.
  $\mathbf{M}_{d,c}^{embed} \leftarrow \{\mathbf{M}_i^{embed}, \forall i \in leaf\_nodes_{d,c}\}$
  $\mathbf{h}_{d,c}^{global} \leftarrow AVERAGE(SELF\_ATTENTION(\mathbf{M}_{d,c}^{embed}, num\_heads))$

  //FFN is a 2-layer MLP.
  $P(e_{d,c} = 1|\mathbf{x}) \leftarrow FFN(CONCAT(\mathbf{h}_{d,c}^{local,k}, \mathbf{h}_{d,c}^{global}))$
end for

Table 2. Description of Fields

Column Name	Description
device_os	Device OS version number
device_platofrm	Device platform (e.g., iOS, Android)
device_ip	Device IP
device_brand	Brand of device (e.g., Xiaomi)
device_id	Device ID
device_language	For example, en, zh
device_network	Whether is wifi (e.g., True, False)
device_carrier	Mobile carrier (e.g., China Mobile)
device_name	For example, Samsung
install_country	For example, China
install_city	For example, Beijing
channel_id	Channel ID
campaign_id	Campaign ID
package_name	The unique package name of the app promoted by advertisers

Table 3. Statistics Data of Datasets

Dataset	#Dev	#Chan-Camp	#Normal Install (Train, Test)	#Bots Install (Train, Test)
dataset-1	1,676,101	1,347	1,245,650, 162,960	270,815, 20,560
dataset-2	1,313,073	1,190	1,049,610, 195,792	139,349, 9,596
dataset-3	1,299,895	1,139	1,153,705, 181,437	77,708, 12,016

#Dev denotes the number of device nodes, #Chan-Camp denotes the number of channel-campaign nodes, #Normal Install (Train, Test) denotes the number of normal installs in train data and test data, and #Bots Install (Train, Test) denotes the number of bots installs in train data and test data.

are detailed in Table 2. It is worth mentioning that the geographic information is actually obtained through IP information.

For a fair comparison, we follow the same rules to construct our datasets, which is extracting 8-consecutive-day install data, of which the first 7 days were used as the training data and the last day is used as test data. The statistics data of the three datasets are detailed in Table 3. It is worth explaining that the normal installs and bots installs correspond to the normal edges and bots edges of the graph.

4.2 Feature Engineering

Feature engineering can strongly affect the performance of the model. In the feature engineering phase, we perform data cleaning first to normalize the raw data, then we extract as many features as we can to capture the potential pattern of the bots installs. Finally, we perform feature selection to achieve better performance with our model.

4.2.1 Data Cleaning. Raw install data of mobile is messy due to the different standards of different downstream channels. Therefore, we perform different data cleaning methods for different fields of raw data. Now we will introduce some cleaning methods on several fields. There are three columns in Table 4, named device_id, device_brand, and device_os. Considering the users' personal privacy issues, the column device_id is encrypted by md5. For column device_brand, we will merge some sub-brands into one big brand. For example, "redmi note" and "redmi" actually belong to the Redmin series of Xiaomi Inc. Thus, we uniformly replace it with redmi. As for device_os, we truncated the OS version number of a mobile phone, and we only kept the first two digits of entire OS version number since some downstream channels only report the major OS version number.

Table 4. Data Cleaning Example

device_id	device_brand	device_os
d18c76ae66a45745c0531d168ae1ed8e	Redmi Note	5.4
9f89c84a559f573636a47ff8daed0d33	Redmi	8.1.5
b7f5911f43b7597b89b1ac22a882fa2a	Samsung	6
b05982545b8bca8632249fef063daa16	iPhone	7.1

In this way, the 8.1.5 will be transformed to 8.1. We normalized other fields similarly, so we will not detail them here.

4.2.2 Feature Extraction.

(1) *Statistical features:* The statistical features are mainly composed of repetition features, distribution features, and other statistical features. Repetition features calculate the repetition of the device or geographic information in different channels, such as IP repetition features, which are important features for bots install fraud detection. Distribution features contribute to mining bots installs since bots installs tend to be clustered. We used time distribution, geographic distribution, and device information related distribution in our experiments. As for other statistical features, they refer to some count, min, max features, and so on.
(2) *Categorical features:* The category features are mainly composed of device information, geographic information, campaign information, and channel information, which are detailed in Table 2. We deal with these categorical features differently for different models. We use a label encoder to encode categorical features in the LightGBM model because it supports the processing of categorical features directly instead of using one-hot encoder. As for the deep learning model, we put them into the embedding layer to get their dense vector in low-dimensional space.

4.2.3 Feature Selection. Feature selection is an important step toward building a robust prediction and classification model and can help to prevent overfitting of the data. In our experiments, we try different feature selection techniques including principal component analysis [50] and wrapper subset evaluation [23] with our model. We chose the wrapper method at last since it resulted in the highest performance compared to principal component analysis. After feature engineering, we get a total of 140 features, which will be used in the next experiments.

4.3 Baseline Models

To validate the effectiveness of our BotSpot++ model, we chose one traditional machine learning method and developed four deep learning methods as baselines. At the beginning, we considered R-GCN as one of the baselines, which is a great model. However, R-GCN aims to deal with multi-relation directed graphs, and there is only single relation undirected graph in our scenario. Even if we migrate it to our scenario, its essence is a GCN model. Due to the shortcomings of the GCN and the implementation of GraphSAGE, we do not intend to use R-GCN as one of the baselines.

The hyperparameters of all deep learning models are set as follows. The batch size of dataset is set to 256, the embedding dimension of category features is set to 16, the learning rate for Adam optimizer is set to 0.001. All deep learning models also end up with an FFN module, which is a 2-layer MLP model. We use the same bipartite graph for all GNN models with a 1-layer graph convolution layer. These baseline models are detailed as follows.

LightGBM. LightGBM, widely adopted by industry, is a well-known gradient boosting framework that uses a tree-based learning algorithm. We apply grid search to find optimal parameters. Finally, we set the hyperparameters of our baseline model to 500 decision trees and the maximum depth to 5.

MLP. We implemented a 2-layer MLP model with a dropout mechanism [41] to avoid overfitting. The number of neuron units of this 2-layer MLP are set to 64 and 32, respectively, with the probability of 0.2 for dropout mechanism.

GraphSAGE. GraphSAGE is a framework for inductive representation learning on large graphs and is especially useful for graphs that have rich node attribute information. Therefore, we developed it as one of our baseline models with a sampling size of 100.

GraphConsis. We also implemented a GraphConsis model that uses the GNN and and the attention mechanism to alleviate the fraud detection problems in recent work [28]. Since there is no relation inconsistency problem in our scenario, we only apply the mechanisms that tackle context inconsistency and feature inconsistency problems. To reduce time complexity, we performed stratified sampling for nodes with more than 11,000 neighbors when calculating the attention score—that is, keep normal neighbors within 9,000 and bots neighbors within 2,000. It is worth explaining that we use two different linear layers to transform the current channel-campaign node c and its neighbors (i.e., device nodes) separately before applying the attention mechanism to them, because c and its neighbors are heterogeneous nodes in our bipartite graph. Because c and its neighbors are heterogeneous nodes in our bipartite graph. The hyperparameters of our model are consistent with the parameters in the work of Liu et al. [28].

GAT. We implemented another inductive GNN model with the attention mechanism, which is the GAT model. We also simplified the GAT model to reduce the time complexity—that is, if the current node has more than 1,000 neighbors, we randomly sample 1,000 neighbors from them instead of all the neighbors to calculate the attention score.

Veličković et al. [46] found extending the self-attention mechanism to employ multi-head attention to be beneficial. Therefore, we also conducted GAT multi-head experiments on all datasets with one to four heads. The experimental results show that GAT with two heads achieves the best performance except for dataset-3, on which four-heads GAT is the best. This proves that increasing the number of heads could improve the performance of GAT. However, increasing the number heads of GAT also brings a lot of time overhead and memory overhead. To trade off the model performance and time overhead, we finally adopted the GAT model with two heads.

Therefore, the hyperparameters of GAT are set as follows. The number of heads is set to 2, the probability of dropout is set to 0.1, and the alpha of LeakyReLU is set to 0.2.

BotSpot. BotSpot is an anti-fraud method based on a heterogeneous graph that incorporates both local context and global context via GNNs and the gradient boosting classifier to detect bots installs at Mobvista. Considering that our work is based on this work, we also take it as our baseline model.

4.4 Experimental Settings

For performance evaluation, we chose to adopt *Recall@N%Precision* as our basic criterion, which is detailed in Equation (10). Before we introduce *Recall@N%Precision*, let us understand how to calculate the *Precision* and *Recall* as shown in Equations (8) and (9). Among them, TP denotes the number of samples whose ground truth is positive and predicted to be positive. FP denotes the number of samples whose ground truth is negative but predicted to be positive. FN denotes the number of samples whose ground truth is positive and predicted to be negative. Based on the definitions of *Precision* and *Recall*, we defined *Recall@N%Precision* as the value of *Recall* when its *Precision* is equal to $N\%$.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{8}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9}$$

$$\text{Recall@N\%Precision} := \textit{Recall Where Precision} = N\% \tag{10}$$

In the scenario of mobile bots install fraud detection, we must ensure high model precision to reduce the false-positive rate, which avoids damaging the advertiser's business and the reputation of the advertising platform. This is because false positives indicate cases where installs were falsely identified as fraudulent and may potentially penalize legitimate channels. This could harm the relationship with its quality channel partners rather than protect it from malicious ones. As a result, *Recall@N%Precision* is an essential condition for a model to be deployed. Therefore, we set the N to 90, 85, and 80 to compare the performance of different models. We abbreviate Recall@N%Precision as Recall@N%P in the following.

We implement LightGBM using the Microsoft LightGBM library and other deep learning models using PyTorch [32], which are publicly accessible.[7] We conducted our experiments on the Amazon p3.8xlarge instance with a 32-core CPU, four GPUs (NVIDIA Tesla V100), and 244 GB of memory.

4.5 Results and Analysis

In this section, we will compare the experimental results using the baseline models mentioned in the last section with our BotSpot++ model. We address RQ1 by evaluating our BotSpot++ model and baseline models on multiple different time period offline datasets introduced in Section 4.1. We also deploy multiple sub-models of BotSpot++ by removing some parts from its entire architecture to address RQ2. Furthermore, we address RQ3 and RQ5 by conducting a series of parametric search experiments and time complexity experiments, respectively. Then, to address RQ4, we deploy LigthGBM, MLP, GraphSAGE, BotSpot, and our proposed BotSpot++ on the real online production environment to compare their performance. In addition, we address RQ6 by conducting a case study and data visualization. Finally, to address RQ7, we also conduct down-sampling experiments.

4.5.1 Offline Experiments. First, we answer RQ1 by evaluating Recall@90%P, Recall@85%P, and Recall@80%P with our proposed model and baseline models on three datasets for different time periods collected from the advertising platform of Mobvista. It should be noted that we only show the best one among the multiple super nodes construction methods in BotSpot++.

For RQ1, as shown in the tables for various durations, our proposed BotSpot++ is able to outperform all the other competitive baseline methods on all datasets given a relatively high fixed precision. Therefore, BotSpot++ is able to recall more bots installs given the same precision (e.g., 80%, 85%, and 90%) compared with other methods, which proves the effectiveness of incorporating domain knowledge and cluster algorithms into the graph-structured data.

It is obvious to observe that BotSpot++ outperforms BotSpot on all offline datasets at all metrics, which means the performance gain is attributed to the introduction of super device nodes and the self-attention mechanism on leaf embedding. Especially, the BotSpot++ model has a relatively significant improvement on dataset-1 and dataset-3 compared to BotSpot. It is worth mentioning that Recall@90%P is improved about 4.1%, Recall@85%P is improved about 5.1%, and Recall@80%P is improved about 7.86% on dataset-1. This proves that grouping of super device nodes and the self-attention mechanism can help BotSpot achieve better performance.

However, we should also note that the GraphSAGE model outperforms the GraphConsis model in most cases. The possible reasons for the performance margin are as follows. First, the channel-campaign node c and its neighbors (i.e., device nodes) are heterogeneous nodes. This will cause an inability to calculate the attention (similarity) score directly between them. As a result, we

[7] https://github.com/mobvistaresearch/BotSpot-Plus.

Table 5. Evaluation Results for Dataset-1

	Recall@90%P	Recall@85%P	Recall@80%P
LightGBM	0.1952	0.2234	0.2341
MLP	0.1892 (−3.07%)	0.2106 (−5.73%)	0.2423 (+−3.50%)
GraphSAGE	0.2684 (+37.50%)	0.3030 (+35.63%)	0.3455 (+47.59%)
GAT (two heads)	0.2770 (+41.91%)	0.2917 (+30.57%)	0.3212 (+37.21%)
GraphConsis	0.2388 (+22.34%)	0.3053 (+36.66%)	0.3424 (+46.26%)
BotSpot	0.2732 (+39.96%)	0.3033 (+35.77%)	0.3632 (+55.15%)
BotSpot++ (gmm)	**0.2812 (+44.06%)**	**0.3147 (+40.87%)**	**0.3816 (+63.01%)**

LightGBM is the base model, and +x% represents the percentage of current model performance improvement compared with LightGBM. The bold numbers denote the best performance of all models. BotSpot++ (gmm) denotes that BotSpot++ uses gmm to construct super device nodes.

transform the current node c and its neighbors into new features with the same dimensions by using two different linear layers and calculate the attention (similarity) score on them. But the attention score calculated in this way may be inaccurate, which will cause the performance of a model with attention score based sampling to be inferior to the random sampling in convolution operation. Second, the GraphConsis model introduces a trainable context embedding not only for every channel-campaign node but also every device node, which is different from GraphSAGE. But the introduction of embeddings for device nodes may cause overfitting easily in our scenario.

Another observation can be made that the GAT model outperforms the GraphConsis model in most cases. This is because GAT uses the additive attention mechanism, whereas GraphConsis uses a mechanism similar to the multiplicative attention mechanism when calculating the attention/similarity score, which may hurt the performance of GraphConsis more than GAT when c and its neighbors are heterogeneous nodes.

4.5.2 Ablation Experiments. To answer RQ2, we conduct a series of ablation experiments on dataset-1 and dataset-3 to investigate the impact of different modules of BotSpot++'s architecture on its overall performance. Therefore, we build the following variants of the BotSpot++ model: (1) − $self\_attention$: the self_attention module is removed from BotSpot++'s architecture; (2) − $leaf\_embedding$: all leaf_embedding features are removed from BotSpot++'s architecture. In other words, the right part is removed from BotSpot++'s architecture as shown in Figure 2; (3) − $label\_smoothing$: the label_smoothing mechanism is removed from the loss function of BotSpot++; (4) − gnn: the gnn part is removed from BotSpot++'s architecture. In other words, the left part is removed from BotSpot++'s architecture as shown in Figure 2. It should be noted that we use the GMM clustering algorithm for constructing super device nodes in BotSpot++.

Tables 8 and 9 compare the predictive performance of our different ablations on dataset-1 and dataset-3. The results show that the self_attention, leaf_embedding, label_smoothing, and gnn modules all contribute to the BotSpot++ model, of which the gnn module or the leaf_embedding module contributes the most at Recall@90%P. It should be noted that the leaf_embedding module includes a self_attention module. Another observation is made that label_smoothing can recall more bots installs at Recall@85%P and Recall@80%P. Furthermore, the leaf_embedding module results in a 0.01 to 0.02 increase in all three metrics. The self_attention module also leads to a 0.01 increase in all three metrics.

4.5.3 Parametric Search Experiments. For RQ3, we conduct a series of parametric search experiments on dataset-1 and dataset-3 to investigate the impact of different hyperparameters of the BotSpot++ model. As Figure 7(a) and Figure 8(a) show, the performance of BotSpot++ is very close

Table 6. Evaluation Results for Dataset-2 of Offline Datasets

	Recall@90%P	Recall@85%P	Recall@80%P
LightGBM	0.4428	0.4520	0.4572
MLP	0.4540 (+2.53%)	0.4638 (+2.61%)	0.4729 (+3.43%)
GraphSAGE	0.4654 (+5.10%)	0.4733 (+4.71%)	0.4820 (+5.42%)
GAT (two heads)	0.4619 (+4.31%)	0.4710 (+4.20%)	0.4807 (+5.14%)
GraphConsis	0.4560 (+2.98%)	0.4650 (+2.88%)	0.4754 (+3.98%)
BotSpot	0.4682 (+5.74%)	0.4781 (+5.77%)	0.4857 (+6.23%)
BotSpot++ (package_name)	**0.4685 (+5.80%)**	**0.4791 (+6.00%)**	**0.4877 (+6.67%)**

LightGBM is the base model, and +x% represents the percentage of current model performance improvement compared with LightGBM. The bold number denotes the best performance of all models. BotSpot++ (package_name) denotes that BotSpot++ uses package_name to construct super device nodes.

Table 7. Evaluation Results for Dataset-3 of Offline Datasets

	Recall@90%P	Recall@85%P	Recall@80%P
LightGBM	0.4747	0.5816	0.6424
MLP	0.4967 (+4.63%)	0.5400 (−7.15%)	0.5753 (−10.45%)
GraphSAGE	0.5897 (+24.23%)	0.6347 (+9.13%)	0.6749 (+5.06%)
GAT (2 heads)	0.5868 (+23.61%)	0.6348 (+9.15%)	0.6606 (+2.83%)
GraphConsis	0.4983 (+4.97%)	0.5615 (−3.46%)	0.6031 (−6.12%)
BotSpot	0.5903 (+24.35%)	0.6500 (+11.76%)	0.6908 (+7.53%)
BotSpot++ (package_name)	**0.6137 (+29.28%)**	**0.6622 (+13.86%)**	**0.7033 (+9.48%)**

LightGBM is the base model, and +x% represents the percentage of current model performance improvement compared with LightGBM. The bold number denotes the best performance of all models. BotSpot++ (package_name) denotes that BotSpot++ uses package_name to construct super device nodes.

Table 8. Ablation Results for Dataset-1 of Offline Datasets

	Recall@90%P	Recall@85%P	Recall@80%P
BotSpot++ (gmm)	**0.2812**	**0.3147**	**0.3816**
− ***self_attention***	0.2757 (−1.96%)	0.3052 (−3.02%)	0.3500 (−8.28%)
− ***leaf_embedding***	0.2788 (−0.85%)	0.3056 (−2.89%)	0.3694 (−3.20%)
− ***label_smoothing***	0.2740 (−2.56%)	0.3095 (−1.65%)	0.3528 (−7.55%)
− ***gnn***	0.2728 (−2.99%)	0.3134 (−0.41%)	0.3705 (−2.91%)

Recall@N%Precision is shortened to Recall@N%P. The bold number denotes the best performance of all models. BotSpot++ (gmm) denotes that BotSpot++ uses gmm to construct super device nodes.

when num_trees is set to 200 and 300. But it is better to set *num_trees* to 200 due to the low calculation time. We can find that BotSpot++ achieves the best performance when *num_heads* is set to 2 from Figure 7(b) and Figure 8(b). We can see that BotSpot++ achieves the best performance when the alpha is set to 0.95 based on Figure 7(c) and Figure 8(c). As depicted in Figure 7(d) and Figure 8(d), an observation is made that the construction methods for super device nodes of *gmm* and *package_name* can achieve the best performance in dataset-1 and dataset-3, respectively.

4.5.4 Time Complexity Experiments. For RQ5, Table 10 compares the elapsed time of the offline training phase and the online inference phase. The result shows that GAT is the slowest model in the training phase, which is because GAT calculates the attention score of each nodes' neighbors and uses all of its neighbors to get the new representation of the current node. We can also find

Table 9. Ablation Results for Dataset-3 of Offline Datasets

	Recall@90%P	Recall@85%P	Recall@80%P
BotSpot++ (gmm)	**0.6087**	**0.6593**	**0.7021**
– ***self_attention***	0.6038 (−0.81%)	0.6456 (−2.08%)	0.6907 (−1.62%)
– ***leaf_embedding***	0.5888 (−3.27%)	0.6464 (−1.96%)	0.6967 (−0.77%)
– ***label_smoothing***	0.6018 (−1.13%)	0.6327 (−4.03%)	0.6934 (−1.24%)
– ***gnn***	0.5980 (−1.76%)	0.6468 (−1.90%)	0.6945 (−1.08%)

Recall@N%Precision is shortened to Recall@N%P. The bold number denotes the best performance of all models. BotSpot++ (gmm) denotes that BotSpot++ uses gmm to construct super device nodes.

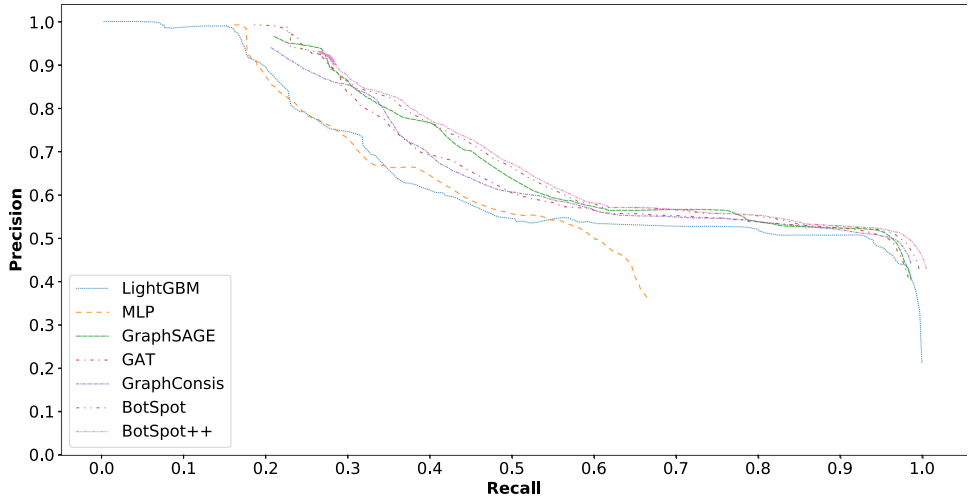

Fig. 4. Precision-Recall curve for dataset-1.

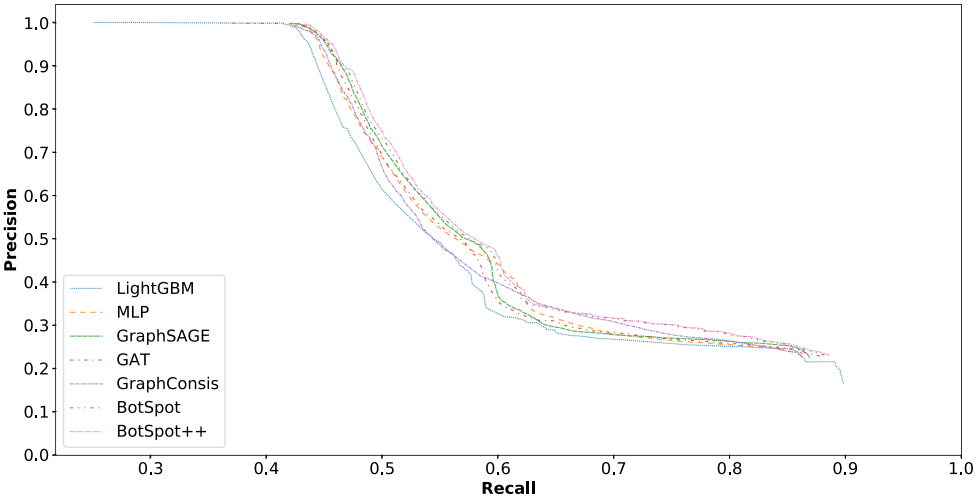

Fig. 5. Precision-Recall curve for dataset-2.

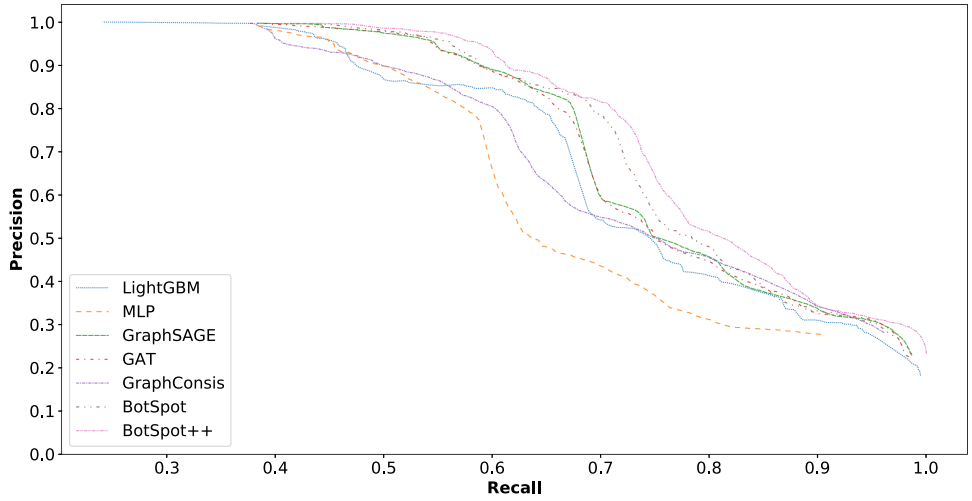

Fig. 6. Precision-Recall curve for dataset-3.

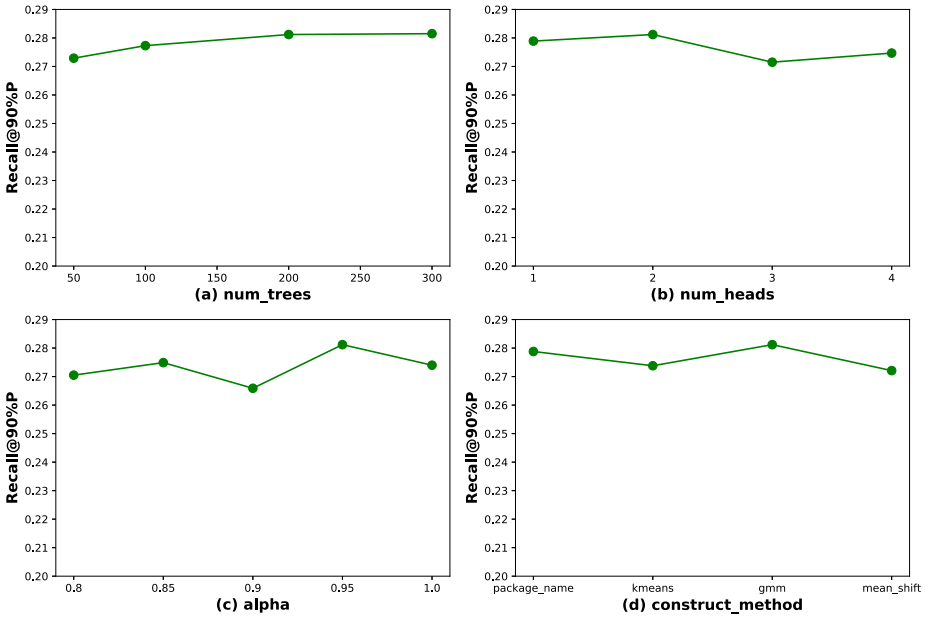

Fig. 7. The impact on different parameters of BotSpot++ on dataset-1.

that GraphConsis is the second slowest model in the training phase, which is because GraphConsis calculates the similarity score more efficiently than GAT and samples a fixed number of neighbors of the current node. As for LightGBM, it takes the shortest time in the training phase, which is obvious. It should be noted that GraphSAGE, BotSpot, and BotSpot++ take much less training time than other graph models, which is because we apply the pre-sampling strategy to these models to reduce the elapsed time in the training phase. We use multi-process technology for pre-sampling to further speed up the process.

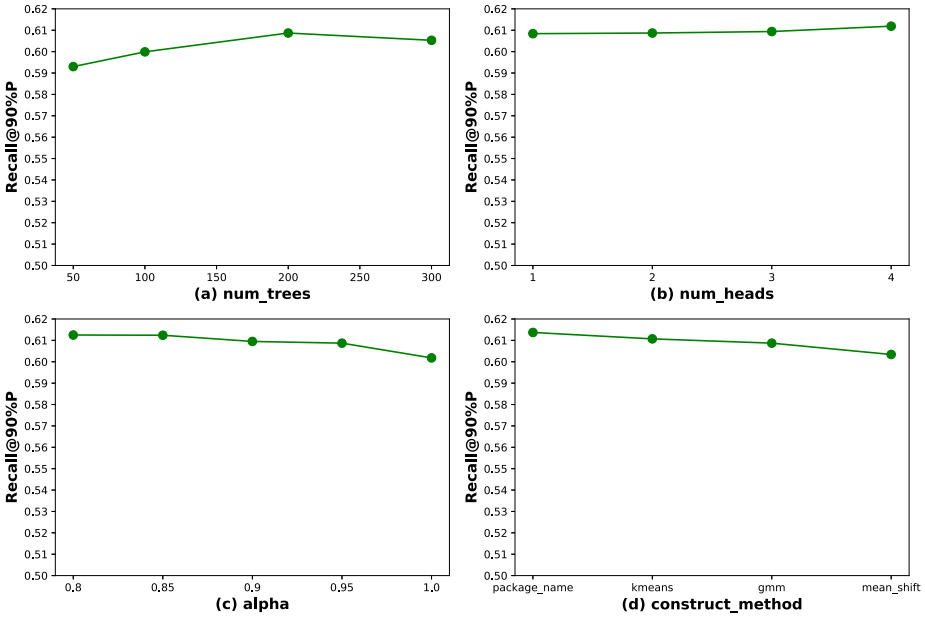

Fig. 8. The impact on different parameters of BotSpot++ on dataset-3.

Table 10. Elapsed Time Results of Different Models for Offline Training (Per Epoch/Tree) and Online Inference (Per Sample)

	Offline Training Time	Online Inference Time
LightGBM	2 s	1 ms
MLP	1 min	1 ms
GraphSAGE	3 min	5 ms
GAT (2 heads)	**7 h, 10 min**	**18** ms
GraphConsis	3 h, 18 min	8 ms
BotSpot	51 min	8 ms
BotSpot++	1 h, 3 min	10 ms

The elapsed time is the average time on the three datasets.

As for online inference time, we can find that LightGBM and MLP are the fastest models, and GAT is the slowest. Considering that we cannot apply a multi-process technology to pre-sampling neighbors for GraphSAGE, BotSpot, and BotSpot++ during online inference, the gap between these models and the remaining models in the inference time is not as large as the training time.

It is worth mentioning that we only calculate the attention score for part of the neighbors in attention module of GAT and GraphConsis, which can speed up the calculation.

4.5.5 Online Experiments. Then we answer RQ4 by conducting a series of experiments for 7 consecutive days on our online production environment, in which each day's install data is used as the test set, the day before each day is used as the validation data, and the previous 6 days of the validation data is used as the training data.

Since the high memory overhead of GAT and GraphConsis and their performance is not as good as GraphSAGE, we only deploy LightGBM, MLP, GraphSAGE, BotSpot, and our proposed BotSpot++ on our advertisement platform for online experiments. Then we evaluate the model

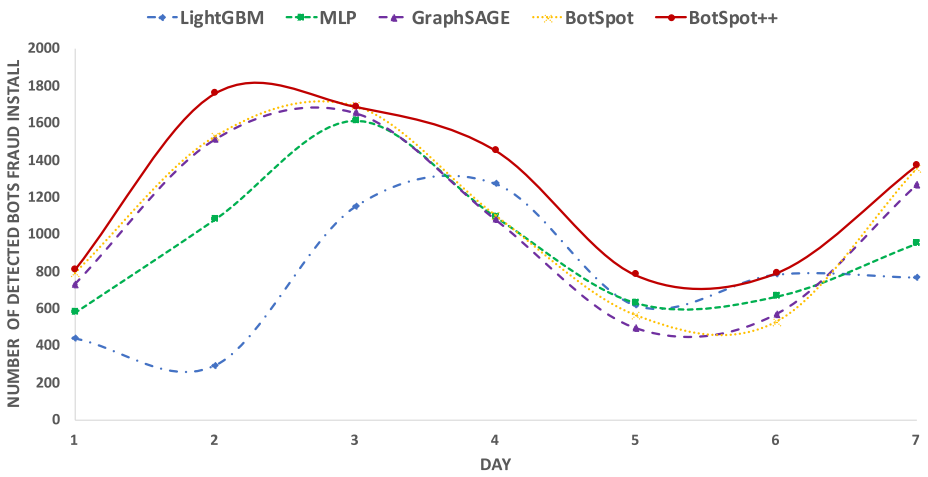

Fig. 9. Online evaluation result. The horizontal axis represents 7 consecutive days, and the vertical axis represents the number of bots installs detected by different models when their precision is fixed to 90%.

Table 11. Total Number of Bots Installs Detected During Online Evaluation

	LightGBM	MLP	GraphSAGE	BotSpot	BotSpot++
Total	5,321	6,612	7,317	7,554	8,655

The proposed method outperforms the baseline models at Mobvista.

performance by comparing the number of bots installs detected by different models. In detail, we determine the split threshold by searching it to make the models reach 90% precision on validation data, then compare the number of bots installs detected by this threshold on test data, which are checked by human experts.

As shown in Figure 9, we can see that our proposed BotSpot++ consistently outperforms BotSpot and other baseline models. We also note that GraphSAGE does not always outperform MLP, and MLP does not always outperform LightGBM. This is because downstream traffic and bots install fraud patterns that are constantly changing, but generally speaking, as shown in Table 11, the performance of different models is regular, and BotSpot++ has the best performance, BotSpot is second, and LightGBM has the worst performance.

4.5.6 Case Study. We make a case study to quantitatively evaluate the effect of BotSpot++ by examining the recalled ground-truth bots installs from them and analyze the portion of positive samples recalled by BotSpot++ yet ignored by BotSpot and GraphSage. Yao et al. [54] examined the model capacity of BotSpot and LightGBM with a similar approach. It has been found that by incorporating local context, BotSpot is able to recall more bots installs compared with LightGBM, where the bots installs recalled by BotSpot but not by the other methods is situated in an ad channel where the device brand distribution is a strong indicator to identify bots and normal installs. In this work, we focus on those positive samples that can only be recognized by BotSpot++ yet not by GraphSAGE and BotSpot in a given time duration. For the chosen dataset, we first set the threshold of the three methods to achieve 85% precision and label the correctly identified bots installs by the three methods. Our analysis shows that there are a total of 853 bots installs recalled, among which 107 bots installs are only recalled by BotSpot++, and 13 bots installs recalled by the other two methods but not by BotSpot++. We then compared the 107 bots installs with other bots

Table 12. Illustration of Several Bots Installs Recognized by BotSpot++ but Not by BotSpot and GraphSAGE

device_id	package_name	chan_bots_ratio	sup_dev_chan_num	sup_dev_chan_bots_ratio_gt_50%_num	sup_dev_chan_bots_ratio_max
**23fad	pack_**3f1	0.12	12	4	54.12%
**021c5	pack_**63a	0.18	18	7	78.53%
**a432b	pack_**3c2	0.15	6	2	92.06%
**ea152	pack_**3c2	0.09	3	1	91.37%
**8df16	pack_**ad3	0.19	14	3	97.29%

Let **chan_bots_ratio** denote the bots ratio of the corresponding ad channel, **sup_dev_chan_num** denote the ad channel numbers connected to the corresponding super device nodes, **sup_dev_chan_bots_ratio_gt_50%_num** be the number of ad channels connecting to the super device node with a bots ratio greater than 50%, and **sup_dev_chan_bots_ratio_max** be the maximum bots ratio of the ad channels connecting to the super device node.

installs recalled by all three methods and found that there are 44 bots installs situated in five ad channels that are not presented in the other positive samples. Furthermore, the *package_name* of these installs is rarely seen in the entire population of bots installs.

We also observe that the bots install ratio of these five ad channels is between 0.09 and 0.19, meaning that most installs are normal installs and these 44 bots installs are actually hard examples for the model to identify. Leveraging the hierarchical graph structure with super device nodes, we observe that the associated super device nodes of the bots devices connect to some high-risk ad channels with a bots install ratio ranging between 0.5412 and 0.9729, and we argue that BotSpot++ is able to detect these 44 bots installs utilizing the propagated information from the multiple high-risk ad channels. In Table 12, we illustrate several bots installs recalled only by BotSpot++ and statistics of their associated ad channels connected to both the device nodes and super device nodes.

4.5.7 Data Visualization. To qualitatively evaluate the effect of BotSpot++, we visualized the dense vector of the last fully connected layer of our BotSpot++. We adopted the *t*-distributed stochastic neighbor embedding (t-SNE) technique [45] to visualize the dense vector. We randomly selected the installs of several channel-campaign nodes in the test set of dataset-1 and dataset-3 for visualization. As Figure 10 shows, the red dots in the figures indicate the bots installs, and the green dots indicate the normal installs. We can see that normal installs and bots installs can be well separated, which further proves the superiority of our BotSpot++ model.

4.5.8 Down-Sampling Experiments. The class distribution is unbalanced for the training data, as shown in Table 3. Therefore, we conduct down-sampling experiments of BotSpot++ on dataset-3 and provide the experimental results. We construct the following three down-sampling datasets: (1) dataset-3 (1:1): down-sampling normal installs on dataset-3 with a size of the bots installs' number; (2) dataset-3 (1:2): down-sampling normal installs on dataset-3 with a size of two times the bots installs' number; and (3) dataset-3 (1:4): down-sampling normal installs on dataset-3 with a size of four times the bots installs' number.

As shown in Table 13, the results show that down-sampling cannot improve the performance of BotSpot++. This is because each node is missing a large number of neighbors when down-sampling is applied, yet the neighbor information is very important for BotSpot++, which will damage the performance of BotSpot++.

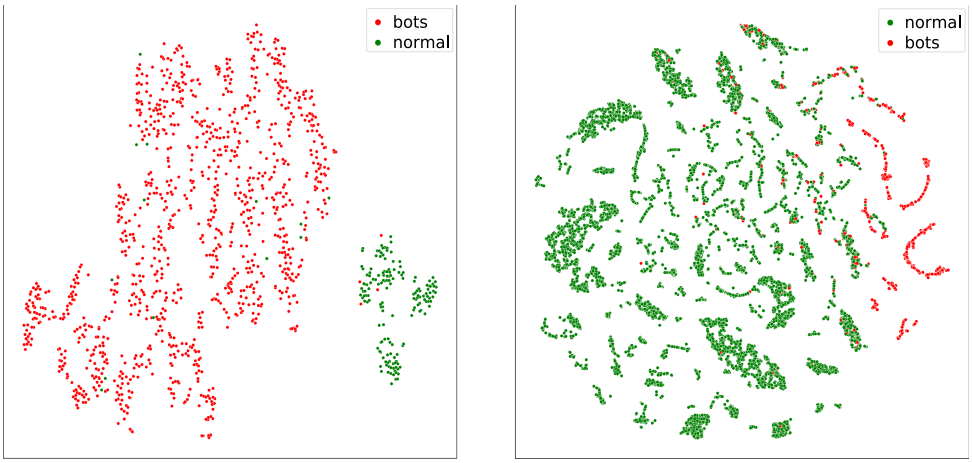

Fig. 10. Visualization of the last fully connected layer data of BotSpot++ with dataset-1 (left) and dataset-3 (right). The red dots indicate the bots fraud installs, and the green dots indicate the normal installs.

Table 13. Down-Sampling Results of BotSpot++ Based on Dataset-3

	Recall@90%-P	Recall@85%-P	Recall@80%-P
Dataset-3	**0.6137**	**0.6622**	**0.7033**
dataset-3 (1:1)	0.5879 (−4.20%)	0.6335 (−4.33%)	0.6771 (−3.73%)
dataset-3 (1:2)	0.5905 (−3.78%)	0.6431 (−2.88%)	0.6820 (−3.03%)
dataset-3 (1:4)	0.5944 (−3.14%)	0.6456 (−2.51%)	0.6918 (−1.64%)

Recall@N%Precision is shortened to Recall@N%P. The bold number denotes the best performance of all models.

5 DISCUSSION

In this section, we will discuss the model convergence problem, the limitations of our proposed model, and further outline how these might be overcome in future work.

Model convergence. We will discuss the model convergence problem from the following two aspects. First, whether the model converges. Generally, the loss of our model has stabilized to about 0.28 after about six epochs. Thus, our model is convergent. Second, how to make the model converge to a better local optimal point. As we all know, neural network models are easy to overfit. To overcome this problem, we mainly apply two mechanisms to prevent the model from overfitting. First, we apply the dropout mechanism and hard negative mining technique. Second, we apply a stratified sampling strategy when sampling the neighbors of channel-campaign nodes. In this way, our model could achieve a better model convergence.

Memory requirement. In the current implementation of BotSpot++, we try to cache the data used frequently, such as the neighbors of every node, as much as possible to speed up the process, which may result in large memory usage. It is a solution to delete useless data immediately after a sample is fed into the model or being predicted.

Supervised model. BotSpot++ is an ensemble model under supervised learning settings. However, we have a lot of unlabeled samples, and the use of these samples can further benefit our model. As we introduced earlier, BotSpot++ is composed of a GBM part and a graph part, where the output vectors of the GBM part are called *global context* and the output vectors of the graph part are called *local context*. To make BotSpot++ be a semi-supervised model, we need to modify the two parts separately. Let us discuss the modification of the graph part first. There are two message

passing mechanisms in the graph part, namely message passing for channel- campaign nodes and message passing for device nodes. The message passing mechanism for device nodes is the same with GraphSAGE, which can be applied to semi-supervised learning settings directly, whereas the message passing mechanism for channel-campaign nodes is proposed with learning biased aggregators, where the neighborhood labels need to be known. In other words, we only need to extend the message passing mechanism for channel-campaign nodes in the graph part. The solution is that we assign new labels to the unlabeled/unknown neighbors of channel-campaign nodes, such as −1. Then the biased representation of the channel-campaign node is calculated by leveraging normal installs, bots installs, and unknown installs (i.e., unlabeled neighbors) with the attention mechanism. As for the GBM part, we train a GBM model leveraging the labeled samples first. Then we predict the unlabeled data using the pre-trained model to get their leaf nodes. Finally, these leaf nodes are fed into the BotSpot++ model. It is worth mentioning that only labeled samples participate in the calculation of loss and back-propagation. Due to limited space, we will implement this part in future work.

6 CONCLUSION

Bots install fraud is a notoriously hard problem in the mobile adverting industry for its dynamically changing behavioral pattern. In this work, we proposed a novel model named *BotSpot++* for bots install fraud detection. We solved the sparsity of the device's neighbor utilizing domain knowledge and clustering algorithms to enrich the graph structure. A self-attention module is also incorporated in BotSpot++ to enhance the interaction among leaf nodes, which is inherently an issue in GBM. Furthermore, we applied the label smoothing module to alleviate the problem of noisy labels. We conducted a series of comprehensive experiments to evaluate the effectiveness and efficiency of BotSpot++. The experimental results demonstrate that BotSpot++ yields the best performance and good efficiency. We also discussed several limitations of our proposed model, including the high memory requirement and that it is suitable only for supervised learning settings. Finally, we outlined how to overcome these issues in future work.

ACKNOWLEDGMENTS

We would like to thank reviewers for their helpful suggestions and thank our colleagues for reviews.

REFERENCES

[1] AppsFlyer. 2019. 2019 Fraud Trends Uncover Fascinating Results. Retrieved September 15, 2020 from https://www.appsflyer.com/blog/mobile-ad-fraud-trends/.

[2] Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (2001), 5–32.

[3] Adam Breuer, Roee Eilat, and Udi Weinsberg. 2020. Friend or faux: Graph-based early detection of fake accounts on social networks. In *Proceedings of the 2020 Web Conference*. 1287–1297.

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357.

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems* 29 (2016), 3844–3852.

[7] Yizong Cheng. 1995. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17, 8 (1995), 790–799.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[9] Yingtong Dou, Weijian Li, Zhirong Liu, Zhenhua Dong, Jiebo Luo, and S. Yu Philip. 2019. Uncovering download fraud activities in mobile app markets. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'19)*. IEEE, Los Alamitos, CA, 671–678.

[10] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. 2020. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. 315–324.

[11] Joan Bruna Estrach, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR'14)*, Vol. 2014.

[12] Matthias Fey. 2019. Just jump: Dynamic neighborhood aggregation in graph neural networks. *arXiv preprint arXiv:1904.04849* (2019).

[13] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks*, Vol. 2. IEEE, Los Alamitos, CA, 729–734.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 855–864.

[15] Ch. Md. Rakin Haider, Anindya Iqbal, Atif Hasan Rahman, and M. Sohel Rahman. 2018. An ensemble learning based approach for impression fraud detection in mobile advertising. *Journal of Network and Computer Applications* 112 (2018), 126–141.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1024–1034.

[17] John A. Hartigan and Manchek A. Wong. 1979. Algorithm AS 136: A *k*-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 28, 1 (1979), 100–108.

[18] Jinlong Hu, Tenghui Li, Yi Zhuang, Song Huang, and Shoubin Dong. 2020. GFD: A weighted heterogeneous graph embedding based approach for fraud detection in mobile advertising. *Security and Communication Networks* 2020, 1 (2020), 1–12.

[19] Jinlong Hu, Junjie Liang, and Shoubin Dong. 2017. IBGP: A bipartite graph propagation approach for mobile advertising fraud detection. *Mobile Information Systems* 2017 (2017), 6412521.

[20] Dervis Karaboga and Bahriye Basturk. 2007. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization* 39, 3 (2007), 459–471.

[21] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[22] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[23] Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97, 1–2 (1997), 273–324.

[24] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. 2019. Spam review detection with graph convolutional networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2703–2711.

[25] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018).

[26] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, Raquel Urtasun, and Richard Zemel. 2018. Graph partition neural networks for semi-supervised classification. *arXiv preprint arXiv:1803.06272* (2018).

[27] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2077–2085.

[28] Zhiwei Liu, Yingtong Dou, Philip S. Yu, Yutong Deng, and Hao Peng. 2020. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1569–1572.

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546* (2013).

[30] Van-Hoang Nguyen, Kazunari Sugiyama, Preslav Nakov, and Min-Yen Kan. 2020. FANG: Leveraging social context for fake news detection using graph representation. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. 1165–1174.

[31] Richard Oentaryo, Ee-Peng Lim, Michael Finegold, David Lo, Feida Zhu, Clifton Phua, Eng-Yeow Cheu, et al. 2014. Detecting click fraud in online advertising: A data mining approach. *Journal of Machine Learning Research* 15, 1 (2014), 99–140.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

[33] Kasun S. Perera, Bijay Neupane, Mustafa Amir Faisal, Zeyar Aung, and Wei Lee Woon. 2013. A novel ensemble learning-based approach for click fraud detection in mobile advertising. In *Mining Intelligence and Knowledge Exploration*. Springer, 370–382.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 701–710.
[35] Siyuan Qi, Wenguan Wang, Baoxiong Jia, Jianbing Shen, and Song-Chun Zhu. 2018. Learning human-object interactions by graph parsing neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 401–417.
[36] J. Ross Quinlan. 2014. *C4.5: Programs for Machine Learning*. Elsevier.
[37] Douglas A. Reynolds. 2009. Gaussian mixture models. *Encyclopedia of Biometrics* 741 (2009), 659–663.
[38] Lior Rokach. 2010. Ensemble-based classifiers. *Artificial Intelligence Review* 33, 1 (2010), 1–39.
[39] Scalarr.io. 2019. What to Expect from Mobile Ad Fraud in 2020. Retrieved September 15, 2020 from https://scalarr.io/blog/articles/mobile-ad-fraud-trends-2020/.
[40] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *Proceedings of the European Semantic Web Conference*. 593–607.
[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[42] Christian Szegedy, V. Vanhoucke, S. Ioffe, Jon Shlens, and Z. Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 2818–2826.
[43] Mayank Taneja, Kavyanshi Garg, Archana Purwar, and Samarth Sharma. 2015. Prediction of click frauds in mobile advertising. In *Proceedings of the 2015 8th International Conference on Contemporary Computing (IC3'15)*. IEEE, Los Alamitos, CA, 162–166.
[44] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.
[45] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008), 2579–2605.
[46] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[47] Guan Wang, Sihong Xie, Bing Liu, and Philip S. Yu. 2012. Identify online store review spammers via social review graph. *ACM Transactions on Intelligent Systems and Technology* 3, 4 (2012), 1–21.
[48] Haobo Wang, Zhao Li, Jiaming Huang, Pengrui Hui, Weiwei Liu, Tianlei Hu, and Gang Chen. 2020. Collaboration based multi-label propagation for fraud detection. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI'20)*.
[49] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous graph attention network. In *Proceedings of the 2019 World Wide Web Conference*. 2022–2032.
[50] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1–3 (1987), 37–52.
[51] Zhang Xinyi and Lihui Chen. 2018. Capsule graph neural network. In *Proceedings of the International Conference on Learning Representations*.
[52] Chao Xu, Zhentan Feng, Yizheng Chen, Minghua Wang, and Tao Wei. 2018. FeatNet: Large-scale fraud device detection by network representation learning with rich features. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*. 57–63.
[53] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7370–7377.
[54] Tianjun Yao, Qing Li, Shangsong Liang, and Yadong Zhu. 2020. BotSpot: A hybrid learning framework to uncover bot install fraud in mobile advertising. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. 2901–2908.
[55] Ting Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*.
[56] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. 2019. Graph transformer networks. *arXiv preprint arXiv:1911.06455* (2019).
[57] Xin Zhang, Xuejun Liu, and Han Guo. 2018. A click fraud detection scheme based on cost sensitive BPNN and ABC in mobile advertising. In *Proceedings of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC'18)*. IEEE, Los Alamitos, CA, 1360–1365.

Received November 2020; revised July 2021; accepted July 2021