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

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

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, of which bots install fraud is undoubtedly 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. In this work, we propose BotSpot++, an improved version of BotSpot [1] for bots install fraud detection. BotSpot++ not only takes into account of the hierarchical structure of device nodes which enriches the graph structure and information flow utilizing domain knowledge, but also incorporate self attention mechanism to enhance the interaction of various leaf nodes. The offline experimental results show that our proposed method has a relative improvement of at least 9% compared with baseline model on the four real-world datasets covers different time periods, our proposed method also outperforms BotSpot over all the datasets, demonstrating the effectiveness of the mechanism design of the proposed method. We also deploy our proposed method and baseline models to our production environments, and the online performance demonstrates the effectiveness of our proposed method. Furthermore, we conducted data visualization and case study with our proposed BotSpot++.

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

Additional Key Words and Phrases: Mobile Advertising, Ad Fraud, Bots Install Fraud, Fraud Detection, Graph Neural Network

ACM Reference Format:
Tianjun Yao, Xiliang Wang, Qing Li, Shangsong Liang, and Yadong Zhu. 2020. Botspot++: A Hierarchical Deep Ensemble Model for Bots Install Fraud Detection in Mobile Advertising. *ACM Transactions on Information Systems* 1, 1 (November 2020), 24 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Mobile advertisement marketing is being expanded in higher velocity due to the rapidly increasing of smartphones and tablets usage in recent years. According to eMarketer [2] latest forecast [12],

[1]This work is based on our previous work [58] published in CIKM 2020, which serves the same purpose of detecting bots installs in mobile advertising, and has proposed several novel algorithmic designs to remedy the drawback in the previous work of BotSpot.
[2]https://pro-na1.emarketer.com/

Authors' addresses: Tianjun Yao, bertrand.yao@mobvista.com; Xiliang Wang, xiliang.wang@mobvista.com; Qing Li, qing.li@mobvista.com, Mobvista Inc., Guangzhou, China; Shangsong Liang, Sun Yat-sen University, Guangzhou, China, liangshangsong@gmail.com; Yadong Zhu, Mobvista Inc., Beijing, China, yadong.zhu@mobvista.com.

1046-8188/2020/11-ART $15.00
https://doi.org/10.1145/1122445.1122456

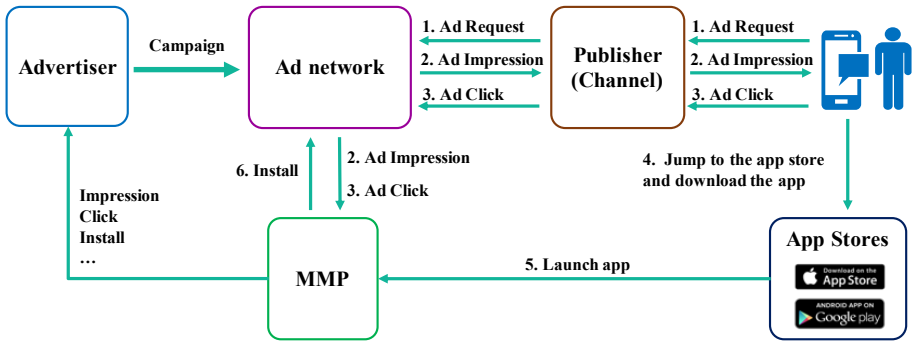

Fig. 1. **Mobile ad workflow**

the total spending on mobile advertising worldwide has reached to $246.05 billion which increases by $11.53 billion from 2019, and will continue to rise to $340.42 billion in 2022. With ad spending continuously growing for mobile marketing budgets, ad fraud attempts continue to rise as well, following the influx of capital. One of the popular pricing models in mobile app advertising is CPI (cost per install), where payment is based on app installs on their mobile device. Therefore, Mobile app install fraud became more popular over time.

The main contributors of mobile advertising market are *advertisers*, *publishers*, and *advertising networks (ad networks)*. Advertiser is the one who makes a contract with ad network to publish advertisements on behalf of himself or a company. Publisher is a traffic channel which could be an app that displays the advertisements (ads) to their users. Ad network plays a broker role between advertiser and publisher, such as Mobvista [3], a leading global mobile advertising company. Ad networks find the suitable publishers to display the ads. When a mobile user is using the app, ads are fetched from the ad networks and displayed to the user. Proper advertisers pay advertising ad network for each customer action (e.g., downloading and installing an app), and ad networks pay publishers a fraction of the revenue for each user action on their apps.

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

(1) An ad request is sent from an ad publisher to the ad server of 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, while the user is redirected to the appropriate app store based on their device OS.
(4) The impression and click will then be recorded with the MMP (Mobile Measurement Partner, an independent third-party platform connecting between advertisers and publishers.), e.g., Adjust [4], Appsflyer [5], 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 MMP's attribution algorithm matches the app's install data with its impression or click engagement data and records it in an attempt to see if the user is authentic or not. Once an ad engagement is matched to the app install the user is labeled as non-organic. The credit for their install will be attributed to the appropriate ad network.

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

Mobile ad fraud implications trickle down to each and every aspect of the mobile advertising marketing initiatives. The most obvious implication is the direct financial loss fraud creates. According to Scalarr's latest mobile ad fraud report [44], Scalarr's projections for 2020 estimate losses close to the $16.1 billion mark as the result of mobile ad fraud. Also mobile ad fraud can ruin the data accuracy and lead advertisers and ad networks into investing and reinvesting in bad publishers (channels) due to the pollution of data being analyzed. Above anything else, fraud is an enormous waste of time and human resources. A lack of fraud treatment could mean losing an ad network's reputation and risking its future business with leading advertisers. Although advertisers and ad networks seek to defend themselves in many different ways, fraudsters have become more sophisticated, using increasingly complex programs and scripts which are able to spoof the source of the clicks and perform the fraud automatically.

Among all the commonly seen fraudulent tricks such as *click injection*, *click spamming*, *device farm* and *bots*. According to the latest report of Appsflyer, they have blocked over 1.6 billion fraud installs over the last three years, the fraud installs resulted by bots accounts for more than 56% for the entire mobile ad install fraud activities [1]. Bots are malicious codes that run a set program or action. While bots can be based on real devices, most bots are server-based. Bots aim to send clicks, installs and in-app events for installs that never truly occurred. Bots install fraud is considered one of the most difficult types of mobile ad fraud to detect. Bots can be applied to automate any action within the user flow or the app itself. They can be used to mimic real user behavior based on behavioral bio-metrics data collected by malware on user devices. Fraudsters adapt to advanced detection logic and train bots to run through in-app engagement measurement points appear as engaged "real" users and evade updated detection rules.

In the context of mobile app install fraud which our work aims to tackle with, 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 some user clicks the ads from this ad channel and installs the app, given that 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 collect money from advertisers without subsequent user actions.

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 install fraud detection in the context of mobile advertising, especially bots install fraud. At Mobvista there are over four million app installs per day. Hence in this work, our goal is to detect bots install fraud at Mobvista such that we can prevent the advertisers' budget from being wasted and the reputation of Mobvista being corrupted.

The fraud detection in mobile advertising is a challenging task, due to the rapid evolving fraud techniques. For example, fraudsters often adjust their fraudulent behaviors accordingly to escape the predefined rules, so traditional rule-based fraud detection systems are usually difficult to adapt to novel anomalies as well as the changing and growing data in face of adversaries. Many of the proposed methods resort to techniques such as ensemble methods, e.g., XGBoost [6], LightGBM [24] . Although these methods are capable of mining rich fraud patterns with sophisticated feature engineering, they are unable to model various types of relations among entities with local structural information.

Recent years have witnessed a growing interest in developing deep-learning based algorithms on graph, e.g., Node2Vec [17], DeepWalk [40], GCN [26], GraphSAGE [19], GAT [50].

In order to unleash the power of graph-based model, we have to address two challenges. The first one is how to construct a suitable graph. The constructed graph should potentially represent

the interaction behaviors between entities such as publisher (channel), users, and ads. The other is an efficient graph learning method should be designed to learn the useful structural and semantic representation from constructed graph, particularly heterogeneous graph [14, 61].

To address the aforementioned challenges, we propose a novel graph-based machine learning method which aims to remedy several drawbacks in BotSpot [58]. Our main contributions can be summarized as following:

(1) We propose to leverage domain knowledge and apply to sparse graph, 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 multi-head self attention module to enhance the interaction among various leaf nodes, which is more adequate to retrieve knowledges from leaf nodes compared with simple average operation.
(3) Extensive online experiments has shown that our proposed method is able to detect more bots installs with the threshold found in the validation dataset when models' precision are fixed to 90%, which proves its robustness given that bots patterns changes dynamically.
(4) Through our experiments, some observation can be made towards the deficiency of the current mechanism design, thus shed light on the future direction of the model improvement.

The remainder of this paper is organized as follows. We firstly introduce the related work in Section 2. Secondly, we describe our proposed method in Section 3. Then we describe our experimental settings and analyze the results in Section 4. Finally, we make some conclusions of our paper in Section5.

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. Haidern et al. [18] proposed an ensemble based method (e.g., J48 [42], RandomForest [2]) to classify each impression, as fraudulent or non-fraudulent. To handle the issue of imbalanced dataset, they have also applied SMOTE [5]. Perera et al. [39] proposed a novel ensemble model which based on 6 different learning algorithms approach for click fraud detection in mobile advertising. Zhang et al. [62] proposed a combination of Cost-Sensitive Back Propagation Neural network (CSBFNN) and the novel Artificial Bee Colony (ABC) [23] algorithm in their research towards detection of click fraud. They optimized feature selection process with BPNN connection weights using ABC to alter the relation between feature and weights. Oentaryo et al. [37] find that 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. [47] proposes a novel framework for prediction of click fraud in mobile advertising which consists of feature selection using Recursive Feature Elimination (RFE) and classification through Hellinger Distance Decision Tree (HDDT). They chose RFE for the feature selection as it has given better results as compared to wrapper approach when evaluated using different classifiers. To deal with class imbalance issue present in the data set, the use HDDT as as classifier. Dou et al. [11] investigates the problem of download fraud in App Market by setting up a honeypot to capture the ground truth fraudulent activities, and identify the download fraud leveraging feature engineering and XGBoost [6]. However, the aforementioned approaches rely on

extracting robust features and robustness of the detection algorithms, since the fraudulent patterns are constantly changing.

For install fraud detection in mobile advertising, inspired by the work of [29], Yao et al. [58] propose an anti-fraud method based on heterogeneous graph that incorporates both local context and global context via graph neural networks (GNN) and gradient boosting classifier to detect bots fraud installs at Mobvista.

2.2 Graph-based Fraud Detection

Inspired by the work of Mikolov et al. [34], various forms of graph embedding methods (e.g., Node2Vec [17], DeepWalk [40], LINE [48], CRSAL [43], SHCN [8], and GHARL [7]) have been proposed to extract the node embeddings from graph without requiring any labels. However, node features are not utilized. Graph neural network is further presented, in which the limitations of graph embedding are compensated. One of the most prominent progress is known as Graph Convolutional Networks (GCN) [26], which exploits multiple propagation layers to aggregate neighborhood information and enlarge the receptive field for each node in the graph. GCN achieves significant improvements compared to previous graph-mining methods such as DeepWalk. Hamilton et al. [19] 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 GCN in transductive settings. Graph Attention Network (GAT) [50] further introduces the masked self-attentional layers [10] into graph convolutional network. By leveraging masked self-attentional layers, GAT can assign different importances to different nodes within a neighborhood.

Wang et al. [51] present a graph-based spam detection method to identify suspicious reviewers. To detect the spam reviews in Xianyu (a second-hand goods app), Li et al. [29] proposed GAS which based on graph convolutional networks. 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. [55] proposed FeatNet which incorporates device features and device relationships in network representation learning to detect fraud device. Breuer et al. [3] study the problem of early detection of fake user accounts on social network. Nguyen et al. [36] 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. [32] 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 propose 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 to identify the fraudulent apps in mobile advertising, Hu et al. [21, 22] proposed two models, namely iBGP and GFD respectively.

3 PROPOSED METHOD

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

Next, we will describe the model architecture of our proposed BotSpot++ from the top level. As shown in Fig. 2, our BotSpot++ concats the heterogeneous graph neural networks information and pre-trained GBDT model information and classifies the samples through a MLP model.

Table 1. **Summary of the Main Notations**

Notation	Explanation
$G(V, E)$	A Graph with the set of vertexes and the set of Edges, shorted as 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.
$K^{classes}$	the number of classes.
K^{trees}	the number of weak tree learners.
α	the hyper parameter 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
$Neighbor\_Sampling(\bullet)$	A function for sampling neighbors
$Device\_Clustering(\bullet)$	A device clustering function for constructing super device nodes
$Node\_Aggregation(\bullet)$	A differentiable function for node aggregation
$In\_Super\_Nodes(\bullet)$	A function to get all origin device nodes in 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 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-campagin c, $\forall c \in C$
$\mathbf{h}_v^{l-1}$	The vector representation for vertex v at the $(l-1)^{th}$ layer
$\mathbf{W}_l^{bots}$	The weight matrix of a linear layer of bots devices at the l^{th} layer
$\mathbf{W}_l^{normal}$	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 l^{th} layer.
$\mathbf{s}_v^{normal,l}$	The attention scores of normal install for vertex i at 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 that in super device node q

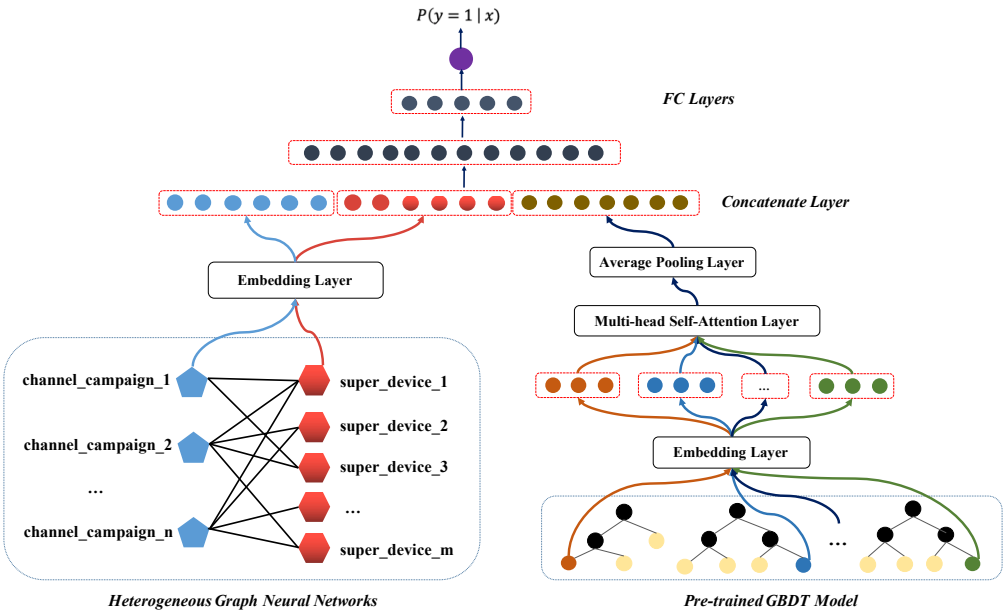

Fig. 2. **Model Architecture of BotSpot++.**

3.1 Preliminary

Graph neural networks are introduced in [16] 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 [4] a model based on 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 Laplacian matrix. [9] introduce a graph neural network with polynomial parametrization for the localized filter to reduce the number of model parameters to alleviate overfitting and reduce computational cost. [26] further refine the model discussed above by proposing a layer-wise linear model where the degree of Chebyshev Polynomial 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 GCN [26], the author further proposes to add self-loop to keep the eigenvalue of the graph matrix to be less than 1, hence avoiding the issue of gradient explosion. The multi-layers propagation rule in GCN is shown in Eq. 1:

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

where $H^{(l+1)}$ is the node representation at the $(l+1)^{th}$ layer, $\tilde{A}$ is the adjacency matrix with self-loop, $\tilde{D}$ is the diagonal matrix in graph G with self-loop, and finally $W^{(l)}$ is the graph convolutional filter at the l^{th} layer.

Hamilton et al. [19] further propose a more generalized version of graph neural networks based on [26], where computational subgraph is built for every vertex in inductive manner, and node representations can be extracted in a bottom-up approach leveraging various kinds of message passing mechanisms such as Averaging, Pooling and LSTM-based aggregation. The propagation rule for GraphSAGE is shown in Eq. 2:

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

where $\mathcal{N}(\bullet)$ is the neighboring function, h_v^k is the node representation for vertex v at the layer k.

Veličković et al. [50] incorporate attention mechanism and multi-head attention to improve the expressive power of each vertex and achieves a new SOTA on various datasets.

Currently, many new deep learning based graph representational learning algorithms has been proposed [15, 31, 52, 54, 60], and is applied in various fields [30, 41, 57, 59]. In this work, we propose a novel algorithm which can be adapt to sparse graph for rare event detection. The underlying notion is to leverage both global context and local context to facilitate the downstream classification problem.

3.2 Problem Definition

As in [58], we also formulate our problem as a edge classification problem on a sparse graph. The graph **BiG** is a heterogeneous bipartite graph with two sets of nodes, namely device nodes D^{origin} and channel-campaign nodes $\mathbf{C}$. A device node represents a mobile device where the user clicked a ad and download the corresponding app. A channel-campaign node represents an ad campaign presented on a ad channel. An edge between a device node and a channel-campaign node represents an app install by a given device node $d \in D^{origin}$ which is attributed to an ad channel-campaign $c \in C$. Our main task is to classify every edge $e_{dc}\left(d \in D^{origin}, c \in C\right)$ into normal edge or bots fraud edge.

3.3 Label Acquisition

In this work, we gather feedbacks of whether an app install is normal install or bots fraud install from advertisers or third-party to annotate each data sample, 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 [58], BotSpot is proposed to tackle this problem by first obtaining the representation of channel-campaign node using tailored message passing scheme to reflect its preference to bots and normal devices which is called local context in [58]. Furthermore, the prior knowledge from Gradient Boosting Classifier is extracted using leaf embeddings, which serves as global context. By leveraging local context and global context, BotSpot is able to classify each edge in the graph with a more holistic view. However, due to the sparsity of the formulated graph, many device nodes merely connects to a single channel-campaign node. As a result, in BotSpot the device nodes only employ FFCN for feature extraction, without incorporating any relational information. Intuitively, this might lead to some issues while extracting device-side features. In this work, we ask the following question: is it possible to adopt domain knowledge to enrich the graph structure and find ad channels with similar patterns to facilitate the message passing for device nodes to better capture its representations in the embedding space, and how to 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 problems mentioned above.

3.5 Proposed Method

In this subsection, we detail how to enhance BotSpot in a variety of algorithmic aspects, so as to address the modeling issues in BotSpot.

3.5.1 Message Passing for Channel-Campaign Node. For channel-campaign node, we leverage the same message passing mechanism as in BotSpot, that is, the explicit normal neighbors and bots neighbors of a channel-campaign node is leveraged separately together with attention mechanism, which helps the channel-campaign node to learn a biased representation towards bots or normal installs. Mathematically, the tailored message passing mechanism can be expressed as Eq. 3:

$$
\begin{aligned}
&h_v^{bots,l} = W_l^{bots} \bullet Node\_Aggregation_l\left(\left\{h_b^{bots,l-1}\right\}\right), \forall b \in Neighbor\_Sampling(v),\\
&h_v^{normal,l} = W_l^{normal} \bullet Node\_Aggregation_l\left(\left\{h_b^{normal,l-1}\right\}\right), \forall b \in Neighbor\_Sampling(v),\\
&z_v^l = \lambda_{\mathrm{v}}^{bots,l} h_v^{bots,l} + \lambda_{\mathrm{v}}^{normal,l} h_v^{normal,l}, \forall v \in D^{origin},\\
&s_v^{bots,l} = FFCN_\theta\left(q_v^{l-1} \| h_v^{bots,l}\right),\\
&s_v^{normal,l} = FFCN_\theta\left(q_v^{l-1} \| h_v^{normal,l}\right),\\
&\lambda_v^{t,l} = \frac{\exp\left(s_v^{t,l}\right)}{\sum_{t \in T} \exp\left(s_v^{t,l}\right)}, \forall t \in T = \{bots, normal\},
\end{aligned}
\tag{3}
$$

where $h_v^{bots,l}$ is the obtained node representations for vertex v at the l^{th} layer for bot devices, and W_l^{bots} is an affine layer for linear transformation of bot devices. Similarly, $h_v^{normal,l}$ is the node representation for vertex v at layer l for normal devices, W_l^{normal} is another set of model parameters for affine transformation. $s_v^{normal,l}$ and $s_v^{bots,l}$ are the attentional scores for bots and normal installs respectively. 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 $s_v^{normal,k}$ and $s_v^{bots,l}$. By performing a linear combination over h_v^l and g_v^l, the representation over channel-campaign node v is able to capture its preference towards bots and normal installs.

3.5.2 Message Passing for Device Node. In BotSpot, representations for device node is extracted using a 2-layer MLP, where graph convolution is not employed. Intuitively, it would be better if some relational information could be leveraged for device nodes to extract a better representation that is aware of its "neighboring" ad channels. In this work, we rely on domain knowledges to enrich the graph structure, that is, some devices are grouped into super devices if they meet some predefined criterion(s), thus we can treat each devices in the super device node as connecting to several or dozens of channel-campaign nodes that the super device node associates to. In many clustering algorithms [20, 35], the key idea underlies is how to measure similarity among the data points in the original space or latent space, and employ some optimizing algorithms to cluster those data points such as EM or DBSCAN [13]. In our scenario, the off-the-shelf algorithms won't help us to obtain the super device node for several reasons. First, the similarity measures based on the device features- country, language and device brand etc. - doesn't lead to high-quality clusters(super nodes) in terms of channel similarity, meaning that connected channels for a given super device node may not be similar to each other, hence won't benefit the message passing for the device nodes inside the super device node. Second, it will be inefficient for the clustering algorithm to cluster million of devices, which might take quite a long time. Hence in this work, we rely on

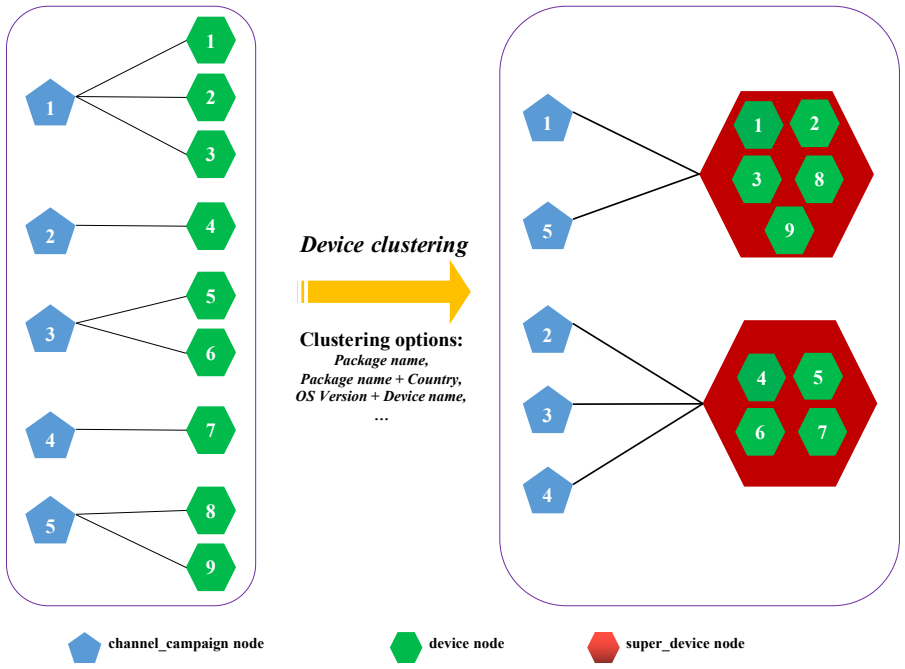

Fig. 3. **An illustration of forming super device nodes.**

expert knowledge to find super device node, by incorporating human knowledges we expect that the super device nodes connect to channel-campaign nodes which is more likely to be similar to each other, in terms of audience, geological locations and traits of ad publishers. Specifically, in this work we adopt *package_name* as the criterion for device node aggregation as ad networks typically distribute the same apps to multiple ad channels with "similar" advertising effect and similar audience. As a result, the devices in a same super device node aggregated by *package_name* can be linked to multiple "similar" ad channels, where similarity is gauged by the advertising experts' domain knowledge instead of similarity in feature space. Hence, by incorporating domain knowledge, the information can be propagated from similar ad channels to device nodes to enrich the representation for device nodes. The message passing scheme for super device node $q \in D^{super}$ and the device nodes $u \in q$ follows Eq. 4:

$$\begin{aligned} h_q^{super} &= Node\_Aggregation\left(x_i\right), \forall i \in Neighbor\_Sampling(q), q \in D^{super}, \\ h_{u,q}^{origin} &= h_q^{super} \| W^{origin} x_u, \forall u \in In\_Super\_Nodes(q), q \in D^{super}, \end{aligned} \tag{4}$$

where x_i denotes the feature vector for device node i and h_q^{super} denotes the convolved feature representation for super device node q, and $h_{u,q}^{origin}$ is the node representation for device node u which is inside super device node q, $In\_Super\_Nodes(i)$ is a function that returns set of device nodes inside super device node i, and W^{origin} denotes the weight matrix of affine transformation for origin device nodes.

The process of forming super device nodes according to various criterions is shown in Fig. 3. By forming super device nodes, hierarchical graph is built and each device is able to leverage similar channel-campaign nodes to learn more interactive representations, thus benefits downstream classification problem.

3.5.3 Incorporating High Cardinality Features. Some categorical features for device nodes serve as strong indicator for bots detection, thus to enhance the interaction between these features, dense embeddings are employed for several device-level features such as device brand, device language, device carrier.

3.5.4 Global Context Extraction. As in [58], in this work we also leverage a pretrained GBM for global context extraction as a fixed decision path in a decision tree corresponds to a series of decision criterion, thus the leaf embeddings $e_{ij} \in E\,(i \in [1,T], j \in [1,K])$ are extracted from a pre-trained LightGBM to enrich the expressive power of our proposed method, where there are T leaf nodes at maximum for each decision tree and K^{trees} decision trees in our pre-trained LightGBM, β is the embedding size for each leaf node. The total model parameters for the leaf embedding matrix E is $T \times K^{trees} \times \beta$. Thus E can be jointly trained end-to-end using optimization algorithms such as SGD and Adam [25]. For each data sample, there are total K^{trees} leaf embedding vectors extracted from the leaf embedding matrix $E \in R^{T \times K^{trees} \times \beta}$ to represent K^{trees} decision paths for every decision tree. We then leverage a aggregation function to obtain the final representation for the total K^{trees} leaf embeddings, where $q_s(i)$ is the indexing function that returns the leaf indice for the i^{th} decision tree for the data sample s. In BotSpot, $Average(\bullet)$ is employed as the aggregation function, however there are some drawbacks for this choice. First, it is a well-known fact that in a gradient boosting machine, the weak learner in the early stage contributes more the those in the later stages, hence would carry more information, yet a simple averaging probably can't capture this statistical pattern. Secondly, as leaf embedding $e_{q_s(j),j} \in R^H$ for each data sample s corresponds to a fixed decision path in the decision tree, the leaf embeddings $e_{q_s(j),j}$ can't attend to each other, thus may lead to some sub-optimality. Hence in this work, we resort to self attention [49] to remedy this drawbacks. Self-attention is widely exploited in many application areas in NLP and Computer Vision [10, 28, 49, 56]. In this work, we employ self-attention to obtain a higher-level representation for each leaf node, hence all the leaf node embeddings attends to the other leaf node embeddings, then the representation after self-attention module is averaged to get the final aggregated embedding. Eq.5 illustrates the entire process:

$$\begin{aligned} h^s_{q(j),j} &= SELF\_ATTENTION(e_{q_s(j),j}), \forall j \in [0, K^{trees}-1], \forall s \in S, \\ h^s_j &= \sum_j h_{q_s(j),j}, \forall j \in [0, K^{trees}-1], \end{aligned} \tag{5}$$

where $h^s_{q(j),j}$ is the representation for the data sample s of the j^{th} decision tree, and h^s_j is the aggregated representation for data sample s.

The final representation h^s_j is concatenated with g^s obtained from graph neural networks and is feed into a MLP with dropout and outputs a probability indicating whether it's a bot or normal install as illustrated in Eq. 6:

$$P(y = 1 \mid x; \theta) = \sigma\left(\mathrm{W}_2\left(\mathrm{W}_1\left(\mathrm{h}^s_j \| \mathrm{g}^s\right) + b_1\right) + b_2\right), \tag{6}$$

where $\sigma(\bullet)$ is the sigmoid function and θ is the model parameter of our proposed method.

3.5.5 Model Training. We resort to binary cross-entropy as the loss function and optimize the model parameters using Adam. As our label is obtained using the feedback from advertisers and third-party, the label may be noisy. Label smoothing [46] is leveraged to prevent the model from overconfident prediction. As shown in Eq. 7, the label y is first smoothed and then the cross-entropy loss is calculated using the smoothed loss over minibatch $S^{minibatch}$:

Algorithm 1 BotSpot++ Algorithm

Input: $BiG((D,C),E)$, $e_{d,c} \in E$, x_d, x_c, M^{embed}, K^{trees},linear transformation layer for channel-campaign node W^c, and similarly W^d for device node, non-linear transformation σ
Output: the estimated probability of input edge $e_{d,c}$ of being bots install

for edge in G **do**
  $d, c \leftarrow edge$
  $h_{Neighbor\_Sampling(c)} \leftarrow Node\_Aggregation(x_d), \forall d \in D$
  $h_c \leftarrow \sigma(CONCAT(W^c x_c, h_{Neighbor\_Sampling(c)}))$
  $h^{sup\_dev} \leftarrow Node\_Aggregation(x_i)$
    $\forall i \in Neighbor\_Sampling(Device\_Clustering(d))$
  $h_d \leftarrow CONCAT(W^d x_d, h^{sup\_dev})$
  $h^{local}_{d,c} \leftarrow CONCAT(h_d, h_c)$
  $t^j_{d,c} \leftarrow LEAF\_INDICE(M^{embed}, CONCAT(x_d, x_c)), \forall j \in 0, 1, ...K^{trees} - 1$
  $h^j_{d,c} \leftarrow LEAF\_EMBEDDING(E, t^j_{d,c})$
  $h^j_{d,c} \leftarrow SELF\_ATTENTION(h^j_{d,c}), \forall j \in 0, 1, ..K^{trees} - 1$
  $h^{global}_{d,c} \leftarrow AVERAGE(h^j_{d,c}), \forall j \in 0, 1, ...K^{trees} - 1$
  $P(e_{d,c} = 1|x) \leftarrow FFCN(CONCAT(h^{local}_{d,c}, h^{global}_{d,c}))$
end for

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

where y_s is label for sample s and $\bar{y}_s$ is the smoothed label for sample s. α is a hyper parameter that controls the smoothness degree, and $K^{classes}$ is the number of labels in the classification problem (i.e. 2).

And 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 datasets used in our experiments were publicly accessible[6]. which will be detailed in the next subsection. As our model is delicately designed for bots install fraud detection, the bots detection will be the main evaluation task. Thus, this principal research question is broken down into the following specific ones to guide our experimental studies.

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

RQ2. Does our proposed BotSpot++ outperform BotSpot model?

RQ3. How does the local mechanism affect the performance of BotSpot++?

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

RQ5. Why does our proposed method work?

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

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	e.g. en, zh
device_network	whether is wifi, e.g. True, False
device_carrier	mobile carrier, e.g. china mobile
device_name	e.g. samung
install_country	e.g. China
install_city	e.g. 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. #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.**

Dataset	#Dev	#Chan-Camp	#Normal Install(Train, Test)	#Bots Install(Train, Test)
dataset-1	1676101	2428	1249059, 163413	270827, 20561
dataset-2	1216796	2135	884970, 141546	884970, 13865
dataset-3	1313073	2073	1051838, 196119	139358, 9598
dataset-4	1299895	1995	1156202, 181953	77717, 12018

4.1 Dataset

To evaluate our proposed model more comprehensively, we built four datasets for different time periods. Each dataset mainly includes device information(such as device brand, device name, ip, etc.), geographic information(such as country, city), campaign information, channel information, which detailed as Table 2. It is worth to mention that the geographic information is actually obtained through ip information.

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

4.2 Feature Engineering

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

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

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, device_os. Considering the users' personal privacy issues, The column device_id is encrypted by md5. We filter out the abnormal data appearing in device_id by regular method and replace it with unknown_device_id. For example, the value "9f89c84a559f573636a47ff8daed0d33" in column device_id is a md5 hash value, and its original value is "00000000-0000-0000-0000-000000000000", which means a abnormal device id. Therefore we treated it as "unknown_device_id". And for column device_brand, we will merge some sub-brands into one big brand. For example, "redmi note" and "redmi" actually belong to Redmin series of Xiaomi Inc. Thus, we uniformly replace it with redmi. As for device_os, we truncated the os(operating system) version number of mobile phone, we only kept the first two digits of all os version number since some downstream channel only report the major os version number. In this way, the 8.1.5 will be transformed to 8.1. And we normalized other fields in the similar way, we will not detailed 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 calculates the repetition of device or geographic information in different channels, such as ip repetition features, which are important features for bots install 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 etc.

(2) categorical features
The category features are mainly composed of device information, geographic information, campaign information, channel information, which detailed in Table 2. We deal with these categorical features differently for different models. we use label encoder to encode categorical features in LightGBM model, due to it directly support the processing of categorical features instead of using one-hot encoder. As for 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 towards 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(PCA) [53] and wrapper subset evaluation [27] with our model. And we chose wrapper method at last since it resulted in the highest performance compared to PCA. After feature engineering, we get a total of 140 features, which will be used in 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. These baseline models detailed as below:

LightGBM: LightGBM, widely adopted by industry, is a well-known gradient boosting framework that uses tree based learning algorithm. We apply Grid Search to find optimal parameters. Finally, we set the hyper-parameters of our baseline model to 500 decision trees and the maximum depth to 5 except on dataset-4, on which its maximum depth is 4.

MLP: We developed a 3-layer MLP model with dropout mechanism [45] to avoid overfitting. We set the embedding dimension to 16, the number of neuron units to 64 and the probability of dropout to 0.1.

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. The GraphSAGE model extracts node features and obtain features for each edge which are fed into a MLP for final edge classification.

BotSpot: BotSpot is an anti-fraud method based on heterogeneous graph that incorporates both local context and global context via graph neural networks (GNN) and gradient boosting classifier to detect bots fraud installs at Mobvista. Due to our work is based on this work, we also take it as our baseline model.

BotSpot_local: BotSpot_local is similar with BotSpot, except that BotSpot_local don't use gradient boosting leaf embedding information.

4.4 Experimental Settings

For performance evaluation, we chose to adopt the *Recall@N%Precision* as our basic criterion, which detailed in Eq. 10. Before we introduce *Recall@N%Precision*, let us understand how to calculate the *Precision* and *Recall* as shown in Eq. (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* equal to $N\%$ as:

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

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

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

In the scenario of mobile bots fraud detection, We must ensure high model precision to reduce the false positive rate, which avoid damaging advertiser' business and the reputation of advertising platform. This is because that false positives indicate cases where installs were falsely identified as fraudulent, and may potentially penalizes legitimate channels. This could harm the relationship with its quality channels 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, 80 to compare the performance of different models. We abbreviate Recall@N%Precision to Recall@N%P in the following.

We implement the LightGBM using Microsoft LightGBM library and other deep learning models using PyTorch [38], which are publicly accessible[7].

4.5 Results and Analysis

In this section, we will compare the experimental results using the baseline models mentioned in the last subsection with our BotSpot++ model. We address **RQ1**, **RQ2** and **RQ3** by evaluating our BotSpot++ model on multiple different time period offline datasets introduced in Chapter 4.1. Then, to address **RQ4**, we deploy LigthGBM, MLP, GraphSAGE, BotSpot and our proposed BotSpot++ on the real online production environment to compare their performance. Finally, we address **RQ5** by conducting a case study.

4.5.1 Offline Experiments. Firstly, we answer **RQ1**, **RQ2** and **RQ3** by evaluating Recall@90%P, Recall@85 %P and Recall@80%P with our proposed model and baseline models on four different time period datasets collected from advertising platform of Mobvista.

For **RQ1**, as shown in the tables for various durations, our proposed BotSpot++ is able to outperform all the other competitive baseline methods given a relatively high fixed precision, that is, BotSpot++ is able to recall more bots given the same precision, e.g. 80%, 85% and 90% compared with other methods, which proves the effectiveness of incorporating domain knowledges into the graph-structured data that benefits the information flow of device nodes. However, we should also note that for some time periods, the component for local context extraction doesn't outperform other graph-based baseline methods such as in dataset-4 when at Recall@80%P. The possible reason might be that the grouping of super device nodes leveraging domain knowledges may introduce noise in some cases as domain knowledge sometimes is not so reliable, e.g., some packages may be distributed to ad channels in an ad hoc manner for exploration, as we leverage mean pooling as the aggregation function, some noise is introduced, this might get even worse as super device nodes in most cases only connect to a few channel-campaign nodes. Another observation can be made that BotSpot_local doesn't outperform GraphSAGE in some periods such as in dataset-4 and dataset-3 when at Recall@85%P and Recall@80%P, which reflect some drawbacks in the mechanism design of the message passing scheme, the tailored mechanism tends to extract representations over the preference towards bots and normal installs with respect to the channel. However, for some normal ad channels, the extracted representation might hurt the performance for the bots associated with it. As a result, a simple average operation adopted in GraphSAGE might be a better choice. We leave this issues as an open problem and will address them in our future work. Although with some defect in the mechanism design, BotSpot++ still outperforms other baseline models by incorporating domain knowledge and prior knowledge from LightGBM. By leveraging the hierarchical graph structure, device nodes tend to be aware of other similar ad channels to extract a better representation for downstream tasks. Another reason that might lead to a better predictive performance for BotSpot++ would be the self-attention module introduced for leaf embeddings, which helps the leaf nodes to be aware of other leaf nodes in both directions. We argue that this will fix some internal deficiency in LightGBM as each week learner(decision tree) is trained in a cascaded manner, thus each decision tree can't interact with the others behind it. As a result, BotSpot++ is able outperform all other competitive baseline methods in most scenarios by at least 5.90% in terms of Recall@90% P.

For **RQ2**, it is obvious to observed that BotSpot++ outperforms BotSpot in all the four offline datasets at all the metrics, which means performance gain is attributed to the introduction of super device nodes and self attention mechanism on leaf embedding. The Recall@90%P, Recall@85%P, Recall@80%P is generally improved about 1% to 2% on all the datasets. It is worth to mention that

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

Table 5. **Evaluation results for dataset-1. LightGBM is the base model, +x% represents the percentage of current model performance improvement compared with LightGBM. Recall@N%Precision shorted as Recall@N%P. The bold number denotes the best performance of all the models.**

	Recall@90% P	Recall@85% P	Recall@80% P
LightGBM	0.1999	0.2189	0.2356
MLP	0.2408(+21.01%)	0.2688(+22.80%)	0.2830(+20.12%)
GraphSAGE	0.2459(+23.57%)	0.2807(+28.23%)	0.3196(+35.65%)
BotSpot_local	0.2654(+33.37%)	0.2901(+32.53%)	0.3296(+39.90%)
BotSpot	0.2601(+30.70%)	0.2959(+35.18%)	0.3329(+41.30%)
BotSpot++_local	0.2604(+30.85%)	0.2898(+32.39%)	0.3294(+39.81%)
BotSpot++	**0.2674(+34.37%)**	**0.3014(+37.69%)**	**0.3376(+43.29%)**

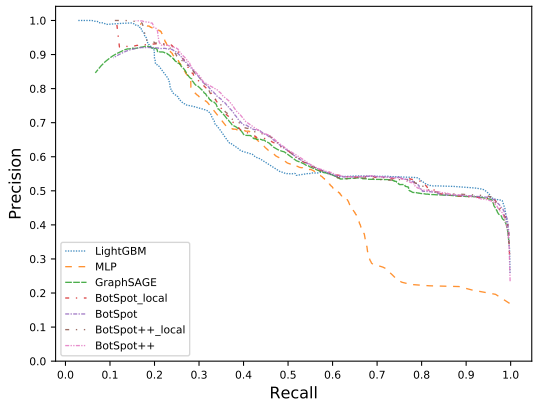

Fig. 4. **Precision-Recall Curve for dataset-1**

Recall@90%P is improved about 8%, Recall@85%P is improved about 5.5% on dataset-4. Therefore, the grouping of super device nodes and self-attention mechanism leveraging domain knowledges and context information respectively can achieve better performance.

For **RQ3**, we can find that BotSpot++ model outperform BotSpot++_local on four different datasets at Recall@90%P, Recall@85%P and Recall@80%P. All the metrics are improved 5% to 7% on dataset-2 and dataset-4, 1% to 2% on dataset-1 and dataset-3. This is because that comparing with BotSpot++_local, BotSpot++ has integrated global-context information and local-context information.

Table 6. **Evaluation results for dataset-2 of offline datasets. LightGBM is the base model, +x% represents the percentage of current model performance improvement compared with LightGBM. Recall@N%Precision shorted as Recall@N%P. The bold number denotes the best performance of all the models.**

	Recall@90% P	Recall@85% P	Recall@80% P
LightGBM	0.3808	0.4031	0.4237
MLP	0.4032(+5.88%)	0.4195(+4.07%)	0.4636(+ 9.42%)
GraphSAGE	0.4058(+6.57%)	0.4281(+6.20%)	0.4561(+ 7.65%)
BotSpot_local	0.4023(+5.65%)	0.4312(+6.97%)	0.4532(+ 6.96%)
BotSpot	0.4067(+6.80%)	0.4302(+6.72%)	0.4683(+10.53%)
BotSpot++_local	0.3952(+3.78%)	0.4203(+4.27%)	0.4572(+ 7.91%)
BotSpot++	**0.4175(+9.64%)**	**0.4401(+9.18%)**	**0.4812(+13.57%)**

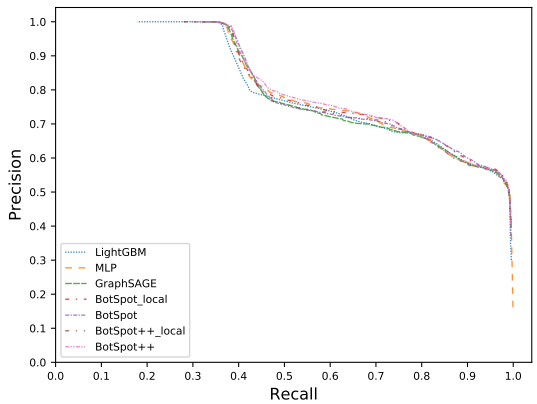

Fig. 5. **Precision-Recall Curve for dataset-2**

Table 7. **Evaluation results for dataset-3 of offline datasets. LightGBM is the base model, +x% represents the percentage of current model performance improvement compared with LightGBM. Recall@N%Precision shorted as Recall@N%P. The bold number denotes the best performance of all the models.**

	Recall@90% P	Recall@85% P	Recall@80% P
LightGBM	0.4344	0.4410	0.4502
MLP	0.4436(+2.42%)	0.4537(+2.88%)	0.4653(+3.35%)
GraphSAGE	0.4671(+8.59%)	0.4721(+7.05%)	0.4782(+6.22%)
BotSpot_local	0.4586(+6.36%)	0.4672(+5.94%)	0.4701(+4.42%)
BotSpot	0.4672(+8.61%)	0.4739(+7.46%)	0.4813(+6.91%)
BotSpot++_local	0.4651(+8.06%)	0.4758(+7.89%)	0.4811(+6.86%)
BotSpot++	**0.4712(+9.66%)**	**0.4788(+8.57%)**	**0.4872(+8.22%)**

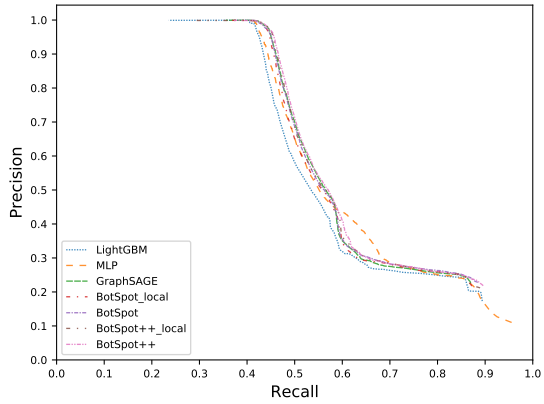

Fig. 6. **Precision-Recall Curve for dataset-3**

Table 8. **Evaluation results for dataset-4 of offline datasets. LightGBM is the base model, +x% represents the percentage of current model performance improvement compared with LightGBM. Recall@N%Precision shorted as Recall@N%P. The bold number denotes the best performance of all the models.**

	Recall@90% P	Recall@85% P	Recall@80% P
LightGBM	0.4493	0.4711	0.5069
MLP	0.4971(+ 8.85%)	0.5377(+9.58%)	**0.5676(+11.97%)**
GraphSAGE	0.5054(+10.66%)	0.5294(+7.89%)	0.5419(+ 6.91%)
BotSpot_local	0.5095(+11.56%)	0.5259(+7.17%)	0.5394(+ 6.41%)
BotSpot	0.5039(+10.34%)	0.5319(+8.41%)	0.5458(+ 7.67%)
BotSpot++_local	0.5159(+12.96%)	0.5243(+6.85%)	0.5392(+ 6.37%)
BotSpot++	**0.5468(+19.73%)**	**0.5613(+14.39%)**	0.5674(+11.94%)

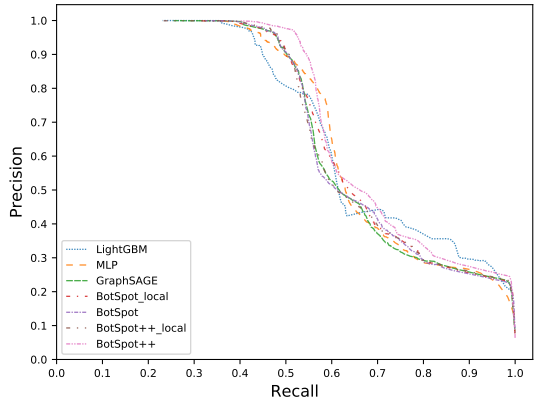

Fig. 7. **Precision-Recall Curve for dataset-4**

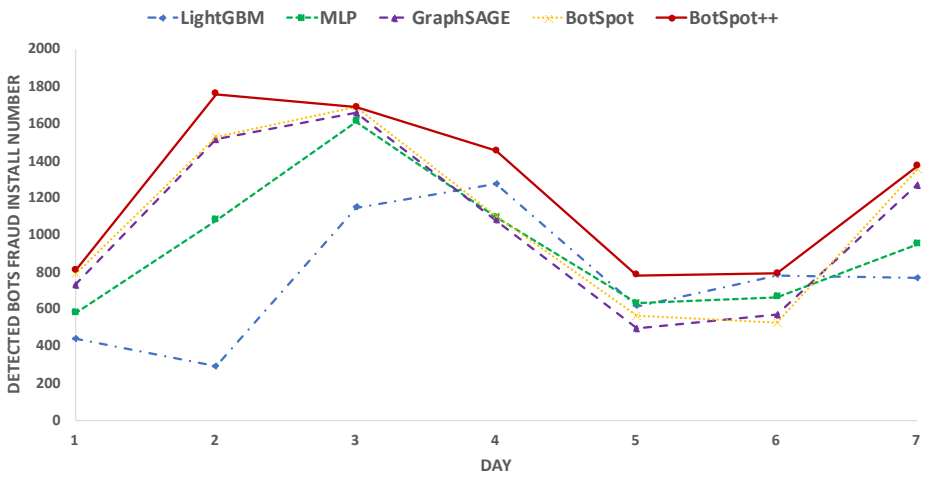

Fig. 8. **Online evaluation result. The horizontal axis represents 7 consecutive days, and the vertical axis represents the number of bots detected by different models when their precision is fixed to 90 %**

Table 9. **Total number of bots install detected during online evaluation. The proposed methods outperforms the baseline models at Mobvista.**

	LightGBM	MLP	GraphSAGE	BotSpot	BotSpot++
Total	5321	6612	7317	7554	8655

4.5.2 Online Experiments. Then we answer **RQ4** by conducting a series 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. In online experiments, We deploy LightGBM, MLP, GraphSAGE, BotSpot and our proposed BotSpot++ on our advertisement platform. Then we evaluate the model performance by comparing the number of bots detected by different models. In detail, we determine the split threshold by searching it to make the models reach 90% precision on validation data, and then compare the number of bots installs detected by this threshold on test data, which are checked by human experts.

As shown in Figure 8, we can find that our proposed BotSpot++ consistently outperforms the BotSpot and other baseline models. We also note that the GraphSAGE not always outperform MLP and MLP not always outperform LightGBM too. This is because that downstream traffic and bots fraud patterns are constantly changing, but generally speaking, as shown in Table 9, the performance of different models is regular, which is that the BotSpot++ has the best performance, BotSpot is second, LightGBM has the worst performance.

4.5.3 Case Study. We study the model capacity of BotSpot++ by examining the recalled ground-truth bots installs from them and analyze the portion of positive samples which is recalled by BotSpot++ yet is ignored by BotSpot and GraphSage. Yao et al. [58] has 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 recalled by BotSpot yet not by the other methods is situated in a ad channel where the device brand distribution is a strong indicator to identify bots and normal installs. In this work, we focus on those

Table 10. **An illustration of several bots installs recognized by BotSpot++ yet not by BotSpot and GraphSAGE. 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_num_50% be the number of ad channels connecting to the super device node with bots ratio greater than 50% and finally sup_dev_chan_bots_ratio_max be the maximum bots ratio of the ad channels connecting to the super device node.**

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%

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 totally 853 bots installs recalled, among which 107 bots are only recalled by BotSpot++, and 13 bots recalled by the other two methods yet not by BotSpot++. We then compare the 107 bots with other bots recalled by all the three methods, and found that there are 44 bots situated in 5 ad channels which are not presented in the other positive samples. Furthermore, the *package_name* of these installs are rarely seen in the entire population of bots installs. However, we should note that *package_name* is not a part in the feature matrix, instead, it is our domain knowledge that help to aggregate device nodes into super device node to facilitate the information flow from multiple ad channels to the corresponding devices. We observe that the bots ratio of these 5 ad channels is between 0.07 and 0.2, meaning that most installs are normal installs and these 44 bots 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 bots ratio ranging between 0.51-0.94, we argue that BotSpot++ is able to detect these 44 bots utilizing the propagated information from the multiple high-risk ad channels. In table 10, we illustrate several bots recalled only by BotSpot++ and statistics of their associated ad channels connected to both the bots device nodes and super device nodes.

4.6 Data Visualization

In order to observe the effect of the model more clearly and intuitively, we visualized the dense vector of the last full connected layer of our BotSpot++. And we adopt t-distributed stochastic neighbor embedding(t-SNE) techniques [33] to visualized the dense vector. We take the visualization results of dataset-1 and dataset-2 as an example, as Fig. 9 shown, the red dot in the figures indicate the bots installs, and the black dot indicate the normal installs. And we can find that the bots installs are distributed around, which are separated a large margin with normal installs on the dataset-1(left figure), while in the dataset-2(right figure), bots installs are mainly concentrated in the lower right corner.

5 CONCLUSION

Bots install fraud is a notoriously hard problem in mobile adverting industry for its dynamically changing behavioral pattern which causes great economical losses for advertisers and undermines

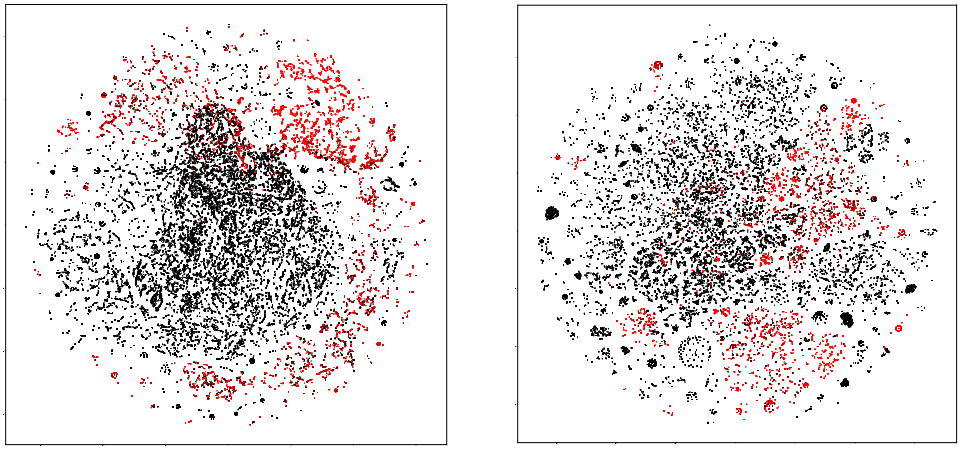

Fig. 9. **Visualization of the last FC layer data of BotSpot++ with dataset-1(left) and dataset-2(right). The red dots indicate the bots fraud installs, and the black dots indicate the normal installs**

the ecosystem of mobile advertising industry. In this work, we propose BotSpot++, which is an enhanced version of BotSpot [58]. Our proposed method remedy several issues in BotSpot, where the underlying notion of leveraging local context and global context remains the same, however, domain knowledge is adopted to enrich the graph structure to facilitate the information flow for the device nodes by proposing super device node which connects to similar ad channels. We show that by enlarging the receptive field of device nodes, the model capacity increases and leads to better predictive performance. Self attention module is also incorporated in BotSpot++ to enhance the interaction among leaf nodes, which is inherently an issue in GBM. However, we show that there are still some deficiencies in the internal mechanism design that needs to be fixed in terms of the information flow of both directions, which will be addressed in the future. We will also try to incorporate global context in a more complete fashion, rather than leveraging leaf embedding which requires a pretrained GBM that increases model training time drastically.

REFERENCES

[1] AppsFlyer. 2019. 2019 Fraud Trends Uncover Fascinating Results. https://www.appsflyer.com/blog/mobile-ad-fraud-trends/. Accessed on 2020-09-15.

[2] L. Breiman. 2004. Random Forests. *Machine Learning* 45 (2004), 5–32.

[3] Adam Breuer, Roee Eilat, and Udi Weinsberg. 2020. Friend or Faux: Graph-Based Early Detection of Fake Accounts on Social Networks. *Proceedings of The Web Conference 2020* (2020).

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *ICLR*.

[5] Nitesh V. Chawla, K. Bowyer, L. Hall, and W. P. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.* 16 (2002), 321–357.

[6] T. Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).

[7] Xiaolin Chen, Xuemeng Song, Ruiyang Ren, Lei Zhu, Zhiyong Cheng, and Liqiang Nie. 2020. Fine-Grained Privacy Detection with Graph-Regularized Hierarchical Attentive Representation Learning. *ACM Transactions on Information Systems (TOIS)* 38, 4 (2020), 1–26.

[8] Xu Chen, Kun Xiong, Yongfeng Zhang, Long Xia, Dawei Yin, and Jimmy Xiangji Huang. 2020. Neural Feature-aware Recommendation with Signed Hypergraph Convolutional Network. *ACM Transactions on Information Systems (TOIS)* 39, 1 (2020), 1–22.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.

[10] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.
[11] Yingtong Dou, Weijian Li, Zhirong Liu, Zhenhua Dong, Jiebo Luo, and Philip S. Yu. 2019. Uncovering Download Fraud Activities in Mobile App Markets. *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (2019), 671–678.
[12] eMarketer. 2020. Mobile Ad spending, Worldwide. https://forecasts-na1.emarketer.com/5a4e4662d8690c0c28d1f233/58a32412bad7b702a0802ec2. Accessed on 2020-09-15.
[13] M. Ester, H. Kriegel, J. Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*.
[14] Shaohua Fan, Junxiong Zhu, Xiaotian Han, C. Shi, Linmei Hu, Biyu Ma, and Yongliang Li. 2019. Metapath-guided Heterogeneous Graph Neural Network for Intent Recommendation. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019).
[15] M. Fey. 2019. Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks. *ArXiv* abs/1904.04849 (2019).
[16] Marco Gori, Dipartimento Ingegneria, and Gabriele Monfardini. 2005. A New Model for Learning in Graph Domains. In *IJCNN*.
[17] Aditya Grover and J. Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).
[18] C. Haider, A. Iqbal, A. Rahman, and Mohammad Sohel Rahman. 2018. An ensemble learning based approach for impression fraud detection in mobile advertising. *J. Netw. Comput. Appl.* 112 (2018), 126–141.
[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
[20] J. Hartigan and M. Wong. 1979. A k-means clustering algorithm.
[21] J. Hu, Teng-Hui Li, Yi Zhuang, Song Huang, and Shoubin Dong. 2020. GFD: A Weighted Heterogeneous Graph Embedding Based Approach for Fraud Detection in Mobile Advertising.
[22] J. Hu, Junjie Liang, and Shoubin Dong. 2017. iBGP: A Bipartite Graph Propagation Approach for Mobile Advertising Fraud Detection. *Mob. Inf. Syst.* 2017 (2017), 6412521:1–6412521:12.
[23] D. Karaboga and B. Basturk. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization* 39 (2007), 459–471.
[24] Guolin Ke, Q. Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and T. Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NIPS*.
[25] Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *ICLR 2015*.
[26] Thomas Kipf and M. Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* abs/1609.02907 (2017).
[27] Ron Kohavi, George H John, et al. 1997. Wrappers for feature subset selection. *Artificial intelligence* 97, 1-2 (1997), 273–324.
[28] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *ArXiv* abs/1909.11942 (2020).
[29] A. Li, Zhou Qin, Runshi Liu, Yiqun Yang, and D. Li. 2019. Spam Review Detection with Graph Convolutional Networks. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2019).
[30] Y. Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and P. Battaglia. 2018. Learning Deep Generative Models of Graphs. *ArXiv* abs/1803.03324 (2018).
[31] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, R. Urtasun, and R. Zemel. 2018. Graph Partition Neural Networks for Semi-Supervised Classification. *ArXiv* abs/1803.06272 (2018).
[32] Ziqi Liu, Chaochao Chen, Xinxing Yang, J. Zhou, X. Li, and L. Song. 2018. Heterogeneous Graph Neural Networks for Malicious Account Detection. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018).
[33] L. V. D. Maaten and Geoffrey E. Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.
[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, G. S. Corrado, and J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *ArXiv* abs/1310.4546 (2013).
[35] A. Ng, Michael I. Jordan, and Yair Weiss. 2001. On Spectral Clustering: Analysis and an algorithm. In *NIPS*.
[36] Van-Hoang Nguyen, Kazunari Sugiyama, Preslav Nakov, and Min-Yen Kan. 2020. FANG: Leveraging Social Context for Fake News Detection Using Graph Representation. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (2020).
[37] R. J. Oentaryo, E. Lim, M. Finegold, D. Lo, F. Zhu, C. Phua, E. Cheu, Ghim-Eng Yap, Kelvin Sim, M. Nguyen, K. S. Perera, B. Neupane, M. Faisal, Z. Aung, W. Woon, W. Chen, Dhaval Patel, and D. Berrar. 2014. Detecting click fraud in online

advertising: a data mining approach. *J. Mach. Learn. Res.* 15 (2014), 99–140.
[38] Adam Paszke, S. Gross, Francisco Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, B. Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *ArXiv* abs/1912.01703 (2019).
[39] K. S. Perera, B. Neupane, M. Faisal, Z. Aung, and W. Woon. 2013. A Novel Ensemble Learning-Based Approach for Click Fraud Detection in Mobile Advertising. In *MIKE*.
[40] Bryan Perozzi, Rami Al-Rfou, and S. Skiena. 2014. DeepWalk: online learning of social representations. In *KDD '14*.
[41] Siyuan Qi, Wenguan Wang, Baoxiong Jia, J. Shen, and S. Zhu. 2018. Learning Human-Object Interactions by Graph Parsing Neural Networks. *ArXiv* abs/1808.07962 (2018).
[42] J. Quinlan. 1992. C4.5: Programs for Machine Learning.
[43] Xuhui Ren, Hongzhi Yin, Tong Chen, Hao Wang, Nguyen Quoc Viet Hung, Zi Huang, and Xiangliang Zhang. 2020. Crsal: Conversational recommender systems with adversarial learning. *ACM Transactions on Information Systems (TOIS)* 38, 4 (2020), 1–40.
[44] Scalarr.io. 2019. What to Expect From Mobile Ad Fraud in 2020. https://scalarr.io/blog/articles/mobile-ad-fraud-trends-2020/. Accessed on 2020-09-15.
[45] Nitish Srivastava, Geoffrey E. Hinton, A. Krizhevsky, Ilya Sutskever, and R. Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15 (2014), 1929–1958.
[46] Christian Szegedy, V. Vanhoucke, S. Ioffe, Jon Shlens, and Z. Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 2818–2826.
[47] Mayank Taneja, Kavyanshi Garg, Archana Purwar, and Samarth Sharma. 2015. Prediction of click frauds in mobile advertising. *2015 Eighth International Conference on Contemporary Computing (IC3)* (2015), 162–166.
[48] Jian Tang, Meng Qu, Mingzhe Wang, M. Zhang, Jun Yan, and Q. Mei. 2015. LINE: Large-scale Information Network Embedding. *ArXiv* abs/1503.03578 (2015).
[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *ArXiv* abs/1706.03762 (2017).
[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[51] G. Wang, Sihong Xie, B. Liu, and Philip S. Yu. 2012. Identify Online Store Review Spammers via Social Review Graph. *ACM Trans. Intell. Syst. Technol.* 3 (2012), 61:1–61:21.
[52] X. Wang, Houye Ji, C. Shi, Bai Wang, Peng Cui, P. Yu, and Yanfang Ye. 2019. Heterogeneous Graph Attention Network. *The World Wide Web Conference* (2019).
[53] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.
[54] Zhang Xinyi and Lihui Chen. 2019. Capsule Graph Neural Network. In *ICLR*.
[55] C. Xu, Zhentan Feng, Yizheng Chen, M. Wang, and Tao Wei. 2018. FeatNet: Large-scale Fraud Device Detection by Network Representation Learning with Rich Features. *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security* (2018).
[56] Z. Yang, Zihang Dai, Yiming Yang, J. Carbonell, R. Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS*.
[57] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph Convolutional Networks for Text Classification. *ArXiv* abs/1809.05679 (2019).
[58] Tianjun Yao, Q. Li, Shangsong Liang, and Yadong Zhu. 2020. BotSpot: A Hybrid Learning Framework to Uncover Bot Install Fraud in Mobile Advertising. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (2020).
[59] Ting Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*.
[60] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and H. Kim. 2019. Graph Transformer Networks. In *NeurIPS*.
[61] Chuxu Zhang, Dongjin Song, Chao Huang, A. Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019).
[62] Xin Zhang, X. Liu, and Han Guo. 2018. A Click Fraud Detection Scheme based on Cost sensitive BPNN and ABC in Mobile Advertising. *2018 IEEE 4th International Conference on Computer and Communications (ICCC)* (2018), 1360–1365.