NGUARD: A Game Bot Detection Framework for NetEase MMORPGs[†]

Jianrong Tao[*‡]
NetEase Fuxi AI Lab
hztaojianrong@corp.netease.com

Jiarong Xu[*]
Zhejiang University
xujr@zju.edu.cn

Linxia Gong[‡]
NetEase Fuxi AI Lab
gonglinxia@corp.netease.com

Yifu Li[‡]
NetEase Fuxi AI Lab
hzliyifu@corp.netease.com

Changjie Fan[‡]
NetEase Fuxi AI Lab
fanchangjie@corp.netease.com

Zhou Zhao
Zhejiang University
zhaozhou@zju.edu.cn

ABSTRACT

Game bots are automated programs that assist cheating users and enable them to obtain huge superiority, leading to an imbalance in the game ecosystem and the collapse of user interest. Therefore, game bot detection becomes particularly important and urgent. Among many kinds of online games, massively multiplayer online role playing games (MMORPGs), such as World of Warcraft and AION, provide immersive gaming experience and attract many loyal fans. At the same time, however, game bots in MMORPGs have proliferated in volume and method, evolving with the real-world detection methods and showing strong diversity, leaving MMORPG bot detection efforts extremely difficult. To deal with the fast-changing nature of game bots, we here proposed a generalized game bot detection framework for MMORPGs termed NGUARD, denoting NetEase Games' Guard. NGUARD takes charge of automatically differentiating game bots from humans for MMORPGs. In detail, NGUARD exploits a combination of supervised and unsupervised methods. Supervised models are utilized to detect game bots in observed patterns according to the training data. Meanwhile, unsupervised solutions are employed to detect clustered game bots and help discovering new bots. The game bot detection framework NGUARD has been implemented and deployed in multiple MMORPG productions in the NetEase Game portfolio, achieving remarkable performance improvement and acceleration compared to traditional methods. Moreover, the framework reveals outstanding robustness for game bots in mutated patterns and even in completely new patterns on account of the design of the auto-iteration mechanism.

[*]Equal contributions.
[†]NGUARD: NetEase Games' Guard, which is committed to detecting game bots and preserving the amusement order.
[‡]NetEase Fuxi AI Lab: named after Fu Xi, the legendary Creator in China, and established to enlighten games with artificial intelligence.

CCS CONCEPTS

• **Gaming systems → Game bot detection**; • **Information systems → Data mining**;

KEYWORDS

Game bot detection, time-interval event2vec, sequence autoencoder, bidirectional lstm, auto-iteration mechanism

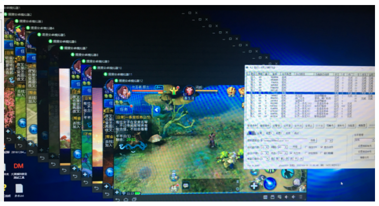
ACM Reference Format:
Jianrong Tao[*‡], Jiarong Xu[*], Linxia Gong[‡], Yifu Li[‡], Changjie Fan[‡], and Zhou Zhao. 2018. NGUARD: A Game Bot Detection Framework for NetEase MMORPGs[†]. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219925

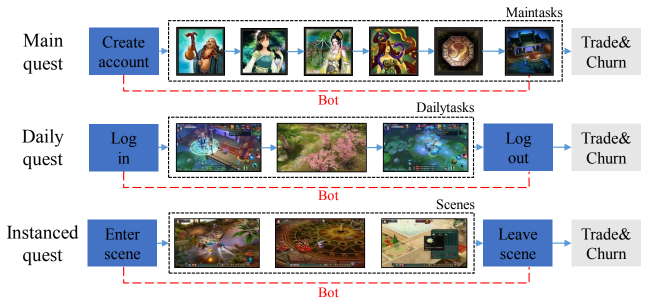
1 INTRODUCTION

As the popularity of online games increases, game players have a greater demand for getting a richer online game experience and a high-quality game entertainment. Since items and currency acquired virtually in games can be sold to other players for real profit in actual currency, illegal activities in online games have sharply increased and become more diverse [16, 28]. Many online game security providers have been victims of these actors. Thus, security has become an important issue in the online game market [11, 27]. Typically, MMORPGs are online games in which thousands of players use characters with specific roles to interact with each other and performs adventure-related tasks in the same continuous and persistent world [9]. MMORPGs are considered to be a profitable business by game bot developers. Hence, one major problem occurs: game bots that come along with almost every MMORPG.

Essentially, game bots are automated programs that reach the system kernel and perform continuously tough or tedious tasks without requiring the rest periods that human players require. Game bots use illegal methods to help users to obtain game advantages, e.g., accumulating more experience points, money or other items than human players. Figure 1 (b) shows an example of a game bot client, through which users can open multiple MMORPG clients simultaneously and leave the bot program to play the games itself, just as Figure 1 (a) indicates. Accordingly, game bot consumers can easily achieve great superiority over honest users, leading to the huge imbalance in the in-game ecosystem [17, 27]. The game industry has suffered serious threats from game bots, and game

(a) Game bot client


(b) MMORPG client



(c) Game bot in MMORPG

Figure 1: In-game screenshots of the game bot implemented

bots detection remains one of the most urgent problems that game publishers need to address.

Game bots are quite diversified, varying with different game conditions and spreading throughout the game universe. Labor- and time-consuming, immature operational technological means prove inadequate to handle the task of detecting these bots. Traditional machine-learning-based methods [1, 3, 14, 15, 23, 25] mainly exploit handcraft features and cannot integrally represent players' behavior in games, resulting in a general lack of performance. Meanwhile, these methods are depend greatly on labeled data and take a lot of time to iterate when a new type of game bot comes forward.

To address this problem, we propose a generalized game bot detection framework for MMORPGs: NGUARD, which represents natural player behavior and has excellent extensibility. NGUARD employs a combination of supervised and unsupervised methods for game bot detection, where supervised methods discriminate between bots and humans according to labeled data and unsupervised methods to detect clustered game bots and help discover new bots. Over the past years, NGUARD has been implemented and deployed in NetEase MMORPGs and received very positive reviews. The major contributions can be summarized as follows:

- We develop a generalized game bot detection framework for MMORPGs: NGUARD. We have created this detection framework by integrating supervised and unsupervised method, which produces accurate estimates. To achieve better and more effective performance, we also propose two different algorithms to initialize the models.
- We evaluate empirically on a very wide real-world dataset collected from NetEase MMORPGs. Through this we achieve remarkable performance improvements compared to traditional methods. Moreover, the framework performs outstanding robustness to game bots in mutated patterns and even in completely new patterns on account of the design of the auto-iteration mechanism.
- NGUARD developed for real game industry application achieves remarkable reviews in NetEase MMORPGs and can be extended to other areas of online games.

2 PRELIMINARIES

2.1 NetEase MMORPGs

Living up to the company's motto of "good games have no borders", NetEase Games has made great popularity and traction in domestic and overseas markets. In 2016, revenues of NetEase's online gaming segment reached $4 billion, up 61.6% year over year. NetEase Games continues to maintain high growth rate and deliver new hit games. To commit to the pursuit of the highest quality games and player experience, NetEase Games has developed and published dozens of popular games especially MMORPGs on pc and mobile, including Fantasy Westward Journey Online, New Westward Journey Online II, Ghost II, Tianxia 3, Fantasy Westward Journey Mobile, Ghost Mobile, New Westward Journey Mobile etc.

2.2 Game bots in NetEase MMORPGs

As one of China's largest MMORPG developer companies, NetEase Games has built up numerous game operation teams to deal with evolving game bots. Despite the great efforts that the game operation teams have made, MMORPG bot detection still remains a challenging task. Game bots in different NetEase MMORPGs have undermined fairness and playability in the game world and show similarity due to the consistent design of the game systems and gameplay methods. In order to work out a general framework for MMORPG bot detection, we went into the existing game bots and found out the three different types of quests that game bots aim to take on (see Figure 1(c)).

Main quests. The main quest usually consists of a series of main tasks that players need to pass through in order to unlock some other side quests. Human players normally complete all the main tasks and are always immersed in the game story before they can see the game endings. However, game bots can easily complete all the main tasks just to harvest game money after creating account. When game bots reach a specific level, they will transfer their accumulated money and churn .

Daily quests. The daily quest is a series of repetitive tasks that players can attend on a daily basis. The rewards for the daily quest are usually very limited due to their low difficulty and repetitiveness. However, those daily tasks will normally reward players with some collectibles that can be used to exchange for some other items. The goal for the daily quest is to stimulate players' incentive to login everyday, which in turn will increase the game's life span. In daily quests, game bots complete the daily tasks automatically and gain experience and daily money shortly after logging in.

Instanced quests. The instanced quest is a series of independent scenes in which a specific copy ("instance") is created for each individual party attempting to enter. As such, the game requires a small time period to create the quest, and it can only be used until a certain time limit before it is destroyed. Therefore, each party has the quest to themselves. Each player in the party has to know how to cooperate with others in order to finish the quest. Due to the high rewards of game experience and items, this type of quest is favored by many players. In instanced quests, game bots can kill monsters and collect money automatically after entering the instance.

3 DATASET DESCRIPTION

Our dataset is collected from a real-world MMORPG in NetEase, involving about 436 billion user logs from May 1st, 2016 to Dec 31st, 2017 which amount to 107TB. Among these users, 2.6 million players are game bots in main quests, 1.1 million players are game bots in daily quests and 0.2 million players are game bots in instanced quests. The game bots are identified and labeled by NetEase Games' operation teams. Considering users privacy, the players in our dataset are ensured by anonymizing all personal identifiable information.

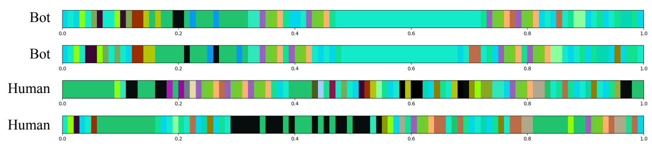
Each user log is composed of game events ordered by time stamp, which represents each player's behavioral sequence. And each game event contains four features as followed:

EventID: The event ID conducted by a game player, which describes the current event in detail. For example, a player uses a certain skill, obtains a certain item.

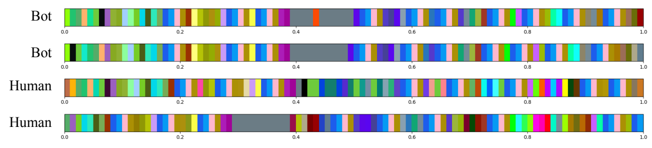
Interval: The time that has passed between the last and the current game event.

Count: The count of times that a certain game event happened during the current sampling time window, e.g., a player uses a certain game skill 10 times, then Count will be recorded as 10.

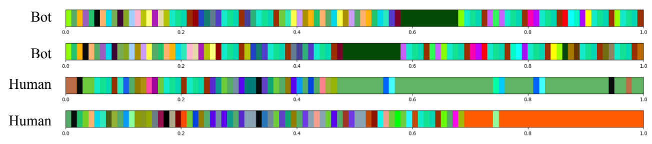
Level: The current game level for the player. The lowest level of each player is 1.



(a) Main quests



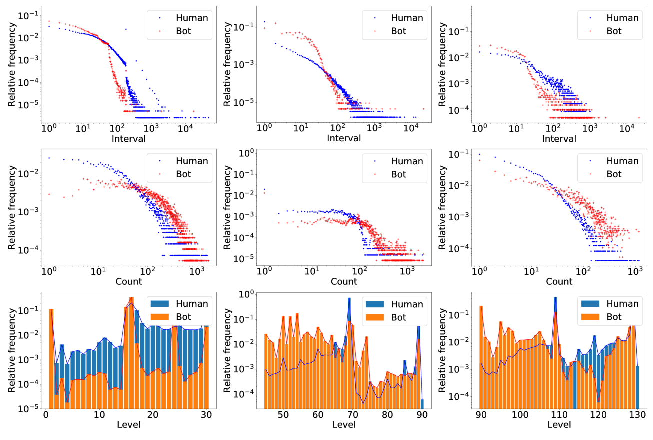
(b) Daily quests



(c) Instanced quests

Figure 2: Behavior sequence of game bots is similar to each other, while behavior sequence of human players shows diversity

Figure 2 visualizes the players' behavioral sequence(i.e., sequence of EventID) for three types of quests, which gives us a general idea of how a game bot's behavior sequence differs from a human player. Each slot represents a game event, and different game events are assigned different colors to differentiate them. As we can see from the figure, the behavior sequences of game bots exhibit a very different pattern compared to human ones. Human players' behavior sequences are more complicated and unpredictable than the game bots, which allow our models to correctly classify the players (as humans or bots).



(a) Main quest **(b) Daily quest** **(c) Instanced quest**

Figure 3: Relationship between features and their relative occurence frequency

To further illustrate, the relationship between features including Interval (in logs), Count (in logs) and Level of a certain EventID and their relative occurrence frequency (in logs) can imply important information for bot detection. As can be seen from the Figure 3, the features for humans and game bots demonstrate very different values. Due to limited space, we do not show the descriptive statistics related to all EventIDs, but these IDs do lead to the same considerations. In Figure Interval-Frequency, we find that the interval-frequency curve of game bots appears to be inclined to the left side of the chart, that is, the frequency of game bots is larger than that of human players where the Interval is relatively small, while the game bots is less frequency than that of human players where the Interval is relatively large. This can be easily explained by the fact that game bots are automatic programs, and that they only take a small time interval to finish a certain event. Similarly, in Figure Count-Frequency, we find that the count-frequency curve of game bots appears to be inclined to right side, that is, the frequency of game bots is larger than that of human players where the Count is relatively large, while the game bots is less frequency than that of human players where the Count is relatively small. Because game bots are not as flexible as human players, game bots need to consume more skills or items to finish a certain task. In Figure Level-Frequency, we find that the level peaks of game bots are different between different quests. In main quests, game bots creep up on some certain levels, because game bots always transfer game money when they have reached these certain levels. In daily quests and instanced quests, game bots are always crowed in a relatively low level range. The reason is that the goals of most game bots are earning game money, as the level goes up, the game quests become increasingly difficult. It is hard for game bots in high levels to make a profit. In conclusion, the above three features: Interval, Level and Count, will produce a good estimates for game bot detection.

4 FRAMEWORK

Our proposed MMORPG bot detection framework, termed NGUARD, is shown in Figure 4. NGUARD consists of a preprocessing module, an offline training module, an online inference module and an auto-iteration mechanism module. The cycle among offline training module, online inference module and auto-iteration module repeats itself on a periodic basis. The preprocessing module segments and samples the raw sequence(i.e., user log) to get high-quality sequence. The offline training module is responsible for training the models offline. The online inference module provides online service to bot detection. The auto-iteration mechanism module collects online data in every cycle phase to reconstruct the training set and perform short-term and long-term auto-iterations.

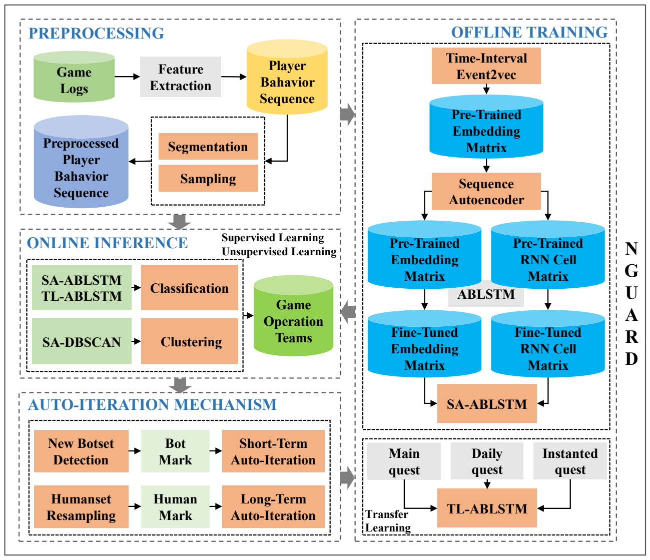


Figure 4: NGUARD: a MMORPG bot detection framework

In this section, we elaborate the above four major modules. The proposed supervised and unsupervised methods used in offline training module and online inference module are given.

4.1 PREPROCESSING

This section provides a flexible way to construct high-quality datasets on raw data by two steps: segmentation and sampling.

4.1.1 Segmentation. Due to the diversity of three types of quests, it is not possible to design and implement a one-size-fits-all segmentation method. To be compatible, the segmentation module is designed to be extensible, which includes three segmentation methods for each quest as follows.

For main quests, we segment each player's behavior sequence by Level. Since the main quest is designed to accompany the player throughout his game life cycle, in other words, the player's level will increase as the game goes on.

For daily quests, we segment each player's behavior sequence by date. Since daily quests repeat on a daily basis, a one-day sequence is enough to reflect the essential information.

For instanced quests, we take only the sequence fragment between on-instance action (enter scene) and off-instance action (leave scene) since this period contains most of the information in instanced quests.

4.1.2 Sampling. Different sampling policies are developed to obtain high-quality samples from the human and game bots dataset.

For data of human players, we describe our sampling method as follows. Let n be the desired number of sub-groups. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a segmentation set containing n sub-groups. Let the level range be $[l_{start}, l_{end}]$. p_k represents the k_{th} sub-group interval:

$$\begin{aligned} p_k =&(\lfloor l_{start} + (k-1)\frac{l_{end} - l_{start}}{n} + 0.5\rfloor, \\ &\lfloor l_{start} + k\frac{l_{end} - l_{start}}{n} + 0.5\rfloor] \end{aligned} \tag{1}$$

where $k = 1, 2, \ldots, n$, and $\lfloor \cdot \rfloor$ denotes round down, $(\cdot]$ denotes left closed right open interval. Then, for each sub-group we will sample approximately the same amount of data points. Hence, the sampled data contains diversified information from every level.

For data of game bots, we sample the data according to the density of the original set. First, we cluster the original data of game bots using SA-DBSCAN (Sequence Autoencoder and DBSCAN), which will be introduced in section § 4.3. Then, we will perform different sampling policies according to density for each cluster obtained. We considerably reduce the sampling frequency for clusters with a high density of data distribution and increase the sampling frequency for clusters with a low density of data distribution.

4.2 Offline Training

In this subsection, we introduce three phases of the offline training module: the pre-training phase, the modeling phase and the transfer-learning phase.

In the pre-training phase, we train Time-interval Event2vec to get the pre-trained embedding matrix of EventID. Then, We use pre-trained embedding matrix from the well-trained Time-interval Event2vec as the initial weights of the embedding layer for Sequence Autoencoder. Finally, we extract the fine-tuned pre-trained embedding matrix and rnn cell matrix from the well-trained Sequence Autoencoder as the initial weights of following classifier.

In the modeling phase, we propose a binary classifier SA-ABLSTM (Sequence Autoencoder and Attention-based Bidirectional LSTM) to distinguish bots from human players. We conduct unsupervised pre-training followed by supervised fine-tuning, i.e., the embedding layer and rnn cell of ABLSTM are initialized by the parameters provided by pre-trained Sequence Autoencoder. We also design an extra features layer concat to the attention layer to maximize the value of extracted features from behavioral sequences.

In the transfer learning phase, we perform transfer-learning by reusuing a pre-trained Sequence Autoencoder for a given (source) game quest on another (target) one.

The proposed models Time-interval Event2vec, Sequence Autoencoder, SA-ABLSTM and TL-ABLSTM will be introduced as follows.

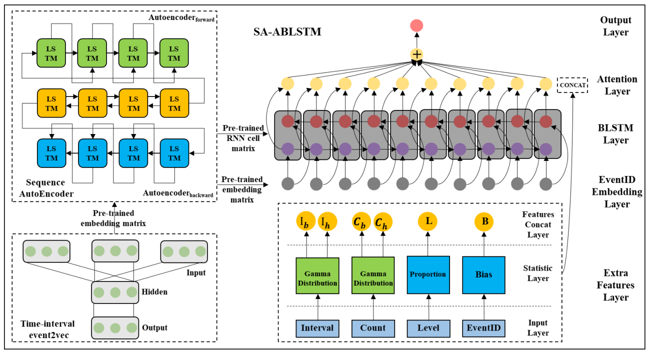


Figure 5: SA-ABLSTM architecture

4.2.1 *Time-interval event2vec.*

Inspired by the work of app2vec by Qiang Ma [19], we propose a model with similar approach called Time-interval Event2vec to deal with event vectorization. In the work of app2vec, the goal is to design a modified word2vec model which considers the weight between apps, where weight is measured by the time elapsed between two app sections. Similar to app2vec, our work of Time-interval Event2vec also considers the time elapsed between two events. Intuitively, the event within shorter time gaps to the target event should contribute more in predicting the target event.

We define the weight of each event e_i to target event e_t to be:

$$w(e_i, e_t) = \alpha^l \tag{2}$$

where α is chosen as 0.8 according to [19], and l is the amount of the time gap (e.g., number of seconds in our case) between event e_i within the current sequence and target event e_t.

Inserting the weight into CBOW model, then the weighted average of event vector $\overline{v}$ can be defined as:

$$\overline{v} = \frac{\sum_{-c \le j \le c, j \ne 0} w(e_j, e_t) v_j}{\sum_{-c \le j \le c, j \ne 0} w(e_j, e_t)} \tag{3}$$

where c is the context window, v_j is the vector representation of j_{th} event.

In our framework, we use the Time-interval Event2vec as a pre-training step to learn a vector representation for each EventID, and use the matrix of ids as the initial parameters of the word embedding matrix of the Sequence Autoencoder as mentioned below.

4.2.2 *Sequence Autoencoder.*

Our approach to Sequence Autoencoder is inspired by the work in semi-supervised sequence learning by Andrew M.Dai [8], which has been successfully used in many classification tasks. Key to this approach is to use seq2seq by Ilya Sutskever [24] as a pre-training step and then use its parameters as a starting point for other supervised training models.

A significant property of the Sequence Autoencoder is that it can be trained with large quantities of unlabeled data to improve its quality, which is especially useful for tasks that have limited labeled data [8].

We use the Sequence Autoencoder as an unsupervised learning model. The objective of the model is to reconstruct its own input sequence, and as such we can use the output of the encoder as the compressed vector representation of the input sequence. More concretely, we use a one-layer bidirectional RNN with LSTM cell in the encoder, and another in the decoder. Moreover, the word embedding matrix weights in Sequence Autoencoder are initialized by the pre-trained Time-interval Event2vec. We will fine tune the embedding layer during the training phase of Sequence Autoencoder.

4.2.3 *SA-ABLSTM.*

We pose the problem of identifying game bots as a binary classification problem. The model combining Sequence Autoencoder and Attention-based Bidirectional LSTM, namely SA-ABLSTM is proposed, i.e., the word embedding matrix weights and LSTM cell weights of the encoder of the Sequence Autoencoder are utilized as initialization weights in corresponding layers of ABLSTM. The structure of SA-ABLSTM is shown in Figure 5. The classification model ABLSTM is illustrated as follows.

Each user activity is assigned to an EventID, and one record of a user's continuous behavior sequence is represented as an EventID sequence $\{E_1, \ldots, E_n\}$. The model receives this sequence as input, followed by an embedding layer which transforms the input sequence to a sequence of vectors $\{e_1, \ldots, e_n\}$. We then feed the sequence $\{e_1, \ldots, e_n\}$ to a bidirectional lstm layer, which can better preserve the global information of the sequence. The bidirectional lstm layer outputs its hidden state vectors $H = \{h_1, \ldots, h_n\}$.

Additionally, we employ the information from the three extra features aforementioned: Interval, Count and Level. Differently from the EventIDs in the sequence, there is no temporal relations between the Interval (or Count, Level) values of the two events which appear in chronological order. Therefore, we apply knowledge-based methods to make use of these supplementary features:

- Based on our knowledge of game events, the occurrence of a certain event is a Poisson process. Thus the Interval of each type of event for a human and for a bot fits a gamma distribution respectively, i.e. $Interval_h \sim \Gamma(\alpha_h, \beta_h)$, $Interval_b \sim \Gamma(\alpha_b, \beta_b)$. The corresponding probability density function is

$$f(x; \alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)} \tag{4}$$

  For the i_{th} event in a sequence, we compute two probabilities:

$$I_{i,h} = f\left(t_i; \alpha_{i,h}, \beta_{i,h}\right), \; I_{i,b} = f\left(t_i; \alpha_{i,b}, \beta_{i,b}\right) \tag{5}$$

  These two probabilities help to efficiently differentiate humans and bots.
- Similarly, we can get another two probabilities from the information of *Count*:

$$C_{i,h} = f\left(c_i; \alpha_{c,h}, \beta_{c,h}\right), \; C_{i,b} = f\left(c_i; \alpha_{c,b}, \beta_{c,b}\right) \tag{6}$$

- In terms of the feature of *Level*, no typical distribution can be observed in general because of the game design. Therefore, we take the conditional probability that indicates the likelihood of bots at this given event e_i and Level l_i.

$$L_i = P(l_i) = \frac{N_{bot}^{l,e}}{N_{bot}^{l,e} + N_{human}^{l,e}} \tag{7}$$

- In the end, we introduce a bias which describes the occurrence probability of game bots for this kind of event.

$$B_i = P(e_i) = \frac{N_{bot}^{e}}{N_{bot}^{e} + N_{human}^{e}} \tag{8}$$

We obtain the extended feature vectors $H' = [h'_1, \ldots, h'_n]^T$ by concatenating the hidden vectors with the statistical values:

$$h'_i = h_i \circ I_{i,h} \circ I_{i,b} \circ C_{i,h} \circ C_{i,b} \circ L_i \circ B_i \tag{9}$$

Self-attention mechanism [26] is introduced to generate a sequence representation which places different importance contribution on the extended feature vectors:

$$a_i = softmax({w_i}^T H' + b_i) \text{ , } r_i = a_i \odot h'_i \tag{10}$$

where r_i is the representation vector at i_{th} time step after self-attention, computed by element-wise multiplying h'_i with the weight vector a_i. The representation of the whole sequence $R = [r_1, \ldots, r_n]$ is connected to a single neuron with sigmoid activation function to get a classification probability.

4.2.4 TL-ABLSTM. In this subsection, we will discuss the possibility of generalizing our models to different game quests, i.e., transfer learning.

Transfer learning aims to adapt knowledge between the related source and target domains [21]. Previous studies have proved the transferability of different tasks, and the transferring results outperform the random weights on different datasets of computer vision and natural language processing[2, 7, 20].

In our paper, we conduct Transfer Learning by reusing a Sequence Autoencoder for a given (source) game quest on another (target) one. For example, after we have pre-trained the Sequence Autoencoder of daily quest, what we can do is to use the parameters from the Sequence Autoencoder of the daily quests as the initial parameters for the training classification model ABLSTM of main quest. This solution skips the process of training Sequence Autoencoder of main quest, and it might not have the same performance as SA-ABLSTM of the main quests. However, compared to ABLSTM, TL-ABLSTM accelerates the training phase and has a better performance.

4.3 Online Inference

In the online inference module, we provide a supervised solution and an unsupervised solution to detect game bots.

4.3.1 Classification: SA-ABLSTM. In the supervised solution, the well-trained ABLSTM model from the offline training phase is deployed to detect game bots. When given an unknown user record, the model provides a probability as a reference to the operation team. The team decides whether to suspend the players after comparing the classification probability with a preset threshold, which is usually chosen according to operation experience by the team. A higher threshold leads to a higher precision, while a lower one to a higher recall.

4.3.2 Clustering: SA-DBSCAN. In the unsupervised solution, we propose a model combining Sequence Autoencoder and DBSCAN, namely SA-DBSCAN. DBSCAN is a data density-based clustering algorithm proposed by Martin Ester [10]. In our solution, we first extract the vector representation of EventID sequences from the well-trained Sequence Autoencoder. Next, perform DBSCAN on the vector representation of EventID sequence. Clusters of player groups with high behavioral similarity can be obtained by DBSCAN. Since human players holds the majority in online data, we can easily locate the clusters of human players. And the small clusters surrounding the clusters of human players are different types of game bots. The operation team takes action to deal with these potential game bots.

4.4 Auto-iteration Mechanism

The increasing online service and rapid renewal of game bots make a model outdated soon after the deployment. An auto-iteration mechanism is introduced in our framework to help in iterative modeling while minimizing human workload. The major steps of the auto-iteration mechanism are identified in this section.

4.4.1 Humanset Resampling. We pull down new records of human players from online servers and re-sample them with the sampling method proposed in section 4.1.2. Intuitively, our humanset resampling work helps our models adapt to real-time behavioral changes, since the game services and the playing methods are continuously changing as time goes by.

4.4.2 New Botset Detection. The clusters of game bots can be easily located through SA-DBSCAN. We distribute the game bots into three categories: known game bots that exist in training data; mutated game bots that derive from known bots but with varied features; unknown game bots that are not yet been detected.

For every cluster of game bots, we use ABLSTM model to predict the probability whether each data point is a game bot. Thus, an output probability distribution for the cluster is obtained, which can help us understand which category of game bots this cluster belongs to. Among the clusters of game bots, the ones with the high prediction probabilities are the known game bots, the ones with the middle probabilities are mutated bots, and the ones with the low probabilities are unknown bots.

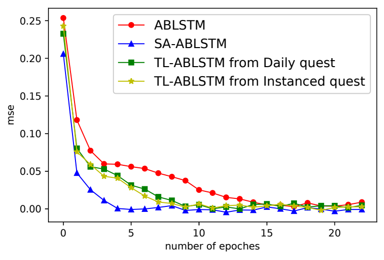
The mutated bots and unknown bots are hard to be detected by ABLSTM method, but easy to be located by clustering. We label these game bots and feed them into the process of iterating models, which is vital for our models to learn the changes in the online environment.

4.4.3 Short-term Auto-iteration. The training dataset is reconstructed based on the online data of human players, mutated game bots and unknown game bots extracted from humanset resampling and new botset detection.

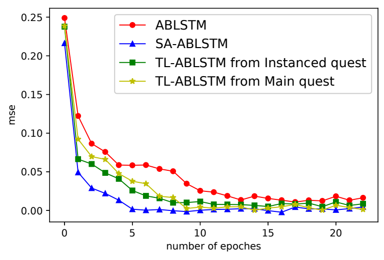
We replace the original negative dataset (human players records) with the dataset from the humanset resampling phase, and merge the original positive dataset (game bots data) with the new dataset

Table 1: Performance comparison of supervised methods

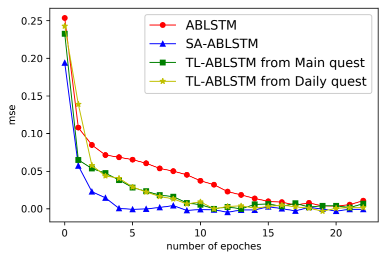
Main quest				Daily quest				Instanced quest			
Model	Precision	Recall	F1	Model	Precision	Recall	F1	Model	Precision	Recall	F1
MLP	0.9618	0.9773	0.9694	MLP	0.9528	0.9609	0.9568	MLP	0.9441	0.9571	0.9506
CNN	0.9721	0.9807	0.9764	CNN	0.9633	0.9712	0.9672	CNN	0.9552	0.9643	0.9597
Bi-LSTM	0.9809	0.9865	0.9837	Bi-LSTM	0.9709	0.9728	0.9718	Bi-LSTM	0.9612	0.9732	0.9672
ABLSTM	0.9851	0.9882	0.9866	ABLSTM	0.9716	0.9774	0.9745	ABLSTM	0.9674	0.9786	0.9730
TL-ABLSTM$_{im}$	0.9878	0.9896	0.9887	TL-ABLSTM$_{id}$	0.9736	0.9721	0.9728	TL-ABLSTM$_{di}$	0.9698	0.9801	0.9749
TL-ABLSTM$_{dm}$	0.9893	0.9906	0.9899	TL-ABLSTM$_{md}$	0.9771	0.9742	0.9756	TL-ABLSTM$_{mi}$	0.9704	0.9808	0.9756
SA-ABLSTM	**0.9904**	**0.9912**	**0.9908**	**SA-ABLSTM**	**0.9838**	**0.9861**	**0.9815**	**SA-ABLSTM**	**0.9721**	**0.9816**	**0.9768**



(a) Main quests

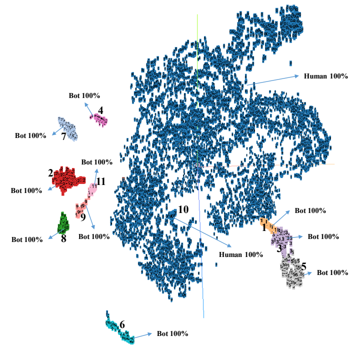


(b) Daily quests

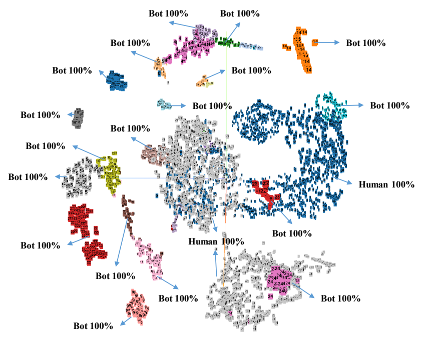


(c) Instanced quests

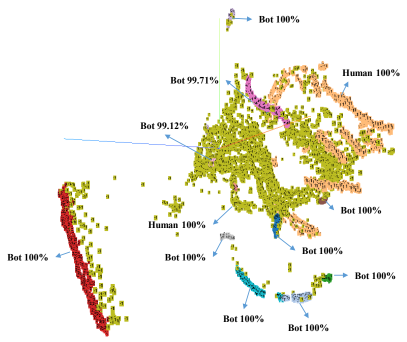
Figure 6: Convergence speed comparison of ABLSTM and its extensions



(a) Main quests



(b) Daily quests



(c) Instanced quests

Figure 7: t-SNE visualization of clustering result of SA-DBSCAN

from the new botset detection phase. The reconstructed training dataset enables our models to detect all types of game bots.

Short-term auto-iteration is performed on ABLSTM part. Since the training time of ABLSTM is relatively short, we update the ABLSTM model based on our reconstructed training dataset in the short-term cycles.

4.4.4 Long-term Auto-iteration. Long-term auto-iteration is performed on SA-ABLSTM. Since the training time of Time-interval word2vec and the Sequence Autoencoder is relatively long, we update these models based on the whole online dataset in the long-term cycles to reduce computation overheads. The updating process of the ABLSTM is the same as the short-term auto-iteration based on the reconstructed training dataset.

5 EVALUATION

In this section, we evaluate our game bot detection framework NGUARD on a real-world dataset collected from a NetEase MMORPG. We balance our dataset using under-sampling. Three sets of experimental studies are conducted:

(1) comparison of supervised methods' performance;
(2) evaluation of unsupervised approach SA-DBSCAN;
(3) verification of the necessity for the auto-iteration mechanism.

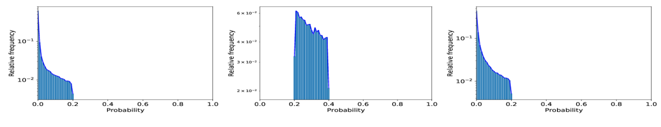
5.1 EXPERIMENTAL RESULTS

5.1.1 Comparisons of supervised solutions. We compare the performance of SA-ABLSTM, TL-ABLSTM, ABLSTM and three other models: (1) MLP model with 2 fully-connected layers, whose input is the frequencies of EventIDs; (2) CNN model with 1 convolution layers, followed by average pooling and 1 fully-connected layer; (3) Bi-LSTM model with 1 layer of Bi-LSTM, following by 1 fully-connected layer. Note that CNN and Bi-LSTM share the same input, i.e., the sequence of EventID concatenated with Interval, Count and Level.

Performance: The experiment results are shown in Table 1, with Precision, Recall and F1 as evaluation metrics: The ABLSTM model outperforms the MLP, CNN and Bi-LSTM models, while TL-ABLSTM[1] improves over ABLSTM a lot, and overall, SA-ABLSTM yields the best results.

Convergence speed: To further illustrate the convergence speed comparison of ABLSTM and its extensions, we plot the objective versus the number of epochs in Figure 6. The SA-ABLSTM model starts with the best initial parameters and converges fastest to the lowest loss value, which proves the advantage of pre-training steps. Also, comparing to the ABLSTM model, the two TL-ABLSTM models start with better initial parameters and converge faster to lower loss value, which demonstrates the feasibility of transfer learning.

5.1.2 Evaluations of unsupervised solution: SA-DBSCAN. We evaluate the effectiveness of our unsupervised clustering method SA-DBSCAN in bot detection. Figure 7 depicts the clustering result of SA-DBSCAN in 2-d t-SNE embedding plots. The clusters are assigned different colors to differentiate them. This visualization helps us quickly understand the similarity between players. We can find out the small-scale clusters from the result, which are game bots to a large extent. Game bots in a certain cluster are more similar to each other than to those in other clusters. The detection accuracy of each cluster of game bots is marked in the figure, which shows a good estimate of bots.


(a) Cluster "10": Human (b) Cluster "1": Bot (c) Cluster "6": Bot

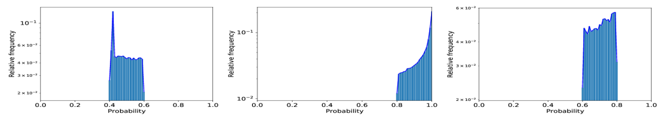
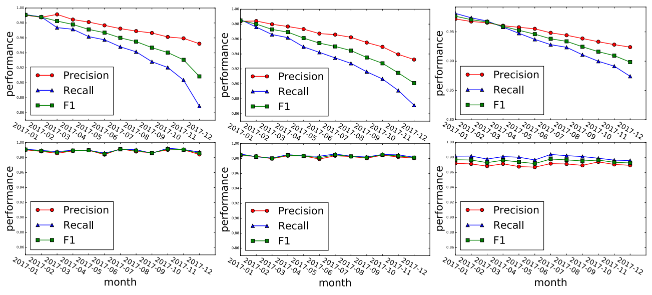

(d) Cluster "3": Bot (e) Cluster "5": Bot (f) Cluster "4": Bot

Figure 8: Clusters' probability distribution in Figure 7(a)

5.1.3 The necessity of the auto-iteration mechanism. The necessity is estimated from two aspects.

Firstly, we evaluate the ability of our unsupervised approach SA-DBSCAN to accurately detect mutated and new game bots. The clusters shown in Figure 7(a) are labeled as ground truth by the operation teams. According to the prediction probability distribution of each cluster shown in Figure 8, we can determine which type of game bots the cluster belongs to. Cluster "10" is the largest cluster with low probabilities, which is considered as human players. Small-scale clusters surrounding cluster "10" are all game bots. For example, cluster "6" with low probabilities is evaluated as unknown game bots, while cluster "5" with high probabilities as known game bots. Cluster "1", "3" and "5" with middle probabilities are evaluated as mutated game bots. The evaluation result remarkably coincides with our solution.

[1]TL-ABSLTM$_{ab}$ means 'a' quest transfers to 'b' quest, where 'm' is an abbreviation of main quest, 'd' is an abbreviation of daily quest, 'i' is an abbreviation of instanced quest.



(a) Main quests (b) Daily quests (c) Instanced quests

Figure 9: Performance with the auto-iteration mechanism

Secondly, we evaluate the performance of our framework with the auto-iteration mechanisms, as shown in Figure 9. In a real MMORPG production, we set the short-term cycle as 1 month and the long-term cycle as 3 months. Of all the quests, we observe the same results that the performance of our framework with auto-iteration mechanism merely decreases slightly during a short-term cycle and almost remains unchanged thanks to long-term cycle. Whereas, the performance of our framework without the auto-iteration mechanisms decreases rapidly over time.

6 RELATED WORK

In this section, we give a brief review of existing methods used to detect game bots.

The studies for game bot detection can be classified into three categories: client-side, network-side and server-side. Client-side game bot detection is signature-based, i.e., it monitors anomalies of the client machine and sends screen captures of the client to the game server. Yampolskiy et al. [29] proposed a protection mechanism for online games. Golle et al. [12] presented a special hardware device embedded CAPTCHA tests. However, game bots can easily detour this detection with disguise. In addition, client-side bot detection causes collisions in the operating system, which brings inconvenience to users. For these reasons, client-side game bot detection is not currently preferred.

Network-side detection detects different reactions of game bots and human players when there is a change in network traffic or network packets. Chen et al. [4] studied the traffic difference between official clients and standalone bot programs. Hilaire et al. [13] found that bots exhibit frequent packet arrivals patterns and send less information than human players. However, performing network-side bot detection can cause network overload and a lag in game time, a significant annoyance for the game experience.

Considering the drawbacks of client-side and server-side game bot detection methods, server-side game bot detection is the most needed detection method for game companies. This method takes game players' log data collected from game server and apply data mining techniques [6], since game bots display repeated and biased behavioral patterns differing from human players. Hence, server-side game bot detection does not cause any side-effects and facilitates game companies without deploying additional programs. In this paper, we adopt server-side game bot detection based on the game players' log, which produce high accuracy and efficiency by pre-defined detection algorithms.

Many works about game bot detection have proven the importance of sequence data for the behavioral analysis. Ahmad et al. [1] analyzed and calculated features from activity sequence to identify gold farmer. Platzer et al. [22] proposed a detection method using combat sequence produce by avatars. Chen et al. [5] is based on avatars movement trajectories sequence. Lee et al. [18] implemented the full action sequence of players on big data platform. Although these approaches considered the time-series information of sequence data, they only focus on feature dimension extracted from the sequence rather than actual time dimension. In our work, we directly model both feature dimension and time dimension, which can learn richer information from the sequence data. Moreover, our work also considers the temporal distance information between events. The temporal distance conveys information essential for bot detection, e.g., the time interval of game bot between two events is rather small.

Some other works employed supervised data mining methods to analyze user behaviors. Kim [15] used Decision Tree to detect bots by analyzing the window events. Kang [14] proposed multi-models such as Naive Bayes, Logistic Regression, Random Forest and Decision Tree to detect bots based on several unique and discriminative behavioral characteristics. Prasetya [23] examined ANN to detect game bots based on a similar pattern of bots. Bernardi [3] used MLP to detect bots based on playing behavior distributions. Although these works gained high accuracy on the training set, they cannot detect game bots in mutated patterns, not to mention completely new patterns as the online environment changes. Our paper solves this problem through unsupervised method and auto-iteration mechanism.

To the best of our knowledge, this is the first work that provides a generalized game bot detection framework for time series classification by integrating supervised and unsupervised models.

7 CONCLUSIONS

Game bots accumulate cyber assets and level up in a fast manner without sufficient effort, which has a severe adverse effect on human players. In this paper, we propose a generalized game bot detection framework for NetEase MMORPGs termed NGUARD. Considering the different behavioral patterns between human players and game bots, we employ a combination of supervised and unsupervised methods to detect game bots based on user behavior sequences. We apply an auto-iteration mechanism to automatically adapt to the mutated and new game bots. Extensive experiments have been performed on a real-world MMORPG dataset, which yield a significant performance gain comparing to traditional methods. What's more, NGUARD has been implemented and deployed in multiple MMORPG productions in NetEase Games and received very positive reviews.

Our promising future work is to deploy NGUARD in First Person Shooter (FPS) and Multiplayer Online Battle Arena (MOBA) productions in NetEase Games.

REFERENCES

[1] M. A. Ahmad, B. Keegan, J. Srivastava, D. Williams, and N. Contractor. 2009. Mining for Gold Farmers: Automatic Detection of Deviant Players in MMOGs. In *2009 International Conference on Computational Science and Engineering*, Vol. 4. 340–345. DOI: http://dx.doi.org/10.1109/CSE.2009.307

[2] Amr Ahmed, Kai Yu, Wei Xu, Yihong Gong, and Eric Xing. 2008. Training Hierarchical Feed-Forward Visual Recognition Models Using Transfer Learning from Pseudo-Tasks. In *Computer Vision – ECCV 2008*, David Forsyth, Philip Torr, and Andrew Zisserman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 69–82.

[3] Mario Luca Bernardi, Marta Cimitile, Fabio Martinelli, and Francesco Mercaldo. 2017. A Time Series Classification Approach to Game Bot Detection. In *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics (WIMS '17)*. ACM, New York, NY, USA, Article 6, 11 pages.

[4] Kuan-Ta Chen, Jhih-Wei Jiang, Polly Huang, Hao-Hua Chu, Chin-Laung Lei, and Wen-Chin Chen. 2009. Identifying MMORPG Bots: A Traffic Analysis Approach. *EURASIP J. Adv. Signal Process* 2009, Article 3 (Jan. 2009), 22 pages.

[5] Kuan-Ta Chen, Hsing-Kuo Kenneth Pao, and Hong-Chung Chang. 2008. Game Bot Identification Based on Manifold Learning. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '08)*. ACM, New York, NY, USA, 21–26.

[6] YeonJun Choi, SungJune Chang, YongJun Kim, HunJoo Lee, WookHo Son, and SeongIl Jin. 2016. Detecting and monitoring game bots based on large-scale user-behavior log data analysis in multiplayer online games. *The Journal of Supercomputing* 72, 9 (01 Sep 2016), 3572–3587.

[7] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2493–2537.

[8] Andrew M Dai and Quoc V Le. 2015. Semi-supervised Sequence Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3079–3087.

[9] Nicolas Ducheneaut and Robert J. Moore. 2004. The Social Side of Gaming: A Study of Interaction Patterns in a Massively Multiplayer Online Game. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW '04)*. ACM, New York, NY, USA, 360–369. DOI: http://dx.doi.org/10.1145/1031607.1031667

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231.

[11] Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang. 2009. Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*. ACM, New York, NY, USA, 256–268. DOI: http://dx.doi.org/10.1145/1653662.1653694

[12] Philippe Golle and Nicolas Ducheneaut. 2005. Preventing Bots from Playing Online Games. *Comput. Entertain.* 3, 3 (July 2005), 3–3.

[13] S. Hilaire, H. c. Kim, and C. k. Kim. 2a010. How to deal with bot scum in MMORPGs? 1–6. DOI: http://dx.doi.org/10.1109/CQR.2010.5619911

[14] Ah Reum Kang, Seong Hoon Jeong, Aziz Mohaisen, and Huy Kang Kim. 2016. Multimodal game bot detection using user behavioral characteristics. *SpringerPlus* 5, 1 (26 Apr 2016), 523.

[15] Hyungil Kim, Sungwoo Hong, and Juntae Kim. 2005. Detection of Auto Programs for MMORPGs. In *AI 2005: Advances in Artificial Intelligence*, Shichao Zhang and Ray Jarvis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1281–1284.

[16] H. Kwon, A. Mohaisen, J. Woo, Y. Kim, E. Lee, and H. K. Kim. 2017. Crime Scene Reconstruction: Online Gold Farming Network Analysis. *IEEE Transactions on Information Forensics and Security* 12, 3 (March 2017), 544–556. DOI: http://dx.doi.org/10.1109/TIFS.2016.2623586

[17] Eunjo Lee, Jiyoung Woo, Hyoungshick Kim, Aziz Mohaisen, and Huy Kang Kim. 2016. You are a Game Bot!: Uncovering Game Bots in MMORPGs via Self-similarity in the Wild. In *NDSS*.

[18] Jina Lee, Jiyoun Lim, Wonjun Cho, and Huy Kang Kim. 2015. In-Game Action Sequence Analysis for Game BOT Detection on the Big Data Analysis Platform. In *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary*

Systems - Volume 2, Hisashi Handa, Hisao Ishibuchi, Yew-Soon Ong, and Kay-Chen Tan (Eds.). Springer International Publishing, Cham, 403–414.

[19] Q. Ma, S. Muthukrishnan, and W. Simpson. 2016. App2Vec: Vector modeling of mobile apps and applications. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 599–606.

[20] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*. IEEE Computer Society, Washington, DC, USA, 1717–1724.

[21] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (Oct 2010), 1345–1359.

[22] C. Platzer. 2011. Sequence-based bot detection in massive multiplayer online games. In *2011 8th International Conference on Information, Communications Signal Processing*. 1–5.

[23] Kusno Prasetya and Zheng da Wu. 2010. Artificial Neural Network for Bot Detection System in MMOGs. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games (NetGames '10)*. IEEE Press, Piscataway, NJ, USA, Article 16, 2 pages.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3104–3112.

[25] Ruck Thawonmas, Yoshitaka Kashifuji, and Kuan-Ta Chen. 2008. Detection of MMORPG Bots Based on Behavior Analysis. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology (ACE '08)*. ACM, New York, NY, USA, 91–94.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 6000–6010.

[27] Jiyoung Woo and Huy Kang Kim. 2012. Survey and Research Direction on Online Game Security. In *Proceedings of the Workshop at SIGGRAPH Asia (WASA '12)*. ACM, New York, NY, USA, 19–25.

[28] Kyungmoon Woo, Hyukmin Kwon, Hyun chul Kim, Chong kwon Kim, and Huy Kang Kim. 2011. What can free money tell us on the virtual black market. *ACM SIGCOMM* (2011).

[29] Roman V. Yampolskiy and Venu Govindaraju. 2008. Embedded Noninteractive Continuous Bot Detection. *Comput. Entertain.* 5, 4, Article 7 (March 2008), 11 pages.