NGUARD+: An Attention-based Game Bot Detection Framework via Player Behavior Sequences

JIARONG XU and YIFAN LUO, Zhejiang University
JIANRONG TAO and CHANGJIE FAN, NetEase Fuxi AI Lab
ZHOU ZHAO and JIANGANG LU, Zhejiang University

Game bots are automated programs that assist cheating users, leading to an imbalance in the game ecosystem and the collapse of user interest. Online games provide immersive gaming experience and attract many loyal fans. However, game bots have proliferated in volume and method, evolving with the real-world detection methods and showing strong diversity, leaving game bot detection efforts extremely difficult. Existing game bot detection techniques mostly rely on handcrafted features or time-series based features instead of fully utilizing player behavior sequences. In this regard, a more reasonable way should be learning user patterns from player behavior sequences when facing the fast-changing nature of game bots. Here we propose a general game bot detection framework for massively multiplayer online role playing games termed NGUARD+ (denoting NetEase Games' Guard), which captures user patterns in order to identify game bots from player behavior sequences. NGUARD+ mainly employs attention-based methods to automatically differentiate game bots from humans. We provide a combination of supervised and unsupervised methods for game bot detection to detect game bots and new type of game bots even when the labels of game bots are limited. Specifically, we propose the following two variants for attention-based sequence modeling: Attention based Bidirectional Long Short-Term Memory Networks (ABLSTM) and Hierarchical Self-Attention Network (HSAN) as our supervised models. ABLSTM is keen on inducing certain inductive biases which makes learning more reasonable as well as capturing local dependency and global information, while HSAN could handle much longer behavior sequences with less memory and higher computational efficiency. Experiments conducted on a real-world dataset show that NGUARD+ can achieve remarkable performance improvement compared to traditional methods. Moreover, NGUARD+ can reveal outstanding robustness for game bots in mutated patterns and even in completely unseen patterns.

CCS Concepts: • **Information systems** → **Data mining;** • **Computer systems organization** → *Embedded systems*;

Additional Key Words and Phrases: Game bot detection, sequence modeling, transfer learning, auto-iteration mechanism, attention mechanism

This research was supported by the National Natural Science Foundation of China (Grant No. 61836002, 61751209, and U1611461), Zhejiang Natural Science Foundation (Grant No. LR19F020006), National Key R&D Program of China (Grant No. 2018AAA0100603), and Fundamental Research Funds for the Central Universities (Zhejiang University NGICS Platform).
Authors' addresses: J. Xu, Y. Luo, Z. Zhao, and J. Lu, Zhejiang University, 38 Zheda Road, Hangzhou, Zhejiang, P.R. China, 310027; emails: {xujr, wilsonlaw, zhaozhou, lujg}@zju.edu.cn; J. Tao and C. Fan, NetEase Fuxi AI Lab, 599 Wangshang Road, Hangzhou, Zhejiang, P.R. China, 310052; emails: {hztaojianrong, fanchangjie}@corp.netease.com.

1556-4681/2020/09-ART65 $15.00
https://doi.org/10.1145/3399711

ACM Reference format:
Jiarong Xu, Yifan Luo, Jianrong Tao, Changjie Fan, Zhou Zhao, and Jiangang Lu. 2020. NGUARD+: An Attention-based Game Bot Detection Framework via Player Behavior Sequences. *ACM Trans. Knowl. Discov. Data* 14, 6, Article 65 (September 2020), 24 pages.
https://doi.org/10.1145/3399711

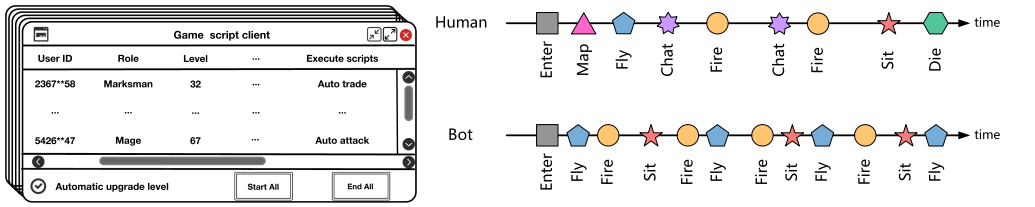
1 INTRODUCTION

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 [24, 43]. Many online game providers have been victims of game bots.[1] Estimation by BattlEye[2] showed that there were over 1,044,000 PUBG[3] bots banned in January, 2018, which may exceed 1 year sales of many games. In addition, PUBG bots directly caused average active users declining by almost 2.7% by the end of February 9, 2018. Among many kinds of games, game bots come along with almost every Massively Multi-player Online Role Playing Game (MMORPG).[4] Figure 1(a) 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 automatically. Accordingly, game bot consumers can easily achieve great superiority over hones and well-behaved users [26, 39, 42]. Thus, game bot detection is one of the most urgent concerns that game publishers need to address.

The most common way to deal with the problem is to adopt machine learning-based methods, which heavily rely on hand-crafted features or time-series-based features. These methods extract delicate features such as player information [1], social interaction diversity [19, 21], action frequency [4, 32], trace diversity [6], and the like. However, several challenges still remain. First, constructing features is labor-intensive and time-consuming. Moreover, we cannot perform direct optimization on the original user behavior sequences. The second challenge is how to deal with long sequence data; that is, to capture the local and global dependency with less memory and time cost. Third, the labels of game bots are expensive to collect while large amount of unlabeled data are easier and cheaper to be available, so how to train our model with limited labeled data or even without labeled data is a challenge. Fourth, another difficult problem is that concept drift may be present on real-world game bot detection where the patterns of game bots will change over time. This kind of mutated game bots will make the predictions lack robustness.

To deal with the first challenges, we develop a new paradigm for refining game bot detection through the analysis of user behavior sequences in order to discover the different patterns between humans and bots. There are three primitive observations for identifying abnormal patterns through user behavior sequences. We explain these by taking an example of MMORPGs in Netease Games[5] in Figure 1(b), where the behavior sequence of a human and the behavior sequence of a bot are given. First, we can observe that the time interval between behaviors in the bot's sequence is much smaller than the human's. This indicates the following common property of bots: they do not need to take much time to response. Second, we find that the bot always conducts "Fly", "Fire",

[1] Automated programs that reach the system kernel and perform continuously tough or tedious tasks without requiring the rest periods.
[2] An anti-cheat company, which provides anti-cheat system for PUBG.
[3] PlayerUnknown's Battlegrounds, an online multiplayer battle royale game.
[4] 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.
[5] A company for game productions,



(a) Game bot client. (b) An illustration of behavior sequence between human and bot.

Fig. 1. In-game screenshots of the game bot implemented.

"Sit" in a cycle mode, which reveals a certain rhythm in its behavior sequence. Third, the behaviors of human are more diversified, e.g., human players tend to check the "map" (game scene) when they first enter the game environment. Moreover, human players tend to prefer social activities such as "chat" during a game. These aforementioned observations show the significance of exploiting behavior sequences, which can fully utilize the user behavior sequences to model human or bot patterns in a more general and direct way.

When it comes to the second challenge, capturing the context dependency within sequential data is crucial for sequence modeling [13]. Recurrent neural networks [11, 20, 33, 41], convolutional neural networks [25], and attention mechanism [3, 28, 40] have been widely applied to deal with context dependency. Here, we propose two variants of attention-based sequence modeling: Attention based Bidirectional LSTM (ABLSTM) and Hierarchical Self-attention Network (HSAN). ABLSTM, the combination of bidirectional LSTM and attention mechanism, is keen on inducing certain inductive biases which makes learning more reasonable, especially when the input sequences are not very long. HSAN is proposed to alleviate the large memory requirement in existing self-attention networks (i.e., the memory grows quadratically in line with the sequence length when storing the alignment scores). To be specific, we split the input sequence into several equal-length subsequences, which can be viewed as a hierarchical way, where the intra-level self-attention networks (SAN) captures the local dependency within each subsequence and the inter-level SAN captures the global dependency between subsequence. HSAN only needs to conduct self-attention mechanism on a short subsequence during intra-level SAN, which saves a lot memory compared to a whole long sequence. Thus, HSAN requires less time and space cost, and also takes the advantage of parallelizable computation and is able to capture the long and local dependency. The ABLSTM and HSAN variant exhibit different advantages when modeling user behavior sequences in different cases.

To deal with the third and fourth challenge, we employ a combination of supervised and unsupervised methods for game bot detection, where supervised methods are applied to discriminating bots and humans according to labeled data, and unsupervised methods are also applied to detecting game bots and can further help discover new type of game bots. In addition, we develop an auto-iteration mechanism to cope with changeable bot environment.

In this work, we propose a general game bot framework for MMORPGs, namely, *NGUARD+*. Over the past years, NGUARD+ has been implemented and deployed in NetEase MMORPGs, which avoids 50% unnecessary economic losses caused by game bots and helps improve 80% work efficiency for operations teams. The major contributions can be summarized as follows:

—We explore the different patterns between humans and bots and develop a new paradigm for game bot detection through the empirical study of user behavior sequences directly and efficiently.

— We propose two variants for sequence modeling: an ABLSTM and a HSAN. Specially, HSAN can be applied for modeling context dependency in sequence data, which requires less memory but inherits all the merits of SAN.
— We integrate supervised and unsupervised method to alleviate the problem of lacking labeled data, and further discover new bots never encountered to enhance detection robustness.
— We develop a generalized game bot detection framework for MMORPGs: NGUARD+. We evaluate empirically on a real-world dataset collected from NetEase MMORPGs, and achieve competitive 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.

The remainder of the article is organized as follows, Section 2 reviews the related works and Section 3 briefly describes the backgrounds of Netease MMORPGs. Section 4 introduces our dataset and performs a deep analysis on it. Section 5 discusses our detailed framework and proposed model. Experiments are conducted in Section 6 and Section 7 concludes the article.

2 RELATED WORK

In this section, we will describe and review some state-of-the-art of game bot detection methods, especially from the aspects of user modeling.

The studies for game bot detection can be classified into the following three categories: client-side, network-side and server-side. Client-side game bot detection has been put into application in most game companies. This kind of method is signature-based which monitors abnormality of a client machine and sends a screen capture of the client to the game server. Yampolskiy et al. [44] proposed a protection mechanism for online games. Golle et al. [15] presented a special hardware device embedded CAPTCHA tests. However, game bots can easily detour this detection method with reverse engineering. In addition, client-side bot detection causes collisions in operating system, which brings inconvenience for users. For these reasons, client-side game bot detection is not currently preferred.

Network-side game bot detection is designed to observe network traffic or network packets. This kind of method detects different reactions of game bots and human players when there is a change in network conditions. Chen et al. [5] studied the traffic difference between official clients and standalone bot programs. Hilaire et al. [17] 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 network-side game bot detection methods, server-side game bot detection is the most needed detection method for game companies. Server-side game bot detection uses game players' log data collected from game server. Game bots display repeated and biased behavioral patterns differing from human players. This kind of method is based on data mining techniques. First, extract the feature set from log data [7]. Then, perform classification using the feature set. Hence, server-side game bot detection does not cause any side-effects and facilitates game companies without deploying additional programs. In this article, we adopt server-side game bot detection based on the game players' log, which produce high accuracy and efficiency by pre-defined detection algorithms.

Traditional methods on bot detection always relied on handcrafted feature design [1, 4, 21, 22, 27, 32, 32, 34]. These studies took a lot of effort during the process of feature representation, feature exploration and feature selection. Though feature extraction can discover the most representative information, it is inefficient and inflexible to perform.

Sequence data have recently drawn significant attention, and learning from sequence data endows the bot detection system with more potential performance. Ahmad et al. [1] analyzed player activity sequence and some other features to discriminate gold farmers and legitimate platers. Platzer et al. [32] proposed a detection method using combat sequences produced by avatars and calculated the similarity between combat sequences. Chen et al. [6] is based on avatar's movement trajectory sequences in game and use manifold learning to distinguish the trajectories of human players and game bots. Lee et al. [27] implemented the full action sequences of players on a big data platform. Bernardi et al. [4] took into account the history of the players' actions and performed a time series classification. Although the above approaches took into account the temporal information of sequence data, they still need to extract features from sequence data for next modeling. In other words, the previous studies only focus on the feature dimension extracted from sequence rather than the complete time dimension. However, the information flow from two viewpoints, i.e., horizontal axis of time and vertical axis of feature has been demonstrated to be very effective on modeling users' characteristic. Moreover, few of them allows end-to-end gradient-based training with original user behavior sequences. Our work directly accepts the original user behavior sequences as inputs, considering the long-term behavior dependency and the time interval between behavior events, which also allows us to learn in an end-to-end architecture.

Almost all the previous works employed supervised data mining methods to analyze user behaviors. Kim et al. [22] used Decision Tree to detect bots by analyzing the window events. Kang et al. [21] 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 et al. [34] examined ANN to detect game bots based on a similar pattern of bots. Bernardi et al. [4] used MLP neural network to detect bots based on playing behavior distributions. Lee et al. [26] introduced the automated maintenance process to manage the detection performance. 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 article solves this problem through unsupervised method and auto-iteration mechanism.

To the best of our knowledge, the current research is the first to provide a generalized game bot detection framework. It is also worth to mention that our work integrates supervised and unsupervised models based on time series classification and clustering.

Besides, there is another line of studies that is related with our model. First, attention-based methods [3, 28, 40] have been widely applied to dealing with context dependency. Recently, SAN is successful in multiple Natural Language Processing (NLP) tasks, which is context-aware by applying attention to capture the dependency in input sequence. Among multiple works of SAN, multi-head attention proposed by [40] use a seq2seq-based structure to project the input sequence into multiple subspaces, where SAN are utilized within each subspace. [37] proposed a directional self-attention network, which encodes temporal order information by computing the alignment score matrix at feature level. These works all achieve the state-of-the-art performance in NLP tasks. However, the large memory requirement in HSAN makes it hard to utilize especially when the sequence is very long. Second, transfer learning is a useful tool to improve the performance of target learners from one domain by transferring knowledge from a related domain [2, 31]. Here, our framework also applies the transfer learning to improve the performance of the target quest from a related quest.

3 PRELIMINARIES

3.1 NetEase MMORPGs

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,

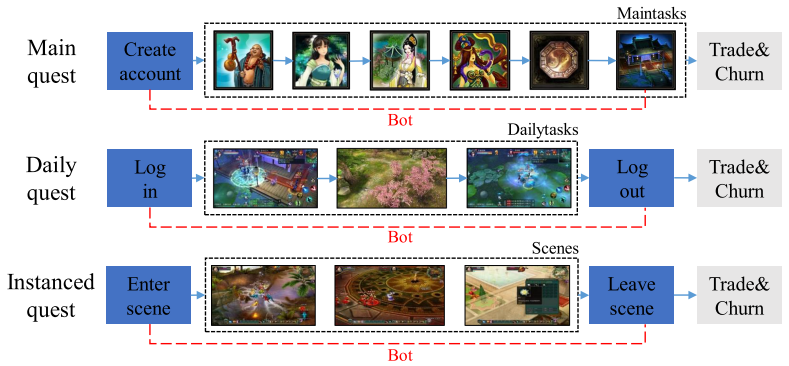


Fig. 2. Game bot in MMORPG.

including Fantasy Westward Journey Online, New Westward Journey Online II, Ghost II, Tianxia 3, and the like. In Netease MMORPGs, a player operates multiple characters, where the character is an avatar with unique features such as appearance, name, level, class, and gender.

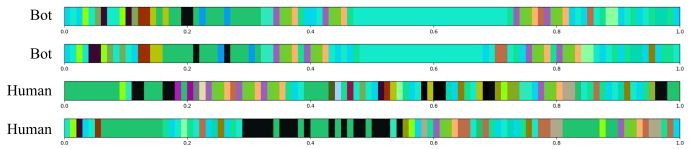
3.2 Game Bots in NetEase MMORPGs

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 2).

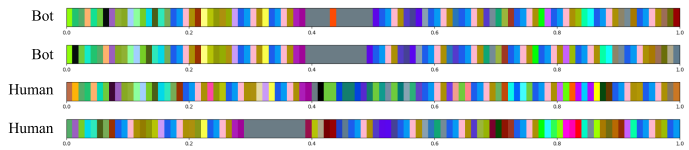
Main quests. The main quest usually consists of a series of main tasks that players need to pass through in order to unlock some other side quests. Human players normally complete all the main tasks and are always immersed in the game story before they can see the game endings. However, game bots can 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.

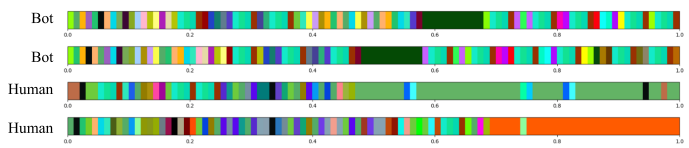
Instanced quests. The instanced quest is a series of independent scenes in which a specific copy ("instance") is created for each individual party attempting to enter. As such, the game requires a small time period to create the quest, and it can only be used until a certain time limit before it is destroyed. 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.



(a) Main quests



(b) Daily quests



(c) Instanced quests

Fig. 3. Behavior sequences of game bots is similar to each other, while behavior sequences of human players shows diversity.

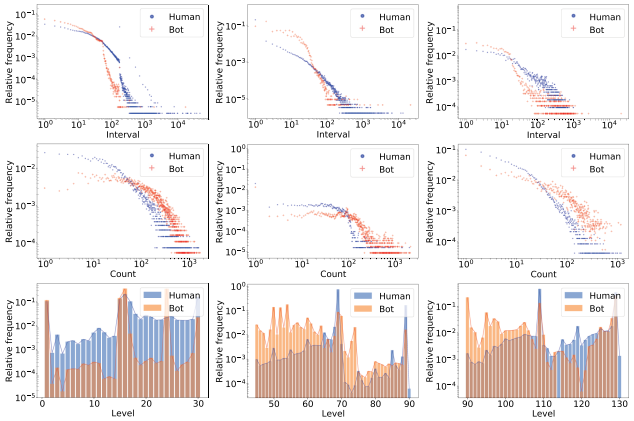
4 DATASET DESCRIPTION

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 107 TB. 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 the count will be recorded as 10.
- **Level**: The current game level for the player. The lowest level of each player is 1.

Figure 3 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 behavioral signatures compared to human ones. Human players' behavior sequences are more complicated and unpredictable than the game bots in most cases (game bots seldom do random actions in order to maximize profit), which allow our models to correctly classify the players (as humans or bots).



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

Fig. 4. Relationship between features and their relative occurrence frequency.

To further illustrate, the relationship between features including the interval (in logs), the count (in logs), and the 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 4, 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 frequency of game bots is smaller than that of human players where the count is relatively small. The reason for this is that game bots are not as flexible as human players, so 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, so it is hard for game bots to make a profit in high levels. Thus, the above three features: the interval, the level, and the count, will produce good estimates for game bot detection.

5 FRAMEWORK

Our proposed MMORPG bot detection framework, NGUARD+, is shown in Figure 5. It consists of a preprocessing module, an offline training module, an online inference module, and an auto-iteration mechanism module. In the preprocessing module, we segment and sample the raw sequences (i.e., user logs) to get high-quality sequences. After training our models offline, the online inference module provides online service to our bot detection system. Finally, we conduct short-term and long-term auto-iterations in order to cope with changeable bot environment, where the

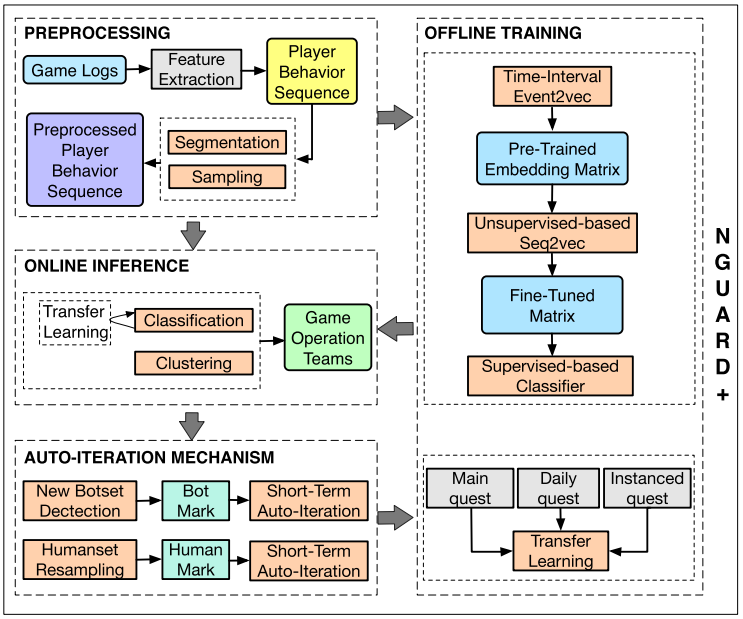


Fig. 5. NGUARD+: a MMORPG bot detection framework, which contains four modules: preprocessing, offline training, online inference, and model iteration.

auto-iteration mechanism can help us collect online data in every cycle phase (i.e., offline training -> online training -> auto-iteration mechanism) and update the training set and our models.

5.1 Preprocessing

This section provides a flexible way to construct high-quality datasets based on raw data. It contains two steps. First, we extract the core segmentations in player behavior sequences for each quest, and then we design careful sampling strategies to construct the human and game bot training set.

5.1.1 Segmentation. Considering the diversity between different types of quests, it's hard to implement a one-size-fits-all segmentation method for all quests. Here, we design three segmentation methods for each quest as follows:

***Main quest segmentation*:** Since the main quest is the quest to be finished throughout a player's whole game life cycle, the player's level will increase as main quest goes on. We therefore segment each player behavior sequence according to player's level.

***Daily quest segmentation*:** The daily quest can be finished by players on a daily basis, which can also be taken as some daily tasks to gain some rewards. We segment each player's behavior sequence by each day, because a one-day sequence is enough to reflect the essential information.

***Instanced quest segmentation*:** For instanced quest, we only take the sequence between on-instance action (enter scene) and off-instance action (leave scene) since this period contains most of the information in this kind of quest.

5.1.2 Sampling. Different sampling strategies are developed to obtain high-quality samples for the human and game bots set.

For data of human players, we group them into S sub-groups, then the human player set P can be segmented into $P = \{p_1, p_2, \ldots, p_S\}$. We define the s-th sub-group interval p_s as:

$$p_s = \left(\left\lfloor l_{\text{start}} + (s-1)\frac{l_{\text{end}} - l_{\text{start}}}{S} + 0.5 \right\rfloor, \left\lfloor l_{\text{start}} + s\frac{l_{\text{end}} - l_{\text{start}}}{S} + 0.5 \right\rfloor \right) \tag{1}$$

where $s = 1, 2, \ldots, S$, l_{start} is the lowest level among all players, l_{end} is the highest level among all players, $\lfloor \cdot \rfloor$ denotes round down, and $(\cdot]$ denotes left closed right open interval. Since we want to make the sampled data contain diversified information from each player's level, we sample approximately the same amount of data points from each sub-group.

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 a proposed Unsupervised-based Seq2vec and DBSCAN (see Section 5.3.2). Then, we will perform different sampling policies according to the 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.

5.2 Offline Training

Three phases of the offline training module (the pre-training phase, the modeling phase, and the transfer-learning phase) will be sequentially conducted during offline training. In the pre-training phase, we propose two models under an unsupervised fashion to utilize the huge amount of unlabeled data. First, the time-interval Event2vec is proposed to learn the EventID embedding matrix as the preliminary step. Then, the unsupervised-based Seq2vec is proposed to further learn a fine-tuned EventID embedding matrix, given the EventID embedding matrix from time-interval Event2vec as initial weights of its embedding layer. In the modeling phase, we propose two variants of supervised-based classifier to distinguish bots from human players, where the embedding layer of the classifiers can be initialized by the fine-tuned EventID embedding matrix as mentioned above. In the transfer learning phase, we perform transfer learning by reusuing a pre-trained unsupervised-based Seq2vec for a given (source) game quest on another (target) one.

In this section, we will first present two variants of supervised-based classifiers in then modeling phase, and then we will introduce time-interval Event2vec and unsupervised-based Seq2vec used on the pre-training phase. Finally, the transfer learning mechanism will be given.

5.2.1 Supervised-based Classifier. The supervised-based classifier is the main model in conducting modeling phase, which consists of the following three parts from its bottom to top: (1) sequence-level modeling; (2) knowledge-level modeling; and (3) prediction-level modeling. Here, we propose two variants of supervised-based classifier: ABLSTM and HSAN, both of which implement the knowledge-level modeling but differ in the implementation of sequence-level modeling and prediction-level modeling. Specifically, ABLSTM is keen on inducing certain inductive biases which makes learning more reasonable; while HSAN provides a more efficient way to model the local and long dependency in user behavior sequences with less time and space cost.

In initial steps, we first assign each user activity an EventID. Then, one record of a user's continuous behavior sequence is represented as an EventID sequence $\{E_1, \ldots, E_N\}$, which will serve

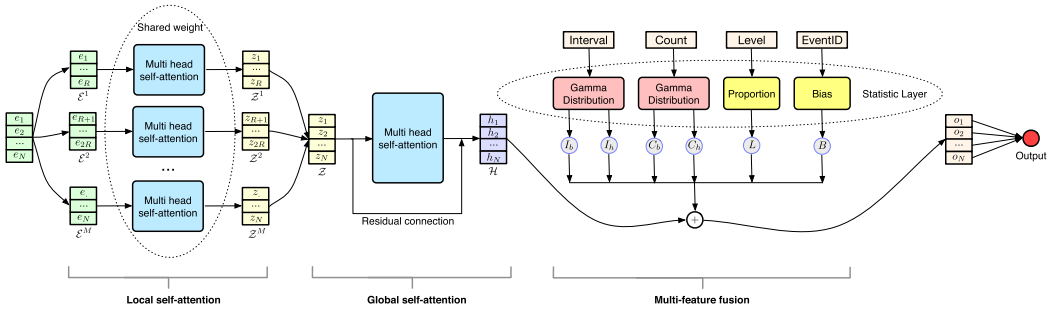


Fig. 6. HSAN architecture.

as the input of our models. Through an embedding layer, the EventID sequence will be mapped into a sequence of latent EventID vectors $\{e_1, e_2, \ldots, e_N\} \in \mathbb{R}^{N \times d_{\text{event}}}$.

(1) Sequence-level modeling: The sequence-level modeling aims to explicitly model the local and global dependency of activities in player's behavior sequence. Here, we propose two variants as follows:

***Variant I*: ABLSTM.** Dehghani et al. [10] states that the recurrent structure is keen on inducing certain inductive biases which can make learning process more reasonable. To this end, we propose an ABLSTM, where LSTM which is also a kind of recurrent neural networks can better capture the long-term dependency in sequence and attention mechanism can help us preserve some global information which LSTM may forget after a long periods of time (attention mechanism will be introduced in later prediction-level modeling). To be specific, we feed the sequence $\{e_1, e_2, \ldots, e_N\}$ to a bidirectional lstm layer [18, 35, 36]. The bidirectional lstm layer outputs its hidden state vectors $\mathcal{H} = \{h_1, h_2, \ldots, h_N\}$.

***Variant II*: HSAN.** Considering recurrent structure precludes parallelization and is memory consumption, here we use self-attention architecture eschewing recurrence as an alternative variant, which can model the global dependencies from different positions of a single sequence [40]. However, the behavior sequences of players in games are always significantly long, it's really hard to apply a single Vanilla SAN due to its huge amount of memory requirement in accord with the sequence. To address this issue, we propose a novel HSAN, which is consisted of local self-attention and global self-attention. HSAN is capable of capturing both the local dependency and the global dependency of user behavior sequences, while the time cost and memory cost are largely saved. Our main idea is to split the long input sequence into several length-equal sub-sequences and every player has the same interval range in each sub-sequence. Local self-attention is conducted in each sub-sequence and global self-attention is conducted between sub-sequences. The detailed structure is shown in Figure 6.

Local self-attention aims to capture the local dependency of player's behavior sequence. We split the high-dimension input sequence into several blocks and conduct self-attention module in each block, which significantly contributes to the improvement in computational efficiency through its mechanism of shared parameters and capability of parallelizable computation.

We observe that players' event sequences tend to lack of diversity and their behaviors are much sparser when players in the low level areas, i.e., the number of players events is fewer. Event sequences will alter obviously and become more diverse as players level up. So we split an input event sequence of embedding vector $\{e_1, e_2, \ldots, e_N\}$ into M sub-sequences $\mathcal{E}^m$ with equal length R according to the occurrence probability of each level $p(l_i)$, $0 \leq i \leq l_{max}$, where $\sum_{l_i=1}^{l_{max}} p(l_i) = 1$

and l_{max} is the max level of all players. Each sub-sequence $\mathcal{E}^m$corresponds to a level set L_m. We keep the sum of occurrence probability $p(l_i)$ in each L_m almost equal.

$$\sum_{i \in L_1} p(l_i) \approx \sum_{i \in L_2} p(l_i) \approx \cdots \approx \sum_{i \in L_M} p(l_i) \tag{2}$$

$$\{\mathcal{E}^m\}_{m=1}^{m=M} = \{\mathcal{E}^1, \mathcal{E}^2, \ldots, \mathcal{E}^M\}, M = \frac{N}{R} \tag{3}$$

where $\mathcal{E}^1 = \{e_1, e_2, \ldots, e_R\}$, $\mathcal{E}^2 = \{e_{R+1}, e_{R+2}, \ldots, e_{2R}\}, \ldots,$ and $\mathcal{E}^M = \{e_{N-R+1}, e_{N-R+2}, \ldots, e_N\}$ (with padding if necessary).

Multi-head self-attention [40] is applied to $\mathcal{E}^m$ in order to accurately capture the local dependency of players' behaviors inside each block, i.e.,

$$\begin{aligned} \mathcal{Z}^m &= MultiHead(Q_m, K_m, V_m) \\ &= Concat\left(head_m^1, head_m^2, \ldots, head_m^D\right) W^O \end{aligned} \tag{4}$$

$$\begin{aligned} head_m^d &= attention\left(Q_m W_d^{Q_m}, K_m W_d^{K_m}, V_m W_d^{V_m}\right) \\ &= softmax\left(\frac{Q_m W_d^{Q_m} (K_m W_d^{K_m})^T}{\sqrt{R}}\right) V_m W_d^{V_m} \end{aligned} \tag{5}$$

where D is the number of parallel attention layers or heads; Q_m, K_m, V_m are local queries matrix, local keys matrix, and local values matrix of the m-th sub-sequence respectively; the projections are parameter matrices $W_d^{Q_m} \in \mathbb{R}^{d_{\text{event}} \times R}$, $W_d^{K_m} \in \mathbb{R}^{d_{\text{event}} \times R}$, $W_d^{V_m} \in \mathbb{R}^{d_{\text{event}} \times R}$ and $W^O \in \mathbb{R}^{d_{\text{event}} \times d_{\text{event}}}$.

The purpose of global self-attention is to capture the global dependency among the whole sequence of players. To be specific, we concatenate the output of local SAN as the input of global SAN, i.e., $\mathcal{Z} = Concat(\mathcal{Z}^1, \mathcal{Z}^2, \ldots, \mathcal{Z}^M) \in \mathbb{R}^{N \times d_{\text{event}}}$. Then, a global multi-head self-attention mechanism is employed to learn the long-term dependency between sub-sequences.

$$\mathcal{G} = MultiHead(Q, K, V) \tag{6}$$

where Q, K, V are global queries matrix, global keys matrix, and global values matrix respectively.

However, models with deep layers are hard to train, and may suffer from vanishing or exploding gradient problems. Considering these problems, here we further employ a residual connection [16] around the input and output of global SAN, and then we can get the output $\mathcal{H} = \{h_1, h_2, \ldots, h_N\}$ as:

$$\mathcal{H} = U\mathcal{Z}V + \mathcal{G} \tag{7}$$

where $U \in \mathbb{R}^{N \times N}$ and $V \in \mathbb{R}^{d_{\text{event}} \times d_{\text{event}}}$are two learnable parameter matrices.

(2) Knowledge-level modeling: The knowledge-level modeling can be considered as a kind of multi-feature fusion. Empirically, we employ the prior information from the three extra features aforementioned: the interval, the count, and the level, which are also useful in bot detection. Differently from the EventIDs in the sequence, there is no temporal relations between the interval (or the count, the 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 on game events, we find that the occurrence of a certain event can be taken as a Poisson process. Here we fit two gamma distributions for human's interval of each event type and bot's respectively, i.e., $Interval_h \sim \Gamma(\alpha_h, \beta_h)$, $Interval_b \sim \Gamma(\alpha_b, \beta_b)$. We show the corresponding probability density function below:

$$f(x; \alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)} \tag{8}$$

Specifically, for the i-th event in a sequence, we will compute two probabilities:

$$I_{i,h} = f\left(t_i;\, \alpha_{i,h}, \beta_{i,h}\right),\ I_{i,b} = f\left(t_i;\, \alpha_{i,b}, \beta_{i,b}\right) \tag{9}$$

These two probabilities help us efficiently differentiate humans and bots.

— Similarly, we can get another two probabilities from the information of the count:

$$C_{i,h} = f\left(c_i;\, \alpha_{c,h}, \beta_{c,h}\right),\ C_{i,b} = f\left(c_i;\, \alpha_{c,b}, \beta_{c,b}\right) \tag{10}$$

— In terms of the player's 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 the level l_i:

$$L_i = P\left(l_i\right) = \frac{N_{bot}^{l,e}}{N_{bot}^{l,e} + N_{human}^{l,e}} \tag{11}$$

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

$$B_i = P\left(e_i\right) = \frac{N_{bot}^{e}}{N_{bot}^{e} + N_{human}^{e}} \tag{12}$$

We obtain the extended feature vectors $O = [o_1, \ldots, o_N]^T$ by concatenating the hidden vectors with the statistical values:

$$o_i = h_i \circ I_{i,h} \circ I_{i,b} \circ C_{i,h} \circ C_{i,b} \circ L_i \circ B_i \tag{13}$$

(3) Prediction-level modeling: Lastly, we aim to predict whether a player is a game bot or not. Also, different prediction-level modeling fits different variants (ABLSTM and HSAN) mentioned above, respectively.

For ABLSTM variant, we further apply a attention mechanism [40] in conjunction with LSTM to fully exploit the global information in player's sequence. The insight is that LSTM may leave out important information from the beginning especially when the input sequence become much longer, while attention mechanism can help us model the dependencies in sequential data regardless of their distance in the input or output sequences [3, 23]. To be specific, it outputs a sequence representation which places different importance contribution on the extended feature vectors:

$$a_i = softmax\left(w_i{}^T O + b_i\right),\ r_i = a_i \odot o_i \tag{14}$$

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

For HSAN variant, we directly connect O to a single neuron with sigmoid activation function to get a classification probability.

5.2.2 Time-interval Event2vec (Pre-training). Inspired by the work of App2vec by Qiang Ma [29], 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, if the time gap between a specific event and the target event is

shorter, this event should contribute more in predicting the target event. To this regard, we define the weight of each event e_i to target event e_t as:

$$w(e_i, e_t) = \alpha^l \tag{15}$$

where α is chosen as 0.8 according to [29], and l is the amount of the time gap (e.g., calculated based on 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:

$$\overline{v} = \frac{\sum_{-c \le j \le c, j \ne 0} w(e_j, e_t) v_j}{\sum_{-c \le j \le c, j \ne 0} w(e_j, e_t)} \tag{16}$$

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 unsupervised-based Seq2vec as mentioned below.

5.2.3 Unsupervised-based Seq2vec (Pre-training). A significant property of the unsupervised-based Seq2vec is that it can be trained with large quantities of unlabeled data, which is especially useful for tasks that have limited labeled data [9]. Here, we take advantage of the encoder-decoder structure to reconstruct the own input sequence, and for this the output of encoder can be taken as the latent vector representation of the input sequence. Here, two variants of unsupervised-based Seq2vec models are proposed: Sequence Autoencoder corresponding to the ABLSTM and H-Trans corresponding to HSAN, respectively.

***Variant I*: Sequence Autoencoder.** Sequence Autoencoder can be considered as one of the pre-training step of ABLSTM. Our approach to Sequence autoencoder is inspired by the work in semi-supervised sequence learning by Andrew M. Dai [9], which has been successfully used in many classification tasks. Key to this approach is to use Seq2Seq by Ilya Sutskever [38] as a pre-training step and then use its parameters as a starting point for other supervised training models.

More concretely, we use a one-layer bidirectional recurrent neural network (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.

***Variant II*: H-Trans.** Here, we propose a novel H-Trans (Hierarchical Transformer) model which can be considered as one of the pre-training steps of HSAN. Our approach to H-Trans is inspired by the popular Transformer structure [9]. Instead of using the RNN or convolutional neural network, H-Trans computes position wised attention score directly by several fully-connected networks. We apply a local-attention mechanism with shared parameters which is same to the part of HSAN in encoder, and another in decoder.

In our work, we pre-train all players' event embedding vector by H-Trans, and we take its encoder's parameters as the initial weights of HSAN's encoder for fine-tuning.

5.2.4 Transfer Learning. 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 [31]. 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, 8, 30].

In our article, we conduct transfer learning by reusing a unsupervised-based Seq2vec model for a given (source) game quest on another (target) one. For example, after we have pre-trained the

unsupervised-based Seq2vec model of daily quest, what we can do is to reuse the parameters from it of the daily quests as the initial parameters for the supervised-based classifier of main quest. This solution skips the process of training the unsupervised-based Seq2vec model of main quest, which can still get ideal performance but largely accelerates the pre-training phase.

5.3 Online Inference

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

5.3.1 Classification Solution. In the supervised solution, the well-trained supervised-based classifier 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.

5.3.2 Clustering Solution. In the unsupervised solution, here we propose a model combining unsupervised-based Seq2vec and DBSCAN. DBSCAN is a data density-based clustering algorithm proposed by Martin Ester [12]. In our solution, we first extract the vector representation of EventID sequences from the well-trained unsupervised-based Seq2vec model. Next, DBSCAN is performed 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.

5.4 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.

5.4.1 Humanset Resampling. We pull down new records of human players from online servers and re-sample them with the sampling method proposed in Section 5.1.2. 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.

5.4.2 New Botset Detection. The clusters of game bots can be easily located through the Unsupervised Seq2vec and 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 supervised-based classifier 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 the supervised-based classifier, 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.

5.4.3 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 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 supervised-based classifier. Since the training time of supervised-based classifier is relatively short, we update the supervised-based classifier model based on our reconstructed training dataset in the short-term cycles.

5.4.4 Long-term Auto-iteration. Long-term auto-iteration is performed on offline training stage. Since the training time of time-interval Event2vec and the Unsupervied-based Seq2vec 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 supervised-based classifier is the same as the short-term auto-iteration based on the reconstructed training dataset.

6 EVALUATION

6.1 Experimental Setup

In this section, we evaluate our game bot detection framework NGUARD on a real-world dataset collected from a NetEase MMORPG. The dataset details can be found in Section 4. We use undersampling to keep our dataset balanced.

For a fair comparison, we (*i*) fix the embedding size of EventID to 128 according to the best-performed SA-ABLSTM and HSAN, and (*ii*) fix the length of players' behavior sequences N to 1k (padding when sequence length is less than 1k, truncation when sequence length is greater than 1k). The vocabulary size of EventID is limited to 13k. A dropout rate of 0.5 is used for all embedding layers, convolutional layers, LSTM layers, and fully-connected layers (for all baselines and our models). In the training procedure, the parameters are initialized by the xavier [14] method and then optimized with the Stochastic Gradient Descent. The batch size is 64. For ABLSTM model, the concatenation of forward and backward LSTMs gives us a dimension of 256. For HSAN, we set the number of encoder/decoder blocks as 1 and the number of attention heads D as 8. To achieve the best performance, we conduct carefully parameter analysis in Section 6.5.

In what follows, four sets of experimental studies are conducted:

- —We compare our proposed supervised approaches with state-of-art baselines of supervised methods.
- —We evaluate the performance of unsupervised based models.
- —We verify the necessity for the auto-iteration mechanism. Also, we conduct a case study to identify its ability of detecting mutated and unknown game bots.
- —Finally, we further conduct a parameter analysis experiment to analyze the influence of important parameters.

6.2 Comparisons of Supervised Solutions

Baseline Method.

For our supervised solution, we compare with the following baselines to detect bots from players, which can also be formulated as a binary classifier.

— *Multi-layer Perceptron (MLP)*: This method is the most straight-forward way. It takes the frequencies of EventIDs as inputs. It then feeds them into two fully-connected layers get the prediction.
— *Convolutional Neural Network (CNN)*: This method is commonly used to capture the local patterns of players' behavior sequences, which operates on the sequential data of EventID concatenated with additional features of the interval, the count and the level. It then feeds the sequence data into an embedding layer to get the latent representation of EventID, following a convolution layer, a average pooling layer and finally a fully-connected layer to make the prediction.
— *Bidirectional LSTM (Bi-LSTM)*: This method aims to capture the global dependency and temporal information of players' behavior sequences, which also operates on the sequential data of EventID concatenated with additional features of the interval, the count and the level. It then feeds the sequence data into an embedding layer to get the latent representation of EventID, following a layer of Bi-LSTM and finally a fully-connected layer to get the result.
— *ABLSTM*: This is one of the Sequence-level Modeling Variants in our work. (See model details in Section 5.2.1)
— *TL-ABLSTM*[6]: Here, transfer learning is conducted among the pre-trained Sequence Autoencoder of task-specific SA-ABLSTM (i.e., main quest, daily quest, and instanced quest). (See model details in Section 5.2.4)
— *SA-ABLSTM*: This is the ABLSTM pre-trained with Sequence Autoencoder (Here, we pretrain the Sequence Autoencoder for each quest, respectively).
— *HSAN*: This is another Sequence-level Modeling Variants in our work. (See model details in Section 5.2.1)
— *TL-HSAN*[6]: Here, transfer learning is conducted among the pre-trained H-Trans of task-specific HSAN (i.e., main quest, daily quest, and instanced quest). (See model details in Section 5.2.4)
— *Trans-HSAN*: This is the HSAN pre-trained with H-Trans (Here, we pretrain the H-Trans for each quest, respectively).

Performance. The experiment results are shown in Table 1, with Precision, Recall and F1 as evaluation metrics. The bold numbers are the best results of all. First, we find that the performances of MLP, CNN, and Bi-LSTM are not good enough. The reason might be that they cannot effectively capture the local dependency and global dependency of players' behavior sequences. Next, let's take a look at the utility of transfer learning in our framework. Surprisingly, we observe that the performance of TL-ABLSTM (or TL-HSAN) only slightly decreases compared with SA-ABLSTM (or Trans-HSAN). However, with the help of transfer learning, we could save lots of training time in unsupervised-based Seq2vec but still get really good result. Furthermore, we can observe that the two variants of supervised-based classifier yields the best result on different quests, though SA-ABLSTM and Trans-HSAN process their unique advantages on specific task over the other. This is likely due to the fact that SA-ABLSTM tends to be incapable of remembering long sentences, while Trans-HSAN shows more advantage in capturing the global dependency in long sequence. Since player sequences of main quest and instanced quest are relatively long, Trans-HSAN always performs better in this case. On the other hand, Trans-HSAN foregoes the inductive bias of RNN-based model, while SA-ABLSTM's capability of learning iterative or recursive transformations is really crucial especially when the sequence is not very long. Since daily quest is the kind of task

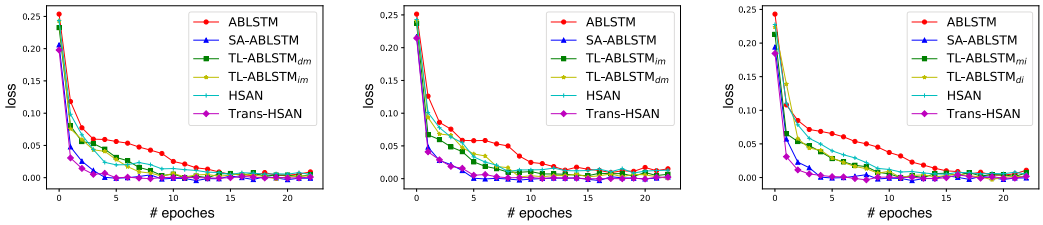
[6]TL-ABSLTM$_{ab}$ or TL-HSAN$_{ab}$ means 'a' quest transfers to 'b' quest, where 'm' is an abbreviation of main quest, 'd' is an abbreviation of daily quest, 'i' is an abbreviation of instanced quest.

Table 1. Performance Comparison of Supervised Method

		Precision	Recall	F1
Main quest	MLP	0.9618	0.9773	0.9694
	CNN	0.9721	0.9807	0.9764
	Bi-LSTM	0.9809	0.9865	0.9837
	ABLSTM	0.9851	0.9882	0.9866
	TL-ABLSTM$_{im}$	0.9878	0.9896	0.9887
	TL-ABLSTM$_{dm}$	0.9893	0.9906	0.9899
	SA-ABLSTM	0.9904	0.9912	0.9908
	HSAN	0.9713	0.9753	0.9733
	TL-HSAN$_{im}$	0.9902	0.9931	0.9916
	TL-HSAN$_{dm}$	0.9912	0.9935	0.9923
	Trans-HSAN	**0.9921**	**0.9943**	**0.9932**
Daily quest	MLP	0.9528	0.9609	0.9568
	CNN	0.9633	0.9712	0.9672
	Bi-LSTM	0.9709	0.9728	0.9718
	ABLSTM	0.9716	0.9774	0.9745
	TL-ABLSTM$_{id}$	0.9736	0.9721	0.9728
	TL-ABLSTM$_{md}$	0.9771	0.9742	0.9756
	SA-ABLSTM	**0.9838**	**0.9861**	**0.9849**
	HSAN	0.9713	0.9753	0.9733
	TL-HSAN$_{id}$	0.9789	0.9800	0.9794
	TL-HSAN$_{md}$	0.9793	0.9809	0.9801
	Trans-HSAN	0.9802	0.9855	0.9828
Instanced quest	MLP	0.9441	0.9571	0.9506
	CNN	0.9552	0.9643	0.9597
	Bi-LSTM	0.9612	0.9732	0.9672
	ABLSTM	0.9674	0.9786	0.9730
	TL-ABLSTM$_{di}$	0.9698	0.9801	0.9749
	TL-ABLSTM$_{mi}$	0.9704	0.9808	0.9756
	SA-ABLSTM	0.9721	0.9816	0.9768
	HSAN	0.9723	0.9818	0.9770
	TL-HSAN$_{di}$	0.9743	0.9744	0.9743
	TL-HSAN$_{mi}$	0.9765	0.9754	0.9759
	Trans-HSAN	**0.9784**	**0.9823**	**0.9803**

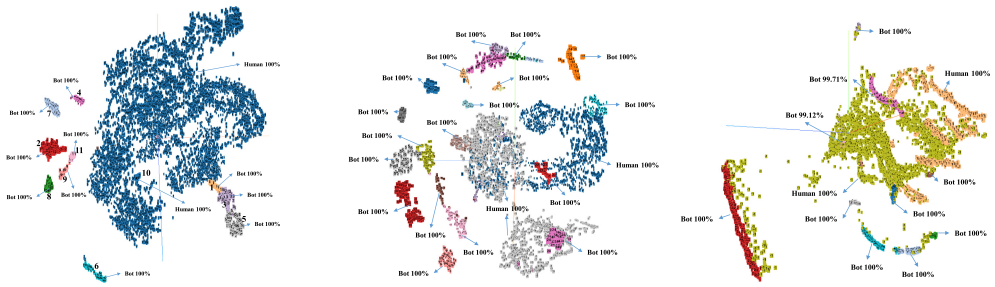
that will not occupy too much time, the player sequences are relatively shorter, which is the main reason why SA-ABLSTM performs better in daily quest.

Convergence speed. To further illustrate the convergence speed of our method, we plot the objective vs. the number of epochs in Figure 7. We find that Trans-HSAN always starts with the best initial parameters and converges fastest to the lowest loss value in all quests, which illustrates the advantage of Trans-HSAN in efficiency. Notice that SA-ABLSTM shows a similar convergence speed with Trans-HSAN in daily quests. The reason is that the parameter number of SA-ABLSTM is relatively few when the sequence is not very long, which contributes to its convergence speed. Furthermore, we observe that the pre-training step can really help our model converge well (ABLSTM vs. SA-ABLSTM and HSAN vs. Trans-HSAN). Also, we present the results of TL-ABLSTM models



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

Fig. 7. Convergence speed comparison of ABLSTM and its extensions.



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

Fig. 8. t-SNE visualization of clustering result of SA-DBSCAN.

to illustate the utility of transfer learning. 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.

6.3 Evaluation of Clustering Solution

We further conduct experiments to validate the effectiveness of our clustering solution in bot detection. Due to space limit, we only present the experimental result of SA-DBSCAN (Sequence Autoencoder and DBSCAN) in Figure 8, while the Trans-DBSCAN (H-Trans and DBSCAN) shows similar result. Figure 8 visualizes the clustering result of SA-DBSCAN in 2-d t-SNE embedding plots. The clusters are assigned different colors to differentiate them. Their relative positions and densities provide rich information, which can help us quickly understand the similarity between players and the diversity of each cluster. 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.

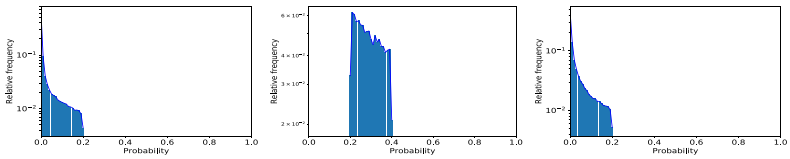
6.4 The Necessity of the Auto-iteration Mechanism

Based on Figure 8(a), we further extract the top six typical clusters to analyze their characteristic. Specifically, the type of each typical cluster is labeled as ground truth by the operation teams, shown in Table 2. What we want to focus on is the clusters belonging to mutated game bots and unknown game bots, which are indispensable for constructing new botset during auto-iteration mechanism.

We compute the prediction probability distribution of each cluster in order to determine the cluster type it belongs to. The results are shown in Figure 9. We can see that, Cluster "10" is the largest cluster with low probabilities, which is considered as human players. Small-scale clusters

Table 2. Cluster Type of Figure 8(a)

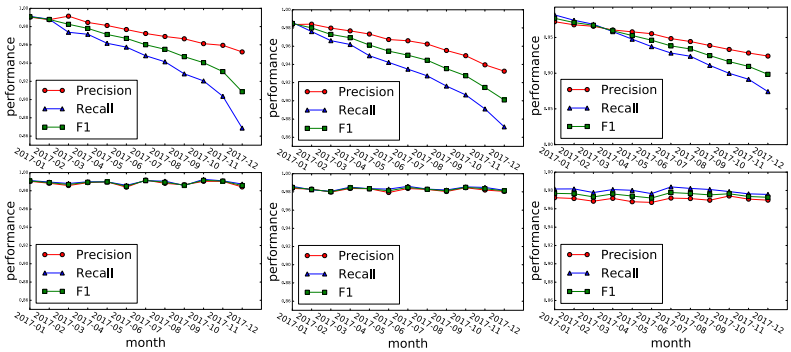
Cluster ID	Cluster Type Description
10	Cluster of human
6	Cluster of unknown game bots
5	Cluster of known game bots
1, 3, 4	Cluster of mutated game bots



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

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

Fig. 9. Clusters' probability distribution in Figure 8(a).

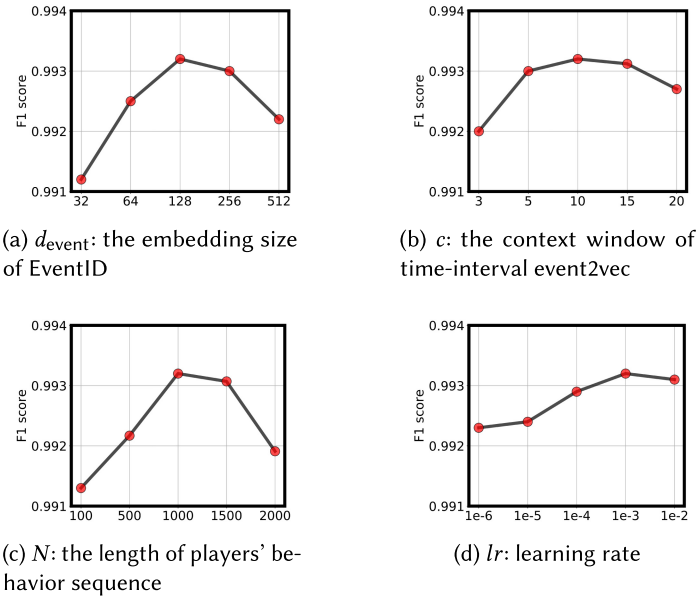


(a) Main quests (b) Daily quests (c) Instanced quests

Fig. 10. Performance with the auto-iteration mechanism.

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 "4" with middle probabilities are evaluated as mutated game bots. This demonstrates the effectiveness of identifying cluster type according to probability distribution when conducting the auto-iteration mechanism.

Secondly, we evaluate the performance of our framework with the auto-iteration mechanisms, as shown in Figure 10. The left columns are our framework without the auto-iteration mechanisms, while the right columns are our framework with the auto-iteration mechanisms. In a real MMORPG production, we set the short-term cycle as 1 month and the long-term cycle as



(a) d_{event}: the embedding size of EventID

(b) c: the context window of time-interval event2vec

(c) N: the length of players' behavior sequence

(d) lr: learning rate

Fig. 11. Hyper-parameter analysis.

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.

6.5 Parameter Sensitivity

In this section, we present the suitable range of some crucial hyper-parameters for reference. There are three important types of hyper-parameters affecting the model performance, which are the the embedding size of EventID d_{event}, the context window of time-interval event2vec c, the length of players' behavior sequence N, and the learning rate lr. To decide the default parameters settings, we vary the values of major parameters to observe how the F1 changes in the task of bot detection. In detail, we set the embedding size of EventID d_{event} as 32, 64, 128, 256, and 512; the context window of time-interval event2vec c as 3, 5, 10, 15, and 20; the length of players' behavior sequences N as 100, 500, 1,000, 1,500, and 2,000; and the learning rate as 1e-6, 1e-5, 1e-4, 1e-3, and 1e-2, respectively. Please be noted that the value of context window c remains as 5 while studying the embedding size d_{event}, the length of players' behavior sequence N and the learning rate lr; the value of embedding size d_{event} remains as 128 while studying the context window c, the length of players' behavior sequence N and the learning rate lr; the length of players' behavior sequence N remains as 1,000 while studying the context window c, the embedding size d_{event} and the learning rate lr; and the learning rate lr remains as 1e-2 while studying the context window c and the embedding size d_{event}.

Due to space limitations and easier handling, we only present the results on HSAN in Figure 11. From Figure 11(a), we observe that our model yields good results when d_{event} is set to 128. The reason is that bigger d_{event} needs more parameters to be learned which will become difficult to train, while smaller d_{event} is not enough to represent the latent information. From Figure 11(b), we can see that the best performance can be achieved when c is set to 10. It can be explained that small

context window can not provide us enough information, while too large context window would in turn prevent us from capturing the most important part of the sequence data. From Figure 11(c), we observe that our model yields good results when N is shorter than 1,000, while the performance will decrease if N is too long. The reason is that a too short sequence length can not represent the players' portrait, while too long sequence length is not necessary and will significantly decrease the algorithm efficiency. Figure 11(d) shows that our model performs best when lr is set to 1e-2. A bigger learning rate may lead to divergence, while a smaller learning rate may lead to slow convergence.

Based on these results, for NGUARD, the embedding size of EventID d_{event} is set to 128, the context window of time-interval event2vec c is set to 10, the length of players' behavior sequence N is set to 1,000, and the learning rate lr is set to 1e-2, because that these default settings can achieve good performance in the task of bot detection.

7 CONCLUSION

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 online games. The game industry has suffered serious threats from game bots, and game bot detection remains one of the most urgent problems that game publishers need to address. Hence, our major problem is that how to detect game bots that come along with almost every MMORPG.

In this article, we propose a generalized game bot detection framework for 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. 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. For supervised models, we propose two variants for sequence modeling: ABLSTM and HSAN, which are capable of capturing both local and global dependency in user behavior sequences. Furthermore, we propose an auto-iteration mechanism to automatically adapt to the mutated and new game bots, which is suitable for rapidly changing online environment. 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.

For future work, it will be interesting to analyze user characteristics from multi-source data, for example, integrating the social relationships between users into our framework. In addition, we mean to generalize our model to more kinds of games such as First-person Shooting games. It's also a great challenge to detect game bots in some simple game tasks where the human actions are very similar to game bots.

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 *Proceedings of the 12th International Conference on Computational Science and Engineering*.

[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 *Proceedings of the 10th European Conference on Computer Vision*.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations*.

[4] 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*. ACM.

[5] 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 Journal on Advances in Signal Processing* (2009), 797159.
[6] 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*. ACM.
[7] 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 (2016), 3572–3587.
[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12 (2011), 2493–2537.
[9] Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*. Curran Associates, Inc.
[10] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Proceedings of the 7th International Conference on Learning Representations*.
[11] Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14, 2 (1990), 179–21.
[12] 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 2nd International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 226–231.
[13] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2019. A survey of parallel sequential pattern mining. *ACM Transactions on Knowledge Discovery from Data* 13, 3 (2019), 1–34.
[14] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. PMLR, 249–256.
[15] Philippe Golle and Nicolas Ducheneaut. 2005. Preventing bots from playing online games. *Computers in Entertainment* 3, 3 (2005), 3–3.
[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[17] Sylvain Hilaire, Hyun chul Kim, and Chong kwon Kim. 2010. How to deal with bot scum in MMORPGs? In *Proceedings of the IEEE International Workshop Technical Committee on Communications Quality and Reliability*.
[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[19] Adele Lu Jia, Siqi Shen, Ruud Van De Bovenkamp, Alexandru Iosup, Fernando Kuipers, and Dick H. J. Epema. 2015. Socializing by gaming: Revealing social relationships in multiplayer online games. *ACM Transactions on Knowledge Discovery from Data* 10, 2 (2015), 1–29.
[20] Michael I. Jordan. 1986. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 531–546.
[21] Ah Reum Kang, Seong Hoon Jeong, Aziz Mohaisen, and Huy Kang Kim. 2016. Multimodal game bot detection using user behavioral characteristics. *SpringerPlus* 26, 5 (2016), 523.
[22] Hyungil Kim, Sungwoo Hong, and Juntae Kim. 2005. Detection of auto programs for MMORPGs. In *Proceedings of the 18th Australian Joint Conference on Advances in Artificial Intelligence*.
[23] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *Proceedings of the 5th International Conference on Learning Representations*.
[24] Hyukmin Kwon, Aziz Mohaisen, Jiyoung Woo, Yongdae Kim, Eunjo Lee, and Huy Kang Kim. 2017. Crime scene reconstruction: Online gold farming network analysis. *IEEE Transactions on Information Forensics and Security* 12, 3 (2017), 544–556.
[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
[26] 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 *Proceedings of Network and Distributed System Security Symposium*.
[27] 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*. Springer International Publishing.
[28] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 14th Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1412–1421.
[29] Qiang Ma, Shan Muthu Muthukrishnan, and William Simpson. 2016. App2Vec: Vector modeling of mobile apps and applications. In *Proceedings of the 6th IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*.

[30] 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 27th IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society.

[31] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[32] Christian Platzer. 2011. Sequence-based bot detection in massive multiplayer online games. In *Proceedings of the 8th International Conference on Information, Communications Signal Processing*.

[33] Jordan B. Pollack. 1991. The induction of dynamical recognizers. *Machine Learning* 7, 2–3 (1991), 227–252.

[34] 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*. IEEE Press.

[35] J. Schmidhuber, F. Gers, and D. Eck. 2002. Learning nonregular languages: A comparison of simple recurrent networks and LSTM. *Neural Computation* 14, 19 (2002), 2039–2041.

[36] M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[37] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018. DiSAN: Directional self-attention network for RNN/CNN-free language understanding. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*.

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*. Curran Associates, Inc.

[39] Jianrong Tao, Jiarong Xu, Linxia Gong, Yifu Li, Changjie Fan, and Zhou Zhao. 2018. NGUARD: A game bot detection framework for NetEase MMORPGs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18)*. ACM, New York, NY, 811–820.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc.

[41] Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1, 2 (1989), 270–280.

[42] Jiyoung Woo and Huy Kang Kim. 2012. Survey and research direction on online game security. In *Proceedings of the 39th International Conference Workshop at SIGGRAPH Asia*. ACM.

[43] 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?. In *Proceedings of the 11th ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM.

[44] Roman V. Yampolskiy and Venu Govindaraju. 2008. Embedded noninteractive continuous bot detection. *Computers in Entertainment* 5, 4 (2008), 1–11.

Received January 2020; accepted May 2020