# FingFormer: Contrastive Graph-based Finger Operation Transformer for Unsupervised Mobile Game Bot Detection

$^{\dagger}$Wenbin Li$^{1,2}$, $^{\dagger}$Xiaokai Chu$^{1,2}$, Yueyang Su$^{1,2}$, $^{*}$Di Yao$^{1}$, Shiwei Zhao$^{3}$, Runze Wu$^{3}$, Shize Zhang$^{3}$, Jianrong Tao$^{3}$, Hao Deng$^{3}$, $^{*}$Jingping Bi$^{1}$
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China$^{1}$
University of Chinese Academy of Sciences, China$^{2}$,
NetEase Fuxi AI Lab, China$^{3}$
{liwenbin20z,chuxiaokai,yaodi,bjp}@ict.ac.cn,suyueyang19@mails.ucas.edu.cn
{zhaoshiwei,wurunze1,zhangshize,hztaojianrong,denghao02}@corp.netease.com

## ABSTRACT

This paper studies the task of detecting bots for online mobile games. Considering the fact of lacking labeled cheating samples and restricted available data in the real detection systems, we aim to study the finger operations captured by screen sensors to infer the potential bots in an unsupervised way. In detail, we introduce a Transformer-style detection model, namely FingFormer. It studies the finger operations in the format of graph structure in order to capture the spatial and temporal relatedness between the two hands' operations. To optimize the model in an unsupervised way, we introduce two contrastive learning strategies to refine both finger moving patterns and players' operation habits. We conduct extensive experiments under different experimental environments, including the synthetic dataset, the offline dataset, as well as the large-scale online data flow from three mobile games. The multi-facet experiments illustrate the proposed model is both effective and general to detect the bots for different mobile games.

## CCS CONCEPTS

• **Computing methodologies → Machine learning**; **Anomaly detection**.

## KEYWORDS

mobile game, bot detection, Transformer, contrastive learning, clustering

**ACM Reference Format:**
$^{\dagger}$Wenbin Li$^{1,2}$, $^{\dagger}$Xiaokai Chu$^{1,2}$, Yueyang Su$^{1,2}$, $^{*}$Di Yao$^{1}$, Shiwei Zhao$^{3}$,, Runze Wu$^{3}$, Shize Zhang$^{3}$, Jianrong Tao$^{3}$, Hao Deng$^{3}$, $^{*}$Jingping Bi$^{1}$ . 2022. FingFormer: Contrastive Graph-based Finger Operation Transformer for Unsupervised Mobile Game Bot Detection. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3512272

$^{*}$ Corresponding authors. $^{\dagger}$ Equal contribution.

*WWW '22, April 25–29, 2022, Virtual Event, Lyon, France*
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9096-5/22/04.
https://doi.org/10.1145/3485447.3512272

## 1 INTRODUCTION

In the past decades, the popularization of mobile terminal devices (*e.g.*, smartphone and tablet) and mobile operating systems has motivated a rapid growth of pluralistic online mobile games[1]. Nowadays, mobile games have accounted for nearly half of the game market share, and their revenue is growing stably every year. For many Internet enterprises, the mobile game business has become an important part of the profit, *e.g.*, it made up fifty percent of NetEase's income with 6 billion dollars in 2020.[2].

On the other hand, a popular mobile game is inevitably impacted by the game bots[3]. Typical bots, such as the cheating programs in a mobile Fighting Game (FTG), simulate the finger operations with unreasonable reaction speed to compete with the honest players in an unfair way. It not only brings frustrating experience to the honest players, but also causes huge economic loss to the game companies [23, 24]. Therefore, an effective bot detection approach is always urgent and of vital importance for any game company.

To date, there has been a great effort to detect game bots based on multi-view information, such as monitoring suspicious processes and building the blacklist [24], analyzing players' profile, behaviors or social relationships [14, 22, 23].

However, most current efforts only focus on the *P*ersonal *C*omputer (*PC*) games, while little attention is paid to the bots in the flourishing mobile games. Compared to the various heterogeneous data in *PC* games, the available data for mobile game bot detection is usually plain and limited. Due to the privacy protocol, in most cases only the *screen sensor data* could be available for analysis [13]. The screen sensor data is collected by a set of sensors under the screen, which records the events triggered by fingers, such as a click or drag, as well as the corresponding screen position and timestamp. Therefore, due to the limited data and the characteristics of the mobile game business, a bot detection approach has to overcome the following challenges:

- **Double hands operating mode.** Most mobile games require the player to play with both hands, *e.g.*, the left hand controls the role's movement while the right hand releases the skills, so the operations of two hands are highly related. Thus, an effective approach should not only analyze the

[1]For brevity, in the rest of the paper, we directly use the term "mobile game" to refer to "online mobile game".
[2]Refer to the financial report of NetEase in 2020 in http://gb.corp.163.com/gbnews/General.html
[3]The game bot in this paper refers to the illegal program/software that pretends to be a player in the game for unfair competition.

finger movement of each hand, but also consider the temporal and spatial associations between the operations of both hands for more precise prediction.
- **Lack of labeled cheating samples.** The daily sensor records are massive (*e.g.,* billions of), while the records generated by bots usually only account for a very small part (*e.g.,* thousands of), thus it is laborious to directly obtain the cheating samples by manual labelling. An unsupervised approach is urgently required to find the potential cheating operations from the tremendous amount of daily data.
- **General for different games or game scenes.** For a game company, it may operate dozens of mobile games at the same time. Also, most mobile games are not limited to a single game scene. For example, different maps or modes (*e.g.*, single-player mode and multi-player mode) will change the game scene, which can also influence the rule and the way to play. Moreover, the available games/scenes for model learning is usually restricted from a commercial perspective, thus the generality of the proposed approach for different games/scenes is also an important issue that we should consider.

In response to these challenges, this paper introduces a novel game bot detection model, called *FingFormer*, which aims to learn distinctive operation embeddings via a graph-based Transformer [29, 34] model. Specifically, to model the temporal-spatial associations between operations of different hands, we regard each record collected by the sensors as a *Finger Operation Graph*. Each node in the graph denotes a recorded finger point, the edge between two nodes of the same hand models the finger movement in the spatial context, and the nodes of different hands are associated by another type of edge which models the temporal relatedness between operations. After that, the proposed FingFormer model takes this graph as input to study the operation embedding via a graph-based Transformer model, which contains a *Graph-based Encoding* module and a *Left-Right Interaction* module. Moreover, in order to optimize the model in an unsupervised way, we introduce two contrastive learning [25] strategies for both finger movement learning and operation habit learning. We conduct a set of experiments under different experimental environments, including the synthetic dataset, the offline dataset sampled from two real scenes, and the large-scale online data from three mobile games. Our proposed FingFormer can always achieve the best performance. In a nutshell, the contributions of this paper are:

- **Unsupervised Mobile Game Bot Detection.** To the best of our knowledge, this is the first work to study mobile game bot detection based on the sensor data and pre-training.
- **A Graph-based Transformer-style model.** To learn both hands operation in the temporal and spatial context, we model a sensor record as a *Finger Operation Graph.* Additionally, we introduce a novel graph-based Transformer model (FingFormer) to study the complex operation embeddings.
- **Contrastive Learning Strategy.** Facing the lack of labels , we introduce two novel contrastive learning strategies to learn distinctive embeddings for the successful inference of bots.
- **Extensive Offline and Online Experiments.** We evaluate the model on both offline experiments and the large-scale online bot detection. Experimental results show that the proposed FingFormer is effective on mobile game bot detection and also generic to different types of mobile games or scenes.

The rest of this paper is organized as follows: Section 2 briefly reviews the previous works and compares the difference between *PC* game bot detection and mobile game bot detection. Section 3 gives some formally definitions. We describe the proposed model FingFormer in detail in Section 4, validate the model by analyzing extensive offline and online experiments in Section 5, and conclude this work in Section 6.

## 2 RELATED WORK

The online games, such as personal computer games and mobile games, have been one of the largest growing Internet business in the world, attracting a large mount of people from different ages, nationalities, and occupations [20]. However, the thriving online game industries are always facing the serious threats from game bots. A game bot usually pretends to be a human player but easily achieves great superiority over the honest players due to its unfair operations, such as the extremely accurate shots or incredible reaction speed in First-Person Shooting (*FPS*) games. It is indisputable that game bots inevitably destroy the game experience of normal players, and seriously hurt the in-game ecosystem and the company's revenue. Therefore, a series of approaches have been proposed to detect game bots in the past decades, while most of them focus on the personal computer (PC) games.

For example, many game companies have developed the anti-cheating systems to serve a better competitive environment for players, in order to keep the game communities growing up healthily. However, the anti-cheating systems also have their inherent problems, namely the system hysteresis. Specifically, it is always being one step behind the most sophisticated cheaters, due to the searching for malware or evidence that game bots has been tampered with [20].

Another trend of game bot detection for PC games is based on the popular machine learning methodology, especially the deep learning models [8, 26, 31]. They usually design specific models based on the utilized information to predict whether a player is a bot or not. For example, literature [1] adopted a concise neural network model to discriminate human users from game bots based on the human-craft features extracted from both player profile and game scenes. While literature [22] designed a combination of supervised and unsupervised methods based on the behavior sequences. Some researchers also detected the suspicious players by analyzing their social relationships. For example, literature [14] inferred the potential bots based on the virtual goods exchange networks. Some other works [4, 5, 17, 21] also incorporated other heterogeneous information to enhance the exploration of bots.

However, the study of bot detection for online mobile games has not been well-explored. Different from the abundant and heterogeneous information in PC games, the available information for bot detection in mobile game is limited. In particular, usually only screen sensor data is available for the bot analysis mainly for the following two reasons: (1) *privacy protection*, screen sensors capture the events on the screen without the request of other permissions of a mobile phone, thus the personal information is unable to be collected by the sensors, which effectively protects the privacy of players (2) *general among mobile games*, the screen sensor data is a common data type for any mobile game, thus an effective sensor data-based detection method will be generic for different mobile games. For instance, literature [30] detected auto clicks in mobile games based on simple features extracted from screen sensor data.

Moreover, the insufficient labelled cheating samples also brings a dilemma for a (semi-)supervised model to distinguish the bots from the massive daily records via the supervised signals. Therefore, this paper aims to deal with the unsupervised mobile game bot detection problem by introducing a systematic framework, FingFormer, for the construction of the real-world cheat recognition applications.

## 3 PROBLEM DEFINITION

Before introducing the approach, we first give the formal definitions related to the studied problem. Without loss of generality, we in this paper focus on the typical two-hands mobile games, *i.e.*, require both hands to operate, as the single-hand games can be regarded as a special case of the studied problem. Moreover, we assume that at each timestamp only one finger can operate on the screen for each hand, but both hands can operate at the same time. For the bot detection of the multi-touch mobile games (more than three fingers), such the music game, we leave them as a future work.

In this paper, we study the problem of mobile game bot detection based on the records collected by the screen sensors, where each record contains a sequence of *finger events* on the screen. A finger event is defined as:

**Definition 1** (*Finger Event*). A finger event $e$ denotes a process of a single finger from pressing the screen to leaving the screen. It is defined as $e = \{\mathcal{P}, \gamma\}$. $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ is a sequence of points on the screen. Each point $p_i$ is a triplet $(x_{p_i}, y_{p_i}, t_{p_i})$, where $(x_{p_i}, y_{p_i})$ is the coordinate of the screen and $t_{p_i}$ is the timestamp of point $p_i$, and $t_{p_i} < t_{p_{i+1}}$. $\gamma \in \Gamma$ denotes the current event event, and $\Gamma$ is a set of all possible event types that the sensors could capture, such as a click or a drag.

The sensors usually collect the data according to a certain time interval, so one finger event can be a sequence of points if it lasts for a long time, like a "drag" or a "long press". Also, it is noteworthy that the screen sensors cannot distinguish which finger (or hand) triggers the event on the screen.

**Definition 2** (*Record*). A record collected by sensors in a given time window is defined as $r = \{\tau, u\}$, where $u$ is the corresponding player id, which has been anonymized but the records of the identical player share the same id. $\tau$ is an event sequence generated by this player, *i.e.*, $\tau = \{e_1, e_2, \dots, e_{|\tau|}\}$.

To be precise, the event sequence $\tau$ is not a strict sequence from the aspect of time. Since the sensors cannot directly distinguish which hand triggers an event, for two events $e_i, e_j \in \tau$ and $i < j$, $e_i$ either happens before $e_j$ (they belong to the same hand) or they intersect in time (they belong to different hands). Finally, we define the studied unsupervised mobile game bot detection problem as:

**Definition 3** (*Mobile Game Bot Detection*). Given a set of records $\mathcal{R} = \{r_1, r_2, \cdots, r_{|\mathcal{R}|}\}$, this task aims to infer the potential cheating records in an unsupervised way.

**Table 1: The primary notations in this paper.**

| | |
|---|---|
| **Sensor Record** | |
| $\mathcal{R} = \{r_1, r_2, \cdots\}$ | a set of records collected by sensors |
| $r = \{\tau, u\}$ | a record |
| $u$ | the player id |
| $\tau = \{e_1, \dots, e_{\|\tau\|}\}$ | a sequence of finger events |
| $e = \{\mathcal{P}, \gamma\}$ | a finger event |
| $\gamma \in \Gamma$ | $\gamma$ is an event type, $\Gamma$ denotes all possible types |
| $\mathcal{P} = \{p_1, \dots, p_{\|\mathcal{P}\|}\}$ | a finger point sequence of the event $e$ |
| $p = (x_p, y_p, t_p)$ | a finger point |
| $(x_p, y_p)$, $t_p$ | the coordinate on the screen and timestamp |
| **Preprocess** | |
| $\mathcal{G} = \{\mathcal{G}^{(l)}, \mathcal{G}^{(r)}, \mathcal{E}^{(l,r)}\}$ | the finger operation graph of a record |
| $\mathcal{G}^{(l)} = \{\mathcal{V}^{(l)}, \mathcal{E}^{(l)}\}$ | the left-hand graph |
| $\mathcal{V}^{(l)} = \{(v_i^{(l)}, \boldsymbol{a}_i^{(l)})\}$ | the node set of the left-hand graph |
| $v_i^{(l)}$, $\boldsymbol{a}_i^{(l)}$ | a left-hand node and its attributes |
| $\mathcal{E}^{(l)}$ | the intra-edges of the left-hand graph |
| $\mathcal{E}^{(l,r)}$ | the inter-edges across different hands |
| $\boldsymbol{b}_{i,j}^{(l)}$, $\boldsymbol{b}_{i,j}^{(l,r)}$ | attributes of edges $(v_i^{(l)}, v_j^{(l)})$ and $(v_i^{(l)}, v_j^{(r)})$ |
| $\Delta t_{i,j}^{(l,r)}$ | the time interval between $v_i^{(l)}$ and $v_j^{(r)}$ |
| $d_{i,j}^{(l)}$ | the distance between $v_i^{(l)}$ and $v_j^{(l)}$ on screen |
| **Model Architecture** | |
| $\boldsymbol{E}$ | the learnable embedding table |
| $\Phi$ | the Graph-based Encoding module |
| $\Psi$ | the Left-Right Interaction module |
| $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ | $\boldsymbol{X}$ is the input of the module $\Phi$ $\boldsymbol{Y}, \boldsymbol{Z}$ are the input and output of $\Psi$, |
| $\boldsymbol{e}$ | the final embedding of a record (graph) |

## 4 FINGFORMER

As shown in Fig 1, the proposed FingFormer contains two concise parts: (1) a preprocess stage first converts each screen sensor record into a *Finger Operation Graph*; (2) the FingFormer model takes the graph as input and generates the graph embedding via two Transform-style modules, *i.e.*, the Graph-based Encoding module and the Left-Right Interaction module. Moreover, we introduce two contrastive strategies to optimize the model under the unsupervised scheme. The primary notations used in this paper is arranged in Table 1, and the following sections detail the whole framework.

### 4.1 Preprocess

To model the temporal-spatial relatedness of the two-hands operations, we regard each record as a *Finger Operation Graph*. We first explain how to build the graph, and then illustrate our motivation for such design.

*4.1.1 Graph building.* As shown in Fig 1, each node in graph is a recorded finger point on the screen. The finger operation graph contains two types of edges: (1) *intra-edge*, which models the finger movement of each hand in the context of space; (2) *inter-edge*, which illustrates the operating association between two hands in time.

Specifically, since the screen sensors cannot directly distinguish different hands, to construct the graph, we first need to divide the finger events into left hand and right hand. The process is primarily based on two empirical rules:

- **Continuity rule**. All the finger points of an event should belong to the same hand. In Fig. 1, for example, the points $p_1, p_2, p_3$ in once drag event should belong to the same hand.

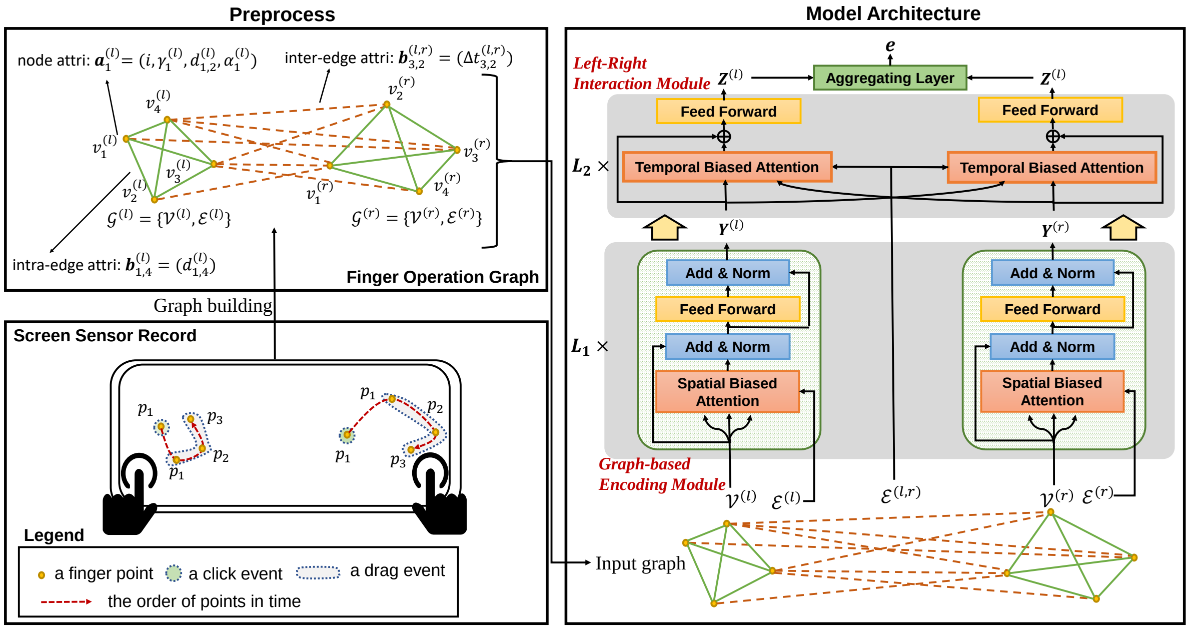
Preprocess
node attri: $\boldsymbol{a}_1^{(l)} = (i, \gamma_1^{(l)}, d_{1,2}^{(l)}, \alpha_1^{(l)})$
inter-edge attri: $\boldsymbol{b}_{3,2}^{(l,r)} = (\Delta t_{3,2}^{(l,r)})$
$\mathcal{G}^{(l)} = \{\mathcal{V}^{(l)}, \mathcal{E}^{(l)}\}$
$\mathcal{G}^{(r)} = \{\mathcal{V}^{(r)}, \mathcal{E}^{(r)}\}$
intra-edge attri: $\boldsymbol{b}_{1,4}^{(l)} = (d_{1,4}^{(l)})$
Finger Operation Graph
Graph building
Screen Sensor Record
Legend
a finger point
a click event
a drag event
the order of points in time
Model Architecture
$\boldsymbol{e}$
Aggregating Layer
Left-Right Interaction Module
Feed Forward
Temporal Biased Attention
$L_2 \times$
Graph-based Encoding Module
Add & Norm
Spatial Biased Attention
$L_1 \times$
Input graph

**Figure 1: An illustration of the proposed FingFormer. Left: the collected sensor data is first processed into a finger operation graph, which models the temporal and spatial relatedness between the collected finger points. Right: the FingFormer model contains two modules, where a Graph-based Encoding module takes the graph as input and studies the finger movement for each hand, while the Left-Right Interaction module fuses the information from both hands and refines the embedding.**

- **Position-based rule**. On top of the continuity rule, we additionally split the points by their positions on the screen. In practice, we regard the left half part of the screen belongs to the left hand, while the right part as the movement area for the right hand. Moreover, for an event that crosses the left and right areas, such as a "drag", we regard it belongs to the area where the first finger point appears.

According to above rules, each point in a record will be assigned to the corresponding hand.

After that, we construct a finger operation graph as

$$\mathcal{G} = \{\mathcal{G}^{(l)}, \mathcal{G}^{(r)}, \mathcal{E}^{(l,r)}\}, \tag{1}$$

where $\mathcal{G}^{(l)}$ is the left-finger operation graph and $\mathcal{G}^{(r)}$ is the right-finger operation graph, they are both complete graphs that model the finger movement of each hand. $\mathcal{E}^{(l,r)}$ refers to the set of inter-edges across $\mathcal{G}^{(l)}$ and $\mathcal{G}^{(r)}$.

We take left graph $\mathcal{G}^{(l)} = \{\mathcal{V}^{(l)}, \mathcal{E}^{(l)}\}$ as an example (same for the right graph). $\mathcal{V}^{(l)} = \{(v_i^{(l)}, \boldsymbol{a}_i^{(l)}) | i \in [1, \cdots, n_l]\}$ denotes the set of $n_l$ left nodes. Each node $v_i^{(l)}$ corresponds to a recorded point of left fingers. $\boldsymbol{a}_i^{(l)}$ denotes a set of attributes of node $v_i^{(l)}$:

$$\boldsymbol{a}_i^{(l)} = (i, \gamma_i^{(l)}, d_{i,i+1}^{(l)}, \alpha_i^{(l)}). \tag{2}$$

The index $i$ denotes the node $v_i^{(l)}$ is the $i^{th}$ point in left-hand operation sequence according to the timestamp, $\gamma_i^{(l)}$ denotes the event type. $d_{i,i+1}^{(l)}$ is the distance to the next finger point in time, and $\alpha_i^{(l)}$ represents the included angle, *i.e.*, the included angle between edges $(v_{i-1}^{(l)}, v_i^{(l)})$ and $(v_i^{(l)}, v_{i+1}^{(l)})$ on the screen. The attributes $d_{i,i+1}^{(l)}$ and $\alpha_i^{(l)}$ reveal the moving characteristic of the finger. $\mathcal{E}^{(l)}$ denotes the intra-edges between each pair of the nodes in left graph:

$$\mathcal{E}^{(l)} = \left\{(v_i^{(l)}, v_j^{(l)}, \boldsymbol{b}_{i,j}^{(l)}) | \quad v_i^{(l)}, v_j^{(l)} \in \mathcal{V}^{(l)}\right\}. \tag{3}$$

$\boldsymbol{b}_{i,j}^{(l)}$ is the edge attribute. Without loss of generality, we here use the distance between the two points as the attribute, *i.e.*, $\boldsymbol{b}_{i,j}^{(l)} = (d_{i,j}^{(l)})$, which models the spatial relatedness between every two points.

Between the two graphs, there is a set of inter edges $\mathcal{E}^{(l,r)}$ which models the temporal association between the left-hand and right-hand operations:

$$\mathcal{E}^{(l,r)} = \left\{(v_i^{(l)}, v_j^{(r)}, \boldsymbol{b}_{i,j}^{(l,r)}) | \quad v_i^{(l)} \in \mathcal{V}^{(l)}, v_j^{(r)} \in \mathcal{V}^{(r)}\right\}. \tag{4}$$

Also, we here only use the interval time between nodes $v_i^{(l)}$ and $v_j^{(r)}$ as the edge attribute, *i.e.*, $\boldsymbol{b}_{i,j}^{(l,r)}$=$(\Delta t_{i,j}^{(l,r)})$.

Moreover, all the attribute values (including node attributes and edge attributes) will be converted into the integer type, serving for the next embedding learning model.

*4.1.2 Discussion.* Here we bring a brief discussion of the finger operation graph by answering the following questions.

(1) *Why we use the distance as the attribute of intra-edge?*

The nodes of each graph is a sequence, *i.e.*, if $i$<$j$, the corresponding node (point) $v_i^{(l)}$ must happen before node (point) $v_j^{(l)}$. In that case, we do not have to model the temporal relationship between different nodes in each hand. On the other hand, we should model the relative distances between each two nodes, as it can reflect the typical finger moving patterns, which help us discover the abnormal samples.

(2) *Why we use the interval time as the attribute of inter-edge?*

As the two hands are limited to each side of the screen, the finger points of different hands are usually independent in space. However, they are highly related in the temporal context. For example, a game requires the left finger to control the direction while the right finger to release the skills. In this case, the operations of the left and right fingers need temporal tacit in order to achieve a good score. Moreover, modeling the temporal association between left and right fingers can help summary the typical operating patterns for different games, benefiting the exploration of the bots.

## 4.2 Model Architecture

The architecture of the proposed FingFormer is presented in the right part of Fig. 1, which takes the finger operation graph as input and generates the graph embedding, reflecting a global view of the record. FingFormer consists of two encoding modules to model the contextually operating information from both hands. In the first module, two separate *Graph-based Encoding* modules take each single-hand operation graph as input, and learns the the embedding by estimating the spatial relatedness among finger movements. After that, a *Left-Right Interaction* module integrates both left-hand and right-hand embeddings and studies their interaction from the temporal perspective.

*4.2.1 Graph-based Encoding Module.* The Graph-based Encoding module $\Phi(\cdot)$ consists of two blocks $\Phi^{(l)}(\cdot)$ and $\Phi^{(r)}(\cdot)$, which process left-hand and right-hand graphs through a multi-layer graph-based Transformer encoders, separately. Since the two blocks are counterparts, we here detail the architecture of left block $\Phi^{(l)}(\cdot)$.

Encoding block $\Phi^{(l)}$ takes the left-hand graph $\mathcal{G}^{(l)}$ as input. In detail, for each node $v_i^{(l)}$ with attributes $\boldsymbol{a}_i^{(l)} = (i, \gamma_i^{(l)}, d_{i,i+1}^{(l)}, \alpha_i^{(l)})$, we first map each attribute value into continuous vector space and obtain the attribute embedding $\boldsymbol{X}_i^{(l)}$ via the follows aggregating function:

$$\boldsymbol{X}_i^{(l)} = \boldsymbol{E}_\gamma(\gamma_i^{(l)}) + \boldsymbol{E}_d(d_{i,i+1}^{(l)}) + \boldsymbol{E}_\alpha(\alpha_i^{(l)}) + \boldsymbol{E}_{pos}(i), \tag{5}$$

where $\boldsymbol{E}_\gamma$, $\boldsymbol{E}_d$, $\boldsymbol{E}_d$, $\boldsymbol{E}_\alpha$, $\boldsymbol{E}_{pos}$ are learnable embedding matrices shared between modules $\Phi^{(l)}$ and $\Phi^{(r)}$, $\boldsymbol{E}_{pos}$ is the positional embedding matrix [29] which preserves the order of finger points in the left hand.

After that, a multi-layer Graph-based Transform-style encoder is adopted to capture the finger movement. In detail, to discover the typical finger movement patterns in each single hand, we introduce a spatial-biased multi-head self-attention $SpatialBiasedAttn(\cdot)$ to study the spatial correlation between different recorded points, where $i^{th}$ self-attention head is:

$$\boldsymbol{A}^{(i)} = \boldsymbol{Q}^{(i)}\boldsymbol{K}^{(i)\top}/\sqrt{d_k} + \boldsymbol{B}_\Phi^{(l)}, \tag{6}$$

$$\boldsymbol{Q}^{(i)} = \boldsymbol{H}\boldsymbol{W}_i^Q, \quad \boldsymbol{K}^{(i)} = \boldsymbol{H}\boldsymbol{W}_i^K, \quad \boldsymbol{V}^{(i)} = \boldsymbol{H}\boldsymbol{W}_i^V, \tag{7}$$

where $\boldsymbol{H}$ is the input of each attention layer and the attribute embedding $\boldsymbol{X}$ is the input of the first layer. $\boldsymbol{W}_i^V$, $\boldsymbol{W}_i^Q$ and $\boldsymbol{W}_i^K$ are the learnable matrices. The offset matrix $\boldsymbol{B}_\Phi^{(l)}$ is the intra-edge embedding matrix corresponding to edge attributes. In particular, for each intra-edge $(v_i^{(l)}, v_j^{(l)})$, its embedding is:

$$\boldsymbol{B}_\Phi^{(l)}(i,j) = \boldsymbol{E}_{intra}(d_{i,j}^{(l)}), \tag{8}$$

where $\boldsymbol{E}_{intra}$ is a learnable embedding matrix shared between $\Phi^{(l)}$ and $\Phi^{(r)}$, which incorporates the spatial correlations among the finger movements into embedding learning. Finally, the Graph-based Encoding module generates the left-hand embedding as:

$$\boldsymbol{Y}^{(l)} = \Phi^{(l)}(\mathcal{G}^{(l)}). \tag{9}$$

*4.2.2 Left-Right Interaction Module.* The previous Graph-based Encoding module can only capture the finger movements of each hand, however, in real gaming, the left-hand and right-hand movements are highly correlated in time. For instance, a game may require the player to drag with his left hand to control the direction so that he faces the enemy, while clicking with his right hand to release the skill. Therefore, we introduce another Left-Right Interaction module $\Psi(\boldsymbol{Y}^{(l)}, \boldsymbol{Y}^{(r)})$ to estimate the temporal associations between the operations of both hands. It contains the multi-layer temporal-biased multi-head attention:

$$TempBiasedAttn(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{B}_\Psi). \tag{10}$$

The calculation of Eq. (10) is similar to Eq. (6) and Eq. (7), but the input matrix $\boldsymbol{H}$ in Eq. (7) is replaced with $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$, respectively. $\boldsymbol{B}_\Psi$ is the inter-edge embedding matrix, which reveals temporal relatedness between left and right operations. For each edge $(v_i^{(l)}, v_j^{(r)})$, its embedding is calculated as:

$$\boldsymbol{B}_\Psi(i,j) = \boldsymbol{E}_{inter}(\Delta t_{i,j}^{(l,r)}), \tag{11}$$

where $\boldsymbol{E}_{inter}$ is a learnable embedding matrix, $\Delta t_{i,j}^{(l,r)}$ is the attribute value of inter-edge $(v_i^{(l)}, v_j^{(r)})$, indicating the interval time between the two points. Similar to Eq. (6), The inter-edge embedding $\boldsymbol{B}_\Psi$ is adopted as a bias in the attention calculation for each layer.

To fully model the interaction between hands, as shown in Fig. 1, in each layer, we adopt a pair of symmetry temporal-biased multi-head attention to jointly learn the left-right association as:

$$\boldsymbol{M}^{(l)} = TempBiasedAttn^{(l)}(\boldsymbol{H}^{(l)}, \boldsymbol{H}^{(r)}, \boldsymbol{H}^{(r)}, \boldsymbol{B}_\Psi), \tag{12}$$

$$\boldsymbol{M}^{(r)} = TempBiasedAttn^{(r)}(\boldsymbol{H}^{(r)}, \boldsymbol{H}^{(l)}, \boldsymbol{H}^{(l)}, -\boldsymbol{B}_\Psi^\top). \tag{13}$$

After that, a skip connection is adopted to send the interaction information as well as the single hand operation patterns to the next layer:

$$\boldsymbol{H}^{(l)} \leftarrow FFN^{(l)}(\boldsymbol{H}^{(l)} \oplus \boldsymbol{M}^{(l)}), \tag{14}$$

$$\boldsymbol{H}^{(r)} \leftarrow FFN^{(r)}(\boldsymbol{H}^{(r)} \oplus \boldsymbol{M}^{(r)}), \tag{15}$$

where $FFN(\cdot)$ is a feed-forward network, $\oplus$ denotes a concatenating operation. $\boldsymbol{H}^{(l)}$ is the left-hand input for each attention layer, and the input of the first layer is $\boldsymbol{Y}^{(l)}$. The last layer of the temporal-biased attention output the embedding matrix of left-hand and right-hand operating embeddings $\boldsymbol{Z}^{(l)}$ and $\boldsymbol{Z}^{(r)}$. Finally, we obtain the whole graph representation $\boldsymbol{e}$ by an aggregating layer, which summaries the information from all the nodes in the graph as:

$$\boldsymbol{e} = Softmax(FFN(\boldsymbol{Z}))^\top \boldsymbol{Z}, \quad \boldsymbol{Z} = \boldsymbol{Z}^{(l)} \oplus \boldsymbol{Z}^{(r)}. \tag{16}$$

The graph embedding $\boldsymbol{e}$ will be used for the model optimization in Section 4.3 as well as finding the bots.

*4.2.3 Complexity Analysis.* The time complexity consists of two parts: the preprocess stage and the FingFormer model. The preprocess stage takes a screen senor record as input and generates the corresponding finger operation graph. Its time complexity is about $O(n^2)$, where $n$ is the number of points in a record. The FingFormer model contains two modules. For the Graph-based Encoding module, it consists of two blocks for different hands, and the time complexity of encoding the left-hand graph is $O(L_1 n_l^2 d)$, where $n_l$ is the number of nodes, $L_1$ and $d$ are the number of layers and hidden dimension, respectively. The time complexity of the Left-Right Interaction module is $O(2n_l n_r d L_2)$, where $L_2$ is the number of layers in the module. Therefore, the total time complexity of FingFormer module is $O(d(n_l^2 + n_r^2)L_1 + 2dn_l n_r L_2)$, which is much lower than the traditional Transformer models which concatenate the left-hand record and right-hand record as input.

## 4.3 Contrastive Learning Strategy

Due to the difficulty of obtaining sufficient cheating samples, an unsupervised learning strategy is urgently encouraged to distinguish the bots from the honest players. Typical unsupervised learning strategies, such as to reconstruct the input, however, usually fail to deal with the screen sensor data. Since the input operation sequence is expected to be fully decoded from the learned embedding, the reconstruction-based learning is very sensitive to the noise. For example, a human player is usually easy to make a touch by mistake on the screen, so that the reconstruction-based learning is not applicable to learn the real operating patterns. To handle this problem, we introduce two contrastive learning strategies for training mobile game sensor data, which learn the finger movements and a player's operating habits, respectively.

*4.3.1 Finger Movement Contrastive Learning.* Since the goal of most bots is to maximize the benefit in game, many cheating programs will display the unusual finger movement patterns in some specific scenarios. For example, when trying to get close to a boss, most human players tend to move around slowly and carefully, trying to find its weakness. However, the mature bots usually directly rush to the boss according to a established track and release the skills. The different behaviors between human players and bots will reflect in their finger movements. Therefore, our first objective is to capture the finger moving patterns by the contrastive strategy. Specifically, inspired by the sampling strategies in [7, 18, 27], for each mobile game record $r_i = \{\tau_i, u\}$ from player $u$, we down-sample another sequence $\tau_{i,\rho}$ from the finger moving sequence $\tau_i$ with probability $\rho$ as:

$$\tau_{i,\rho} = DownSample(\tau_i, \rho). \tag{17}$$

In practice, we set the sampling probability $\rho$ as 0.8. The sampled sequence $\tau_{i,\rho}$ can be regarded as the positive sample of the original sequence, which gives a vague description of the original moving patterns. Then we generate another negative sample $\tau_{z,\rho}$, which is obtained via down-sampling from another random record $r_z$. Thus, the objective of learning the finger movement is to distinguish the positive sample from the negative sample via:

$$\ell_1(r_i) = -log\sigma\Big(\mathcal{D}_1\big(\boldsymbol{e}_i, \boldsymbol{e}_{i,\rho}\big)\Big) - log\sigma\Big(1 - \mathcal{D}_1\big(\boldsymbol{e}_i, \boldsymbol{e}_{z,\rho}\big)\Big), \tag{18}$$

where $\boldsymbol{e}$ is the learned embedding from the FingFormer model. The function $\mathcal{D}_1(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{W}_1 \boldsymbol{y} + \boldsymbol{b}_1$ is a bilinear function, where $\boldsymbol{W}_1$ and $\boldsymbol{b}_1$ are the learnable parameters.

*4.3.2 Operation Habit Learning.* However, the first contrastive strategy cannot covers all the bots, as some "clever" bots can still be disguised as a human player by performing some redundant/random finger movements. We thus require another strategy to enhance the discrimination, which we consider to model the player's operation habits.

In detail, we have observed that a player usually has his/her own operat ion habits, such as the collaborative mode between hands when releasing a combo[4]. Despite the operations of the same player can be diverse for different mobile games, his/her operation habits should retain in different games. Also, it is intuitive that different players usually do not have exactly the same habits. However, for a bot software, it is always sold to as many people as possible for the huge profits. In that case, the cheating players who use the same type of bots will behave similar or exactly the same operation habits. Thus, for the operation records $\tau_i$ and $\tau_j$ from the same player $u$, we randomly select an operation record $\tau_k$ from another player $v$, and introduce the following objective:

$$\ell_2(r_i) = -log\sigma\Big(\mathcal{D}_2\big(\boldsymbol{e}_i, \boldsymbol{e}_j\big)\Big) - log\sigma\Big(1 - \mathcal{D}_2\big(\boldsymbol{e}_i, \boldsymbol{e}_k\big)\Big), \tag{19}$$

The goal of the above objective is to refine the operation habits of a player $u$ by distinguishing his/her habits from another players $v$.

*4.3.3 Objective.* The total objective function thus can be defined as the weighted sum of the two contrastive strategies:

$$\mathcal{L} = \lambda_1 \sum_{r_i \in \mathcal{R}} \ell_1(r_i) + \lambda_2 \sum_{r_i \in \mathcal{R}} \ell_2(r_i) \tag{20}$$

where $\lambda_1$ and $\lambda_2$ are the hyper-parameters to balance the two parts.

## 5 EXPERIMENT

To validate the effectiveness of the proposed model, we conduct extensive experiments in different environments. All the experimental data and environment below are provided by NetEase Games[5].

## 5.1 Experiment Settings

we first detail the datasets, evaluating metrics, and baseline models used in the experimental studies.

*5.1.1 Dataset.* The data can be roughly divided into two parts: a **training dataset** for pre-training the models, and **three test datasets** to verify the performance. The training dataset is arranged as follow: we randomly sample about 100,000 records of a scene $A$ in an online mobile game $Game_0$[6] in a day. The three test datasets are organized as follow:

- **Offline Test Data.** The offline test data contains two parts. The first part is 3,000 labeled records of scene $A$ in $Game_0$, where the normal records and cheating records are both 1,500. Also, to evaluate the generality to different scenes, there are 3,000 labeled samples from another scene $B$ in

[4]A combo refers to a sequence of actions (usually controlled by the left hand) or skills (usually released by the right hand) in a limited time.

[5]http://game.163.com

[6]All the names of games/scenes have been anonymized.

**Table 2: The clustering performance on the synthetic dataset. (bold: best; underline: runner-up.)**

| | traj2vec | Traj2SimVec | Transformer | t2vec | FingFormer | Improvement |
|---|---|---|---|---|---|---|
| *NMI* | 0.005 | 0.117 | 0.419 | <u>0.758</u> | **0.830** | 9.50% |
| *ARI* | -0.008 | 0.027 | 0.387 | <u>0.721</u> | **0.852** | 18.17% |

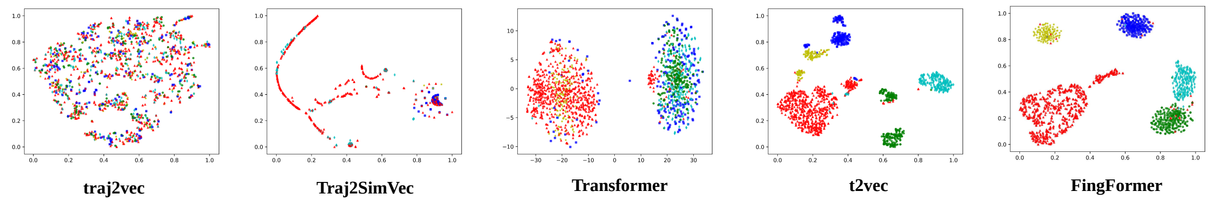

**Figure 2: Visualization of the learned embeddings in the synthetic dataset. The red points denote the normal records, while the other four colors are the cheating records generated by four different bots.**

*Game*$_0$, where the normal and cheating records are also both 1,500.

- **Synthetic Test Data.** We obtained four different game bots before. We use each bot to generate 300 gaming records in the scene $A$ , together with the 1,500 normal records to combine the synthetic test dataset.
- **Online Test Data.** We capture a small daily data flow of three online games from the server (*i.e.*, *Game*$_1$, *Game*$_2$ and *Game*$_3$). It contains about a million operation records, and the proportion of the three games is about 1:3:20. For each game, we require a detection model to infer the top-1% records which are probably to be the bots. Then, the experts will label the predicted records and evaluate the performance.

*5.1.2 Evaluation Methodology.* In the following experiments, we adopt *ARI* [33] and *NMI* [12] as clustering metrics, *Precision* and *F1*-score as classification metrics to evaluate the performance in different experimental environments. *Precision* is only used in the online experiment.

*5.1.3 Baseline.* Since there is no previous work focusing on this problem, we in this paper primarily compare with the state-of-the-art approaches that models the sequential data:

- **traj2vec** [32] focuses on trajectory clustering. It first extracts a feature sequence through a sliding window, then employs a sequence-to-sequence auto-encoder to learn trajectory representations.
- **Traj2SimVec** [35] focuses on the trajectory similarity computation. It simplifies the trajectories for indexing to obtain triplet training samples efficiently, and incorporates sub-trajectory similarity information into embedding learning.
- **t2vec** [15] also focuses on trajectory similarity computation. It introduces a spatial proximity-aware loss function and a de-noise sequence-to-sequence based model to recover the original trajectory.
- **Transformer** [29] is an effective attention-based approach to model the sequential data, which has achieved the-state-of-the-art performance in various tasks in natural language processing [2, 9, 19], computer vision [3, 6, 10], *etc.*

*5.1.4 Experiment Settings.* We need to study both the left-hand and right-hand operation sequences, thus for the models except Transformer, we train two sub-models, each studies the operations of a hand, while the output of two sub-models are concatenated as the final representation of the input record. For Transformer, following [9], we concatenate left-hand sequence and right-hand sequence separated by a special token [SEP]. The input of Transformer is the same as our FingFormer, and we also adopt our contrastive learning strategy to pre-train Transformer.

For our FingFormer, the hidden dimension $d$ is set as 128, the numbers of heads in temporal-biased attention and spatial-biased attention are both set as 8. Based on the statistics of operation records, we empirically set the maximum number of left-hand points $n_l$ as 270 and right-hand points $n_r$ as 130, respectively. The numbers of encoder layers $L_1$ and $L_2$ are both set to 2. We put equal weight on the two contrastive strategies in Eq. (20), and set $\lambda_1$=$\lambda_2$=0.5.

## 5.2 Synthetic Environment Comparison

To verify the effectiveness, we first compare the models in an ideal synthetic environment. The synthetic dataset contains 1,500 normal records and 1,200 cheating records generated from four different bot programs (*i.e.*, each generates 300 records). Each model is asked to generate the embeddings for the records and a classical clustering model (KMeans) [16] is then adopted to group the learned embeddings into five categories. For an effective model, it should not only distinguish the normal records from the cheating records, but should also identify different types of game bots. Therefore, an effective model should achieve good clustering performance in this experiment. The results of *NMI* and *ARI* are presented in Table 2. To present a more intuitive comparison, we use t-SNE [28] to transform the representations into a 2-dimensional vector space and plot the results in Fig. 2.

From the results, we can observe that the proposed FingFormer achieves the best performance, with an improvement of 9.50% on *NMI* and even 18.17% on *ARI* compared with the best-performing baselines. Also, from the visualization in Fig. 2, we observe that FingFormer makes each class tightly cluster together, as well as identifies obvious boundaries between different clusters. For the baseline models, methods traj2vec and Traj2SimVec perform poorly on this dataset. For example, their *NMI* and *ARI* score are nearly zero, and the learned embeddings are entangled severely. We also find that t2vec achieves the satisfactory performance on this synthetic dataset, and performs better than other baselines. This mainly

**Table 3: The performance of *ARI, NMI, F1* on the offline dataset (bold: best; underline: runner-up.)**

| | scene A | | | scene B | | |
|---|---|---|---|---|---|---|
| | *NMI* | *ARI* | *F*1 | *NMI* | *ARI* | *F*1 |
| traj2vec | 0.001 | -0.001 | 0.663 | 0.001 | 0.000 | 0.665 |
| Traj2SimVec | 0.026 | -0.021 | 0.580 | 0.009 | -0.001 | 0.680 |
| Transformer | 0.515 | 0.603 | 0.886 | 0.122 | 0.074 | 0.629 |
| t2vec | <u>0.546</u> | <u>0.621</u> | <u>0.901</u> | <u>0.429</u> | <u>0.515</u> | <u>0.873</u> |
| FingFormer | **0.690** | **0.778** | **0.942** | **0.593** | **0.701** | **0.924** |
| Improvement | 26.37% | 25.28% | 4.55% | 38.23% | 36.12% | 5.84% |

owing to its down-sampling strategies for de-noising, which enhance its robustness in faced with different types of records. While for Transformer, we find that it shows fair performance in this experiment. As shown in Fig. 2, its learned embeddings can extract some basic characteristics of the same class, still, the different classes are entangled severely and nearly indistinguishable. The performance of Transformer indicates that the attention mechanism cannot directly capture the distinctive operation patterns for distinguishing the bots. On the opposite, owing to the graph-based modeling of finger movement and interaction, our FingFormer outperforms all baseline models to a large extent.

## 5.3 Offline Evaluation

We also conduct a set of experiment on the real mobile game data to verify the effectiveness and generality. As described in previous Section 5.1.1, we compare the performance on two scenes (*A* and *B*) of the same game $Game_0$, each scene contains 1,500 normal records and 1,500 cheating records. For each scene, the learned embeddings will be grouped into two categories via KMeans. After that, we calculate both the classification accuracy (*F1*) and the clustering metrics (*NMI*, *ARI*) to evaluate the performance. The results are presented in Table 3.

As the pre-training data are also collected from scene *A*, we observe that most models (except traj2vec and Traj2Simvec) achieve better performance in scene *A* than in scene *B*. Transformer shows comparable performance with the best-performing baseline t2vec in scene *A*, but its effect decreases significantly in scene *B*, indicating the poor generality of Transformer in the real-world mobile game bot detection problem. For our FingFormer, it is consistently superior than all baselines, for example, we gain the improvement of over 25% on NMI and ARI, and 4.55% on F1 score in scene *A*. Also, FingFormer displays good robustness and generality in Scene *B*, with over 35% improvement on NMI and ARI and, as well as satisfactory F1 score.

In conclusion, the results of the offline dataset indicates that our FingFormer is both effective and general to detect the bots in different scenes.

## 5.4 Online Evaluation

To investigate the effectiveness of our proposed FingFormer in the real applications, we conduct a set of online experiments. From the previous experiments, we regard t2vec as a competitor of FingFormer, thus we deploy the two models in the online environment and compare their abilities of finding bots from the data flow of three different online mobile games. In detail, for each online game, the DBSCAN [11] algorithm is adopted to assign the learned embeddings into several clusters since it is not clear how many different kinds of anomalies there are. We regard the records in the biggest cluster as the normal records, while the records in other clusters as the potential bots. Then, each model calculates the center of each suspicious cluster, and infers top-1% possible cheating records according to the embedding distance to the cluster centers. Finally, the experts will label these records and calculate *Precision*. The results are presented in Table 4.

**Table 4: Precision comparisons of the online data.**

| | $Game_1$ | $Game_2$ | $Game_3$ |
|---|---|---|---|
| t2vec | 0 | 0.0297 | 0.108 |
| FingFormer | 0.0392 | 0.0594 | 0.204 |
| Improvement | - | 50.00% | 47.06% |

We can observe that the two models both gain higher precision in $Game_3$. This is mainly because the popularity and the number of active players of $Game_3$ is much larger than the other two games, so there will be more potential cheating players in $Game_3$. Moreover, we can observe that our FingFormer outperforms t2vec to a large extent. In particular, FingFormer has the improvement of about 50% on both $Game_2$ and $Game_3$. In addition, t2vec cannot find any bots from $Game_1$, while our FingFormer still can infer several cheating samples from the massive online data and achieve the precision of 0.0392. Most notably, despite the precision of FingFormer on $Game_1$ and $Game_2$ is not high, it is still of great significance for the practical bot detection application. It means that experts can use FingFormer to quickly find the samples of the new/latest bots from the massive data flow, and design corresponding strategies to avoid the loss of the game company. Therefore, the online experiment results additionally verify the superiority of the proposed FingFormer model for the online bot detection system compared with the baseline.

## 6 CONCLUSION

We have introduced a Graph-based Transformer-style model, FingFormer, to tackle the exposed issues for the unsupervised mobile game bot detection. We process the sensor data into the form of a finger operation graph to explicitly model the contextual temporal-spatial operation information between both hands. Moreover, to embed the graph-based operation data into continuous vector space, we introduce a novel graph-based encoding module and a left-right interaction module. Finally, we propose two types of contrastive learning strategies to effectively refine the distinctive embeddings. Multi-faceted experimental results show that the proposed technique significantly boosts the mobile game bot detection systems.

## ACKNOWLEDGMENTS

This work has been supported by the National Natural Science Foundation of China under Grant No.: 62077044, 61702470, 62002343.

## REFERENCES

[1] 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 2017, Amantea, Italy, June 19-22, 2017.* 6:1–6:11.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-End Object Detection with Transformers. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, Vol. 12346. 213–229.

[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 (2009).

[5] Kuan-Ta Chen, Andrew Liao, Hsing-Kuo Kenneth Pao, and Hao-Hua Chu. 2008. Game Bot Detection Based on Avatar Trajectory. In *Entertainment Computing - ICEC 2008, 7th International Conference, Pittsburgh, PA, USA, September 25-27, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5309).* 94–105.

[6] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020. Generative Pretraining From Pixels. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Vol. 119. 1691–1703.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event.*

[8] 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. *J. Supercomput.* 72, 9 (2016), 3572–3587.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers).* 4171–4186.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021.*

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. 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), Portland, Oregon, USA*, Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad (Eds.). 226–231.

[12] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. 2009. Normalized mutual information feature selection. *IEEE Transactions on neural networks* 20, 2 (2009), 189–201.

[13] Frauke Kreuter, Georg-Christoph Haas, Florian Keusch, Sebastian Bähr, and Mark Trappmann. 2020. Collecting survey and smartphone sensor data with an app: Opportunities and challenges around privacy and informed consent. *Social Science Computer Review* 38, 5 (2020), 533–549.

[14] Eunjo Lee, Jiyoung Woo, Hyoungshick Kim, and Huy Kang Kim. 2018. No Silk Road for Online Gamers!: Using Social Network Analysis to Unveil Black Markets in Online Games. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018.* 1825–1834.

[15] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018.* 617–628.

[16] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[17] Jehwan Oh, Zoheb Hassan Borbora, Dhruv Sharma, and Jaideep Srivastava. 2013. Bot Detection Based on Social Interactions in MMORPGs. In *International Conference on Social Computing, SocialCom 2013, SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013.* 536–543.

[18] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020.* 1150–1160.

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67.

[20] Yuri Seo, Rebecca Dolan, and Margo Buchanan-Oliver. 2019. Playing games: advancing research on online and mobile gaming consumption. *Internet Res.* 29, 2 (2019), 289–292.

[21] Jianrong Tao, Jianshi Lin, Shize Zhang, Sha Zhao, Runze Wu, Changjie Fan, and Peng Cui. 2019. MVAN: Multi-view Attention Networks for Real Money Trading Detection in Online Games. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.* 2536–2546.

[22] 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 & Data Mining, KDD 2018, London, UK, August 19-23, 2018.* 811–820.

[23] Ruck Thawonmas, Yoshitaka Kashifuji, and Kuan-Ta Chen. 2008. Detection of MMORPG bots based on behavior analysis. In *Proceedings of the International Conference on Advances in Computer Entertainment Technology, ACE 2008, Yokohama, Japan, December 3-5, 2008*, Vol. 352. 91–94.

[24] Yuan Tian, Eric Y. Chen, Xiaojun Ma, Shuo Chen, Xiao Wang, and Patrick Tague. 2016. Swords and shields: a study of mobile game hacks and existing defenses. In *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016.* 386–397.

[25] Yonglong Tian, Dilip Krishnan, and Phillip Isola. 2020. Contrastive Multiview Coding. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XI (Lecture Notes in Computer Science, Vol. 12356).* 776–794.

[26] Michail Tsikerdekis, Sean Barret, Raleigh Hansen, Matthew Klein, Josh Orritt, and Jason Whitmore. 2020. Efficient Deep Learning Bot Detection in Games Using Time Windows and Long Short-Term Memory (LSTM). *IEEE Access* 8 (2020), 195763–195771.

[27] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018).

[28] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[29] 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 *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA.* 5998–6008.

[30] Shing Ki Wong and Siu-Ming Yiu. 2019. Detection on auto clickers in mobile games. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 10, 3 (2019), 65–80.

[31] 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 (2020), 65:1–65:24.

[32] Di Yao, Chao Zhang, Zhihua Zhu, Jian-Hui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017.* 3880–3887.

[33] Ka Yee Yeung and Walter L Ruzzo. 2001. An empirical study on principal component analysis for clustering gene expression data. *Bioinformatics* 17, 9 (2001), 763–774.

[34] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Bad for Graph Representation? *CoRR* abs/2106.05234 (2021).

[35] Hanyuan Zhang, Xinyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020.* 3209–3215.