# TS-Finder: Privacy Enhanced Web Crawler Detection Model Using Temporal-Spatial Access Behaviors

Jing Zhao[1], Rui Chen[1], Pengcheng Fan[1]

[1*]School of Software Technology, Dalian University of Technology, Dalian, 116024, Liaoning, China.

Contributing authors: zhaoj9988@dlut.edu.cn; 72117004@mail.dlut.edu.cn; fanpc@mail.dlut.edu.cn;

**Abstract**

Web crawler detection is critical for preventing unauthorized extraction of valuable information from websites. Current methods rely on heuristics, leading to time-consuming processes and inability to detect novel crawlers. Privacy protection and communication burdens during training are overlooked, resulting in potential privacy leaks. To address these issues, we propose a federated deep learning crawler detection model that analyzes access behaviors while preserving privacy. First, individual clients locally host website data, while the central server aggregates information for detection model parameters, eliminating raw user data transmission or access. We then develop an innovative algorithm constructing access path trees from user logs, effectively extracting temporal and spatial behavior features. Additionally, we propose a novel time series model with fused additive attention, enabling effective web crawler detection while preserving privacy and reducing data transmission. Finally, comprehensive evaluations on public datasets demonstrate robust privacy protection and effective detection of emerging crawler types.

**Keywords:** Web crawler detection, federated deep learning, temporal-spatial behavior, access path tree, privacy protection

## 1 INTRODUCTION

Web crawling involves the automated retrieval of target website resources by simulating user requests, without human intervention. Excessive web crawling can occupy normal website bandwidth, degrading quality of service (QoS) [31]. With advancing technology, crawler diversity has increased, enabling usage across Internet industries like search engines and public opinion mining [38]. Concurrently, means of leveraging crawlers to gather information continue evolving. Crawlers often evade detection, illicitly accessing diverse websites, enabling data theft and illegal access. Competitor misuse of illicitly gathered web data can induce major financial losses. Thus, detecting crawlers is an urgent priority. For instance, e-commerce leaders like Weee Inc. and Alibaba employ crawler detection to impede malicious access, securing websites and improving user experience [16, 29].

Although web crawler detection studies have been extensively studied in today's network security field, current crawler detection still faces three major challenges.

The first challenge is that existing crawler detection models rely heavily on security experts to manually create detection rules[18, 28, 36]. These rules mainly look at how different parts of a website are accessed to identify crawlers. Although this approach has worked well in experiments, it requires a lot of human effort and cannot easily detect new crawlers. The second challenge is that servers send local user data to a central training server. This creates significant bandwidth pressure during transmission, heavily occupying precious network resources for crawler training[12, 18, 20, 28, 36, 37]. For example, many website requests during model training can inconvenience normal users accessing the server. The third challenge is that existing models directly use original data containing private user information for training[11, 36]. For example, analyzing user locations to detect crawler geography can lead to sensitive data like phone numbers and device IDs being transmitted. This risks privacy data leakage issues.

To address those challenges currently faced by web crawler detection, we propose a federated deep learning[23] crawler detection model integrated with privacy protection mechanisms, named TS-Finder. To our knowledge, this is the first work to incorporate the concept of privacy protection into the field of crawler detection, as shown in Fig. 1. Initially, the model innovatively adopts a strategy to train with local data on-site, completely abandoning the traditional practice of sending data to a central server for training. The role of the central server is transformed to only aggregate weight parameters uploaded from the local models, significantly reducing the risk of privacy data leakage. Furthermore, we have designed a novel dataset processing algorithm that refines user behavior characteristics by conducting an in-depth analysis of users' access time distribution and request addresses. This enhancement significantly increases the accuracy of the crawler detection model, enabling effective identification of new types of crawling behavior. In the meantime, we have optimized the time series model[39] by incorporating the output of each time point in the time series into the calculation of the hidden layers, thereby greatly minimizing the loss of information during the model training process. Finally, experimental validation on two public datasets[18, 25] demonstrated that the detection model proposed in this study achieved an F1 score of 92.28%. This result not only proves that the model is effective in detecting web crawlers but also efficiently secures user privacy.

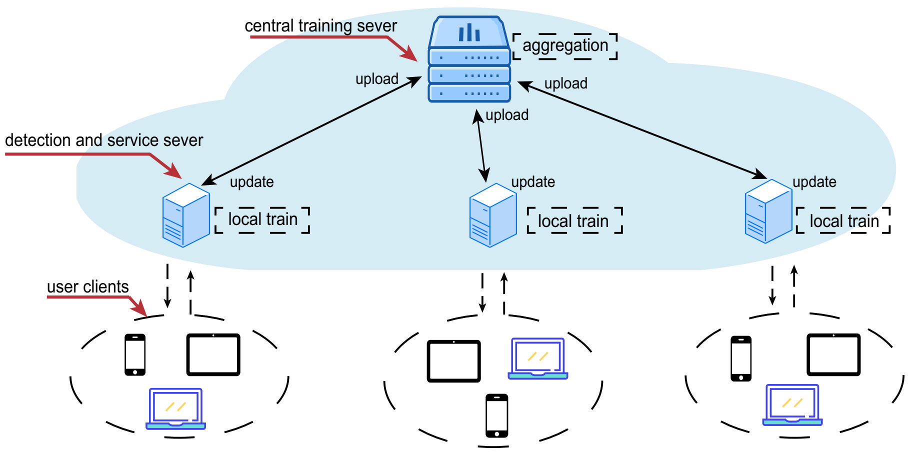


**Fig. 1** An example of crawler detection with hierarchical federated learning

Above all, we are the major contribution of this work is as follows:

- We propose a new web crawler detection model, TS-Finder, which is the first to utilize user behavior analysis and privacy-preserving federated deep learning for web crawler detection. Furthermore, we propose a user behavior analysis algorithm that extracts behavior characteristics by creating a user access path tree based on the user's access addresses and time series.
- We also improve the time-series behavior detection model by extracting access time-series behavior in time steps and using an additive attention mechanism to combine the output of each time step in the time-series model with the final hidden layer, which allows the model to capture more behavior information and achieve higher detection accuracy.
- To train our proposed model, we use a federated learning algorithm, which reduces the communication pressure between clients and the central server. We conduct experiments and validation on two public datasets [18, 25] and compare our model with the state of the art models. The experimental results show that our model achieves F1 scores of 92.28% and 90.99%, respectively.

The remainder of this paper is organized as follows. Section II provides a summary of related work and research background. In Section III, we provide a detailed description of the overall system design of the detection model, the algorithm for processing the dataset based on user behavior, and the detection model implementation and federated deep learning model training process. Section IV presents the results of comparative experiments on the detection models. Finally, in section V, we provide a conclusion to our work.

## 2 RELATED WORK

Web crawlers have had a profound impact on commercial data security, service resource consumption, and business decision-making. Their role in the online environment is now formidable. In particular, malicious web crawlers pose severe challenges to web services. To address this, academia has proposed various techniques for web crawler analysis and detection. In this section, we will discuss the progress of current crawler detection technologies and analyze the gap between them and the challenges they currently face, as shown in Table 1.

**Table 1** Comparison of Related Works on Crawler Detection Model

| **Work** | **Detection Techniques** | **Emerging(Zero-Day) Attack** | **Communication Overhead** | **Behavior Analysis** | **Privacy Protection** |
|---|---|---|---|---|---|
| Lu [22] | H-Markov | | | | |
| Doran[10] | Statistic | | | ✓ | |
| Wan [36] | Rule-Patterns | | | ✓ | |
| Menshchikov[24] | Statistic | | | ✓ | |
| Suchacka[33] | Rule-Patterns | | | ✓ | |
| Ro[28] | Long-Tail | | | ✓ | |
| Chu [6] | Using Log | | | ✓ | |
| Rahman [27] | biostatistics | | | | |
| Li [20] | Trapping | | | ✓ | |
| Lagopoulos [18] | Content-aware | | | ✓ | |
| Li[21] | Honey Pot | | | | |
| Xia [37] | Bert | ✓ | | ✓ | |
| Acien [1] | CAPTCHA | | | | |
| Gao [12] | RL | ✓ | | | |
| **Our Work** | **FL, GRU** | ✓ | ✓ | ✓ | ✓ |

RL: Reinforcement Learning; H-Markov: Hidden Markov; GRU: Gate Recurrent Unit; FL: federated learning; RNN: Recurrent Neural Network; ICA: Independent Component Analysis; MLP: multi-layer perceptron; LSTM: long-short term memory network; Bi-LSTM: bi-directional long-short term memory networks.

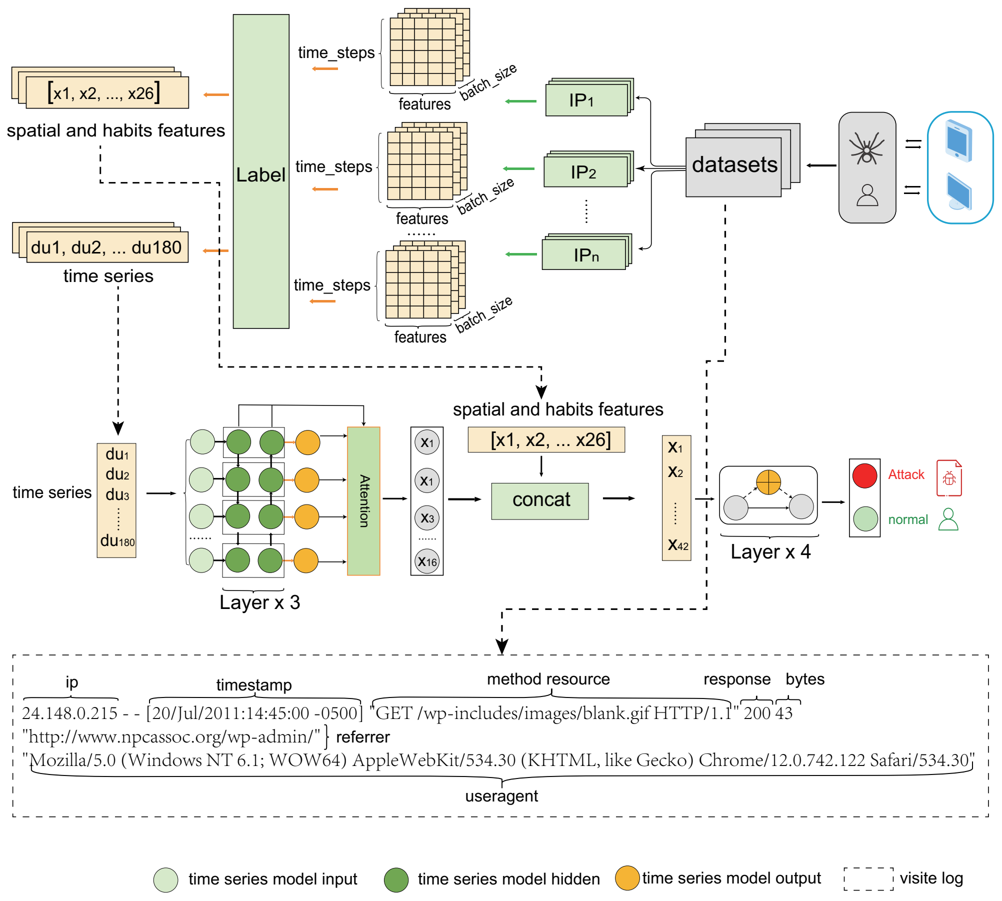
Due to the similarity between web crawlers and other web bots, the survey covers web crawlers and other web bots, such as click fraud bots targeting search engine rankings and paid advertisements and fake browsing bots targeting product recommendation systems[31, 38]. At the same time, considering that some of the work relies on commercially confidential data and cannot disclose detailed technical details or the necessary data descriptions, these deficiencies greatly limit the contribution of such work so we will exclude this part of the study.

There are currently two main technical approaches for detecting malicious web bots: feature engineering and machine learning. This section will present related work according to these two types of approaches. The research on crawlers has been based on feature engineering based on manual analysis. These works had good accuracy and low computational resource consumption for detecting traditional typical crawlers, aim to analyze and mine web bot features and artificially process the original basic features to build a combination of features to detect web bots. For example, Li et al. propose a honeypot system called "honeysites", which was deployed in 100 nodes around the world, recorded and analyzed a large amount of web crawler traffic data, and made a more comprehensive investigation of the behavior patterns and basic features of crawlers [21]. Ro et al. found that the access frequency of normal users decreased with time and proposed a crawler detection method called Long-tail Threshold Model (LTM) [28].

Similarly, some other scholars have performed statistical analysis of HTTP requests and used information such as the type of requested resources [10, 21, 24, 33] and session duration [22] to construct a probabilistic model of the session to identify web bots. In addition, some works have attempted to use biometric techniques. These works monitor the user's mouse and keyboard behavior to detection of web bots [1, 6, 27]. Still, such methods require additional modifications to the system, and the user client may also reasonably block such techniques, rendering such methods ineffective. Also, with the improvement of user privacy laws worldwide, there is no legal support for schemes that additionally capture user information. Most of these studies generally rely on human-constructed interpretable features for detection, which were widely used in practice in the early days due to their universality and interpretability. However, the construction of these features mainly relies on the experience of researchers. It uses artificial ways to determine the weight coefficients of each feature, which is less flexible and less robust. In general, with the development of crawler studies, these artificially constructed features are often easy to be circumvented. Currently, the methods based on feature engineering alone can no longer cope with the challenges posed by intelligent crawlers.

Unlike the above methods, deep learning techniques can learn from sequence-based data. Natural language processing techniques heavily utilize deep learning methods based on attention mechanisms to process sequence-based data, such as Transformers [35], BERT [8], ALBERT [19], and GPT [26]. These methods can be extended from natural language processing to other problems dealing with sequential data, such as fraudulent programs and web bot detection. Liu et al. proposed a method called local intent calibration tree-LSTM to implement the detection of fraudulent programs. While Zhang et al. proposed a model called Attribute Sequence Embedding (AS-E) for fraud detection, the model's core is an encoder-decoder based module that maps sequence-based data into an embedding vector on which a multilayer perceptron is used for classification [40]. Some sequence-based crawler detection studies use additional information, such as researchers from Alibaba who proposed a set of web crawler detection models AANet [37] using user geographic information, but this requires additional system tuning and is not universal.

The current research on web crawler detection has focused on the detection methods, while the model training methods and privacy protection issues have not received sufficient attention. In this paper, we use the federated learning (FL) training algorithm [23] to train the proposed model. Unlike standard deep learning training, the emergence of federated learning addresses the challenge of implementing traditional deep learning in privacy-sensitive scenarios. Federated learning consists of local devices and a global server, where the global server connects to multiple local devices over the network to jointly train the deep neural network model. Unlike the centralized data collection scheme in standard deep learning, the data in federated training is widely distributed across different local devices. Meanwhile, the global server only specifies



```
24.148.0.215 - - [20/Jul/2011:14:45:00 -0500] "GET /wp-includes/images/blank.gif HTTP/1.1" 200 43
"http://www.npcassoc.org/wp-admin/"
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30"
```

**Fig. 2** Design details of user behavior-based dataset processing algorithms and crawler detection models for local detection servers

the initial training model and relevant aggregation algorithms, and does not collect any training data. Furthermore, only the parameters relevant to the model are transmitted during their communication process, and the transmission of raw data and critical statistical information is prohibited. Therefore, while researching efficient web crawler detection algorithm models, privacy protection must also be considered.

## 3 PROPOSED METHOD

This section presents the proposed method and provides a detailed explanation of the implementation details of the dataset processing algorithm and the crawler detection model.

### 3.1 Crawler Detection System Framework

The proposed model system framework design is depicted in Fig. 2, which combines the user's habits, temporal, and spatial behavior of visiting sites to detect crawlers. The dataset comprises records generated by ordinary users visiting web pages, denoted as $data = \{l_1, l_2, l_3, ..., l_n\}$. Each record $l_i$ represents a log of the user's request, as illustrated in Fig. 2, and includes basic information such as referrer, request, method, resource, bytes, response, ip, useragent, and timestamp. To extract the request records of a specific user, we group the data by ip address, resulting in $gip = \{l_a, l_b, l_c, .., l_x\}$, where $l_a, l_b, l_c,$ ... , $l_x$ represent request log records generated from a single ip address. Thus, we treat the request data from a unique ip address as the request data from a distinct user.

The extracted resident time series are trained into temporal features, shown in Fig. 2, using a Gate Recurrent Unit (GRU) [9] as time series model with an additive attention mechanism [5] to obtain the intermediate results of the visited site. The temporal features are combined with the user habits and the spatial features of the sites visited by the users to form input vectors of the whitening residual network. The realization process of this model will be described in detail in the following sections.

We extract the resident time series ts and transform them into temporal features, by training them with time series model (GRU) that incorporates an additive attention mechanism. These temporal features are then combined with the user's browsing habits and spatial features of the visited sites to form input vectors for the whitening residual network. The detailed implementation process of this model will be elaborated in subsequent sections.

### 3.2 Feature Extraction and Labeling Algorithm for User Behavior Analysis

#### 3.2.1 Session Generation

A user's session refers to a continuous sequence of requests made by the user over a period of time. Our analysis reveals that the request data $gip = \{l_a, l_b, l_c, .., l_x\}$ contains at least one user session. A session contains substantial contextual information, user behavior, and insights into the user's access habits, which can effectively help us distinguish between ordinary users and crawlers. To extract the session data, we further divide the user's requests based on the request interval. Specifically, we set the duration dur between requests to 30 minutes [2]. If the duration exceeds 30 minutes, we consider it as a new session, denoted as $session = \{l_a, l_b, l_c, ..., l_s\}$, where $l_s$ is the last record of the session and the length of the session is s.

#### 3.2.2 Request and Temporal Features Extraction

In analyzing access data, we extract users' basic request characteristics and temporal features, categorizing the access into two scenarios: normal access and crawler access, for in-depth analysis. In a normal access scenario, when a regular user navigates to a resource (for example, accessing specific book information on a library website), the user's browser, upon reaching the page, will also automatically request additional dependent resources, such as the site's JavaScript scripts and CSS stylesheets. In contrast, during crawler access, the crawler's objective is typically to extract information contained within the page address, and after achieving this, it often does not request further dependent resources but moves on to the next target address.

Leveraging these behavior characteristics, we abstract these features and convert them into numerical features. In terms of resource types, we calculate the proportion of user requests for HTML, CSS, and image resources. Regarding user request responses, we analyze the ratio of 404 (Not Found) to 200 (OK) response codes in the sessions. For request methods, we tally the frequency of user requests made using HTTP methods such as POST and GET.

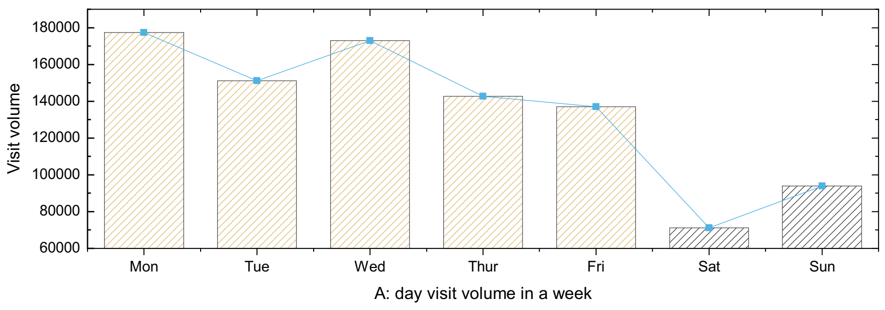
However, if the crawler is sophisticated and mimics the behavior of a standard browser, it might request these dependent resources, making the aforementioned distinctions between normal and crawler accesses insufficient for accurate detection. Therefore, we must consider additional dimensions of features, such as temporal, user request habits, and access patterns, to more accurately identify crawler activities. The specific methods for extracting these features will be detailed in the latter part of the document.

To extract temporal features, we first calculate the time interval $du_i$ between each record $l_i$ timestamp and its previous record $l_{i-1}$ timestamp using $du_i = Time_i - Time_{i-1}$. An example of this user access time calculation is shown in Fig. 4A. If $i = 1$, we set $du_1 = 0$. The computation of $du_2$ starts with $du_i = Time_i - Time_{i-1}$, resulting in a time series $ts = du_1, du_2, ..., du_s$ of length $s$ for a session. Since session lengths vary, we normalize the time series length to facilitate temporal model training. For sessions with $s < 180$, we set $du_{s+1}, ..., du_{180} = 0$. For $s > 180$, we truncate the excess part, giving a series of length 180, $ts = du_1, ..., du_{180}$. We chose 180 based on statistical analysis of our datasets [18, 25], which showed most normal users have sessions $\leq 180$, while crawlers typically exceed 180.

### 3.2.3 User Request Habits Features Extraction

We observe that user visits during weekends and late night/early morning hours differ significantly from other times and can be clearly distinguished. This pattern is consistent with the typical routine of an average user, with weekends and late night/early morning reserved for rest, while weekdays and daytime are dedicated to work. In contrast, crawlers do not follow this pattern, allowing us to differentiate normal users from crawlers based on this characteristic. Fig. 3A and Fig. 3B illustrate the distribution of user visits over a 7-day week and 24-hour day, respectively. The peaks on weekdays and during daytime hours, compared to weekends and nighttime, reflect the human routine described above. This enables distinguishing human user patterns from crawler patterns, which do not follow a weekday/weekend or daytime/nighttime cycle.

We leverage this property to extract a user's session, denoted as $session = \{l_a, l_b, l_c, ..., l_s\}$, and abstract the user's access habits into numerical features. By counting the number of logs $len_{lw}$ with timestamp attribute $l_i \in session$ that fall on weekends and knowing the size of the data volume of s in session, we can obtain the percentage of log data volume accessed on weekends as $len_{lw}/s$. Similarly, we calculate the share of log data volume accessed during rest working hours (19:00-00:00),



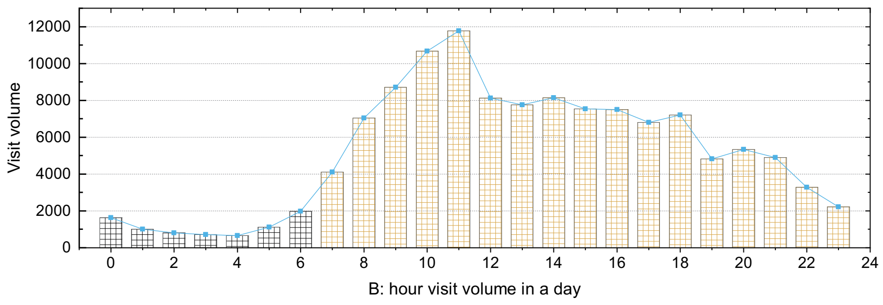


**Fig. 3** Distributed statistics of user access time, A: is the distribution of users visit within a week, B: is the distribution of users visit within 24 hours one day.

working hours (7:00-19:00), and late-night hours. These metrics can be used to derive user habit characteristics.

### 3.2.4 Spatial Features Extraction

The information provided by a website typically exhibits a hierarchical tree structure, and user access generally follows this structured pattern. In contrast, crawlers, which aim to harvest website information, may overlook or interact with the website's hierarchy in a manner different from that of regular users. We use this distinction in access patterns based on the website's hierarchy as a feature to differentiate between normal users and crawlers and define it as a spatial feature.

We utilize the timestamps and resource addresses from user access logs to construct a access path tree, from which we derive spatial-temporal behavioral features. This approach provides a comprehensive representation of user behavior by incorporating both temporal and spatial dimensions of access patterns. From Algorithm 1, the proposed algorithm for drawing the user access path tree in this paper. We extract user access path trees from $session = \{l_a, l_b, l_c, ..., l_s\}$ session to extract the accessed resources in the user session $rsl = \{rs_a, rs_b, rs_c, ..., rs_s\}$ and the list of references $rfl = \{rf_a, rf_b, rf_c, ..., rf_s\}$.

The algorithm first checks if two input lists, $rsl$ and $rfl$, have equal length using an assert statement. It then makes copies of the lists using the copy() method. Next, it initializes an empty dictionary called $vlt$ to store visited links, and a variable $rln$ to store the length of $rsl$. A counter $si$ is initialized to 0. A while loop executes as long as

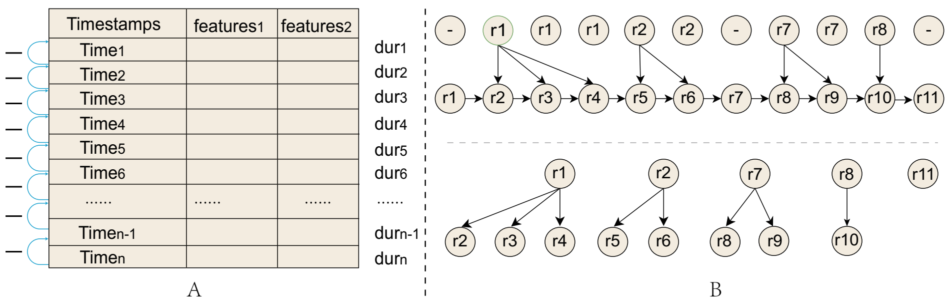


**Fig. 4** Calculation of user access page time series and construction of access path tree.

$si < rln$ and $rlc$ is not empty. Inside the loop, it checks if the current accessed resource appears in the referrer of the previous step. It creates lists $rrfl$ to store referrers containing the current resource, $rfia$ for the indices of those referrers in $rlc$, and $rrl$ for the corresponding resources in $rrfl$. A list $next\_source$ holds resources referred by $rrfl$. The loop removes the referrer and resource from $rlc$ and $rsc$. If $rlc$ is not empty, it adds the current resource to $vlt$, with value $next\_source$. $si$ is incremented by 1 each iteration. After the while loop, the function returns the remaining items in $vlt$ and $rsc$.

As an example, we consider the accessed resource list $rsl = \{r_1, r_2, r_3, ..., r_{11}\}$ and the referenced list $rfl = \{-, r_1, r_1, r_1, r_2, r_2, -, r_7, r_7, r_8, -\}$, both of which have a length of 11, as shown in Fig. 4B. We observe that the first resource accessed is $r_1$, followed by $r_2$,$r_3$, and $r_4$, resulting in a path tree with root node $r_1$ and leaf nodes $r_2$,$r_3$, and $r_4$. We can obtain trees with root nodes $r_2$,$r_7$, and $r_8$, as well as individual nodes such as $r_{11}$, using this method. Based on the generated path trees, we can abstract the behavioral space characteristics with numerical values. Specifically, we calculate the number of generated trees, the tree with the most leaf nodes (in this case, the tree with $r_1$ as the root node has the most leaf nodes with a value of 3), and the tree with only one node (in this case, $r_{11}$).

The datasets processing algorithm designed in this section. We process the user's access timestamps data to obtain access habits and temporal features, and the user's visit site page address to obtain user spatial features from raw log.

### 3.2.5 Raw Log Labeling

Prior to extracting the user's site visitation patterns and temporal-spatial behaviors within a user session, it is necessary to determine which user-agents in the HTTP request URLs belong to crawlers after segmenting the user sessions. To accomplish this, the raw log data can be analyzed to ascertain if resource requests are made to '/robots.txt.' If requests are made to the '/robots.txt' resource, the user-agent under that IP does not need labeling, and the IP can be classified as a crawler IP. This is because '/robots.txt' is a hidden web page that normal users cannot find and request.

The COUNTER [7] and BROWSCAP [3] projects provide user-agent labels and classify log datasets as either human user or crawler user-agents. Three scenarios can occur when labeling using both projects: 1) both label as crawler, 2) only one labels as crawler, or 3) BROWSCAP labels as DEFAULT BROWSER. If either project labels an agent as a crawler, we categorize it as a crawler user-agent. However, if COUNTER does not label it as a crawler but BROWSCAP labels it DEFAULT BROWSER, manual verification is required. This involves combining experience and internet searches to determine the user-agent source and confirm if it is a crawler. Finally, the labeled user-agent information is compiled into a dictionary table indicating whether the original log data belongs to a crawler or human user.

**Algorithm 1** Access path tree drawing algorithm

```
Input: rsl, rfl
Output: vlt, rsc
 1: ASSERT length(rsl) == length(rfl) AND rlc = copy(rfl) AND rsc = copy(rsl)
    AND vlt = {} AND rlen = length(rsl) AND si = 0
 2: while si < rlen AND length(rlc) > 0 do
 3:     rrfl = []
 4:     rfia = []
 5:     rrl = []
 6:     item = str(rsl[si])
 7:     for i = TO length(rlc)-1 do
 8:         referer = rlc[i]
 9:         if item ≠ '/' AND item IN referer then
10:             rrfl.append(referer)
11:             rfia.append(i)
12:             rrl.append(rsc[i])
13:         end if
14:     end for
15:     next_source = []
16:     for ri IN rfia do
17:         next_source.append(rsc[ri])
18:     end for
19:     for referrer_item, resource_item IN zip(rrfl, rrl) do
20:         rlc.remove(referrer_item)
21:         rsc.remove(resource_item)
22:     end for
23:     if length(rlc) > 0 then
24:         vlt[rsl[si]] = next_source
25:     end if
26:     si = si + 1
27: end while
28: return vlt, rsc
```

## 3.3 Detection Model Design

From Fig. 2, following the data processing described in Section 3.2, we obtained the time series of user site visits by session, denoted as $session = \{l_a, l_b, l_c, ..., l_s\}$. Specifically, this processing allowed us to derive the time series of user site visits, denoted as $ts = \{du_1, du_2, ..., du_{180}\}$, meeting the requirements for model training. Additionally, we extracted user request features, user behavior habits, and access space features, represented as $shf = \{x_1, x_2, x_3, ..., x_{26}\}$. To obtain the user's temporal behavior features, it was necessary to train the time series of user page dwell times ts through a temporal model. The model design is as follows: The Gated Recurrent Unit (GRU) addresses defects of traditional Recurrent Neural Networks (RNNs) [39] via gating mechanisms. The GRU introduces input, output, forgetting, and memory cell gates. These gates determine the importance of data and whether it should be retained or discarded, ensuring that key information is propagated. The design principles for each gating unit are as follows.

$$R_t = \theta(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \tag{1}$$

$$Z_t = \theta(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \tag{2}$$

The weight parameters $W_{xr}$ and $W_{xz}$, both belonging to $R^{d\times h}$, are used in conjunction with the bias parameters $b_r$ and $b_z$ (both belonging to $R^{1\times h}$) in the computation of the reset and update gates, respectively, as shown in Eq.1-2. The broadcast mechanism is triggered during summation, and the sigmoid function is used as the activation function to ensure that the output values fall within the interval (0, 1). Subsequently, the reset gate $R_t$ is incorporated with the conventional hidden state update mechanism to obtain the candidate hidden state $\widetilde{H_t} \in R^{n\times h}$ at time step t.

$$\widetilde{H_t} = tanh(X_t Wxh + (R_t \odot H_{t-1}) W_{hh} + b_h \tag{3}$$

The weight parameters $W_{xh} \in R^{d\times h}$ and $W_{hh} \in R^{h\times h}$ are used in conjunction with the bias item $b_h \in R^{1\times h}$, while the Hadamard product operator $\odot$ is used in Eq.3. The non-linear activation function used in the candidate hidden state, tanh, ensures that its value lies within the interval $(-1,\ 1)$. However, when training the time series of a user's resident pages, the GRU model tends to overfit when faced with lengthy sequence features and insufficient samples. To avoid over-fitting due to insufficient samples during the training process, a temporary retreat method is added to the model prior to initialization. Additive attention, which was introduced in [34], is a self-attention mechanism that is believed to be important in the task of using time series detection models. According to the authors, all hidden states are essential but not equally important, and self-attention is required to dynamically adjust the significance of different hidden states.

$$W = tanh(W_q Q + W_k K) \tag{4}$$

$$score = \sum softmax(W_w W) V \tag{5}$$

In Eq.4 and Eq.5, the query of the key is calculated in a unified dimension so that they are added in the same dimensional space, and the resulting value is subjected to the tanh activation function. The softmax[15] operation is then performed to multiply the resulting weights with the values.

Using a time series model, we transformed the time series $ts = \{du_1, du_2, ..., du_{180}\}$ into time features $tf = \{x_1, x_2, x_3, ..., x_{16}\}$, and combined it with $shf = \{x_1, x_2, x_3, ..., x_{26}\}$ to form $features = \{x_1, x_2, x_3, ..., x_{42}\}$. Given that the time series of user activity on a given page may differ in length, the incorporation of an additive attention mechanism is necessary to address the issue of variable-length input sequences. Specifically, this mechanism enables the transformation of the input time series into a fixed-size vector, which can then be combined with the user's habits and spatial access features. The resulting concatenated features $features = \{x_1, x_2, x_3, ..., x_{42}\}$ are trained using a whitening residual network to detect effectively whether the feature is a normal user or a crawler.

$$ResLayer(x) = Dropout(BN(x)) \tag{6}$$

$$ICA_i = ReLU(ResLayer(x)) \tag{7}$$

$$x_{i+1} = Dropout(ICA_i) + LayerOut_i \tag{8}$$

Eq. 6-8 demonstrate that the deep whitening residual network only requires the combination of three operations, namely batch normalization (BN), dropout, and addition, on top of a multilayer perceptron. Specifically, the addition of dropout after each BN layer and the subsequent addition of the value after dropout to the input after ReLU activation are performed. The whitening residual network used for noise reduction eliminates data redundancy, while the 'residual error', which is similar to ensemble learning, enhances the model's performance.

Independent Component Analysis (ICA) originated from the 'cocktail party' problem, which refers to the challenging task of isolating individual speech signals from a noisy mixture of multiple speakers and background music in a cocktail party setting. The human ear is capable of accurately and precisely separating and processing each speaker's voice, despite the overlapping and interfering sounds. ICA is a data-driven signal processing method developed from blind source separation technology, and it is an analysis method based on high-order statistical properties. It employs statistical principles to separate data or signals into linear combinations of statistically independent non-Gaussian signal sources through linear transformation, as described in [17].

Prior to ICA whitening, preprocessing is necessary to standardize the input data. In traditional usage, ICA whitening involves significant computational overheads. However, deep learning frameworks incorporate several optimized operations, such as batch normalization (BN) and dropout, which require minimal computations. BN ensures that the input variables have zero mean and unit variance, which is equivalent to the standardization process prior to whitening, as described in [14]. Furthermore, Dropout randomly deactivates some neurons with a certain probability, as reported in [32]. Chen [4] demonstrated that BN and Dropout can approximate the effect of ICA.

**Table 2** Crawler Detection Features

| Type | Attribute Name | Type | Attribute Name |
|---|---|---|---|
| Request Features | 1.average_time<br>2.standard_deviation<br>3.total_duration<br>4. DATA<br>5-9. request resource rate<br>10-13. request method rate<br>14 - 17. response code rate | Habits, Temporal Features | 18. weekend day rate<br>19. deep_night_rate<br>20. resttime_rate<br>21. worktime_rate<br>22. user_time_series<br>23.width_max_rate<br>24.consecutive_size<br>25.single_flow_rate |

**Table 3** Datasets for Crawler Detection

| Dataset | Total Number | Crawlers | User-Agent |
|---|---|---|---|
| AUTH Log[18] | 4,060,400 | 411,946 | 3467 |
| NPCASSOC Log[25] | 482,055 | 80,839 | 9702 |

## 3.4 Federated Deep Learning Training

The detection server holding user access data $data = \{\ l_1, l_2, l_3, \ \ ... \ \ , l_n\}$ uses local data for training. We denote each server holding user data as $C_i, i \in [1, m]$, where $m$ represents the number of servers. The different detection servers hold local data sets represented by $D = \{d_1, d_2, \ \ ... \ \ , d_m\ \}$. Thus, the detection server $C_i$ only sends its own model weight parameters $g_i$ to the central training server for aggregation.

First, after training with local data, the detection server node $C_i, i \in [1, m]$ obtains the weight parameters $g_i$, which are then sent to the central training server. The central training server calculates the global parameter $G_c$.

$$G_c = \frac{1}{m} \sum_{i=1}^{m} g_i \tag{9}$$

Then, the central training server calculates the model weight parameter $g\prime_i$ for node $C_i$ for the next round of training using the following formula, and sends the updated weight parameter $g\prime_i$ to the detection server node $C_i$. The detection server node $C_i$ updates its local model weight parameters using the updated parameter $g\prime_i$.

$$g\prime_i = W_i^c G_c + W_i g_i \tag{10}$$

Where $W_i^C$ is the global parameter weight and $W_i$ is the local parameter weight. Repeat the above steps until the loss function converges or the set iteration training period is reached, then stop training and save the latest detection model weight parameters.

# 4 EXPERIMENTAL RESULTS AND ANALYSIS

## 4.1 Datasets

In order to assess the efficacy of our proposed crawler detection model, we conducted experiments on two distinct categories of datasets. The features utilized in these experiments are not contingent upon the user's private data, are legally permissible, and are impervious to tampering or forgery, as elaborated in Section 3.2. Table 2 exhibits the features obtained through the processing delineated in Section 3.2 of this article, and the foundation of these fields is generated from the server.

Table 3 presents a comprehensive overview of the datasets employed in this study. The first dataset used comprises the search engine user access data of the Library and Information Center of Aristotle University of Thessaloniki, Greece [18]. The server logs encompass the entire duration of February 28 to March 31, 2018, and comprise 4,060,400 HTTP requests and 3467 user-agents, with an average of 130,980 requests per day. Out of these requests, 411,946 were generated by crawlers, while the remaining 3,648,454 were regarded as normal human access records. The search engine enables users to peruse the availability of books and other paper-based works, as well as facilitating the search of digitized scholarly articles or materials.

The second dataset employed in this study to validate the proposed model is the access logs of the website of the American National Philosophical Counseling Association [25]. This dataset comprises 482,055 HTTP requests, out of which 80,839 were generated by crawlers, and the remaining 401,216 were human access records. The dataset also encompasses 9702 user-agents. The network data traffic in this dataset differs considerably from that of the first dataset utilized in this study, which enables a more robust assessment of the universality and accuracy of the crawler detection model.

From Table 3, it can be observed that the number of samples labeled as crawlers in both datasets is significantly lower compared to those of normal accesses. Training models directly with this imbalanced proportion could lead to a bias in the model, resulting in a distortion of the model's outcomes. To prevent this situation, in the AUTH Log dataset, we selected 411,946 crawler data samples. In the subsequent experiments, we experimented with training at ratios of 1:1, 1:1.5, and 1:2 to balance the data. A similar approach for data balancing was also employed with the NPCASSOC Log dataset.

The Table 2 in Section 3.2 shows the feature vector with 25 features obtained by the data processing algorithm. Feature 22, $user\_time\_series$, represents the user's access time series $ts$, and is processed by a time model to obtain temporal features $tf$. Features 5-9 represent resource request proportions, 10-13 represent request method proportions, and 14-17 represent response proportions in a session. Two additional features, session count and repeated request count, are not listed in the Table 2. These 27 features were obtained through the data processing algorithm in Section 3.2.

**Table 4** Hyperparameter Metrics

| detail | value |
|---|---|
| Round Number | 150 |
| Client Number | 100 |
| Number of clients selected for a round | 10 |
| Local clients batch size | 128 |
| Local clients epoch | 5 |
| Learning rate | 0.015 |
| Learning rate scheduler | 0.95 |
| Optimization function | Adam |
| Loss function | Cross Entropy Loss |
| dropout | 0.3 |

## 4.2 Experiment Setup

The simulations were performed using the Ubuntu/Linux 18.04 operating system. The crawler detection model was implemented with PyTorch 1.13 deep learning framework API in Python 3.10 to construct a deep neural network. Initial model development and experiments were executed on an Nvidia RTX 3060Ti GPU, leveraging a 2.9GHz 8-core Intel Core i7 10700F CPU and 32GB RAM. Each model was trained on the GPU utilizing the Adam optimizer for 150 rounds of federated learning, with 10 local training epochs per client per round. The model learning rate was set to 0.015, decaying by 0.95 each round. Details of hyperparameter settings during training are provided in Table 4.

A 4-layer fusion whitening residual network multi-layer perceptron (MLP) was used, with 42 input layer neurons and ReLU activation for the hidden layers. The output layer also used ReLU activation, generating two outputs to enable loss calculation via array subscript and label. The Adam optimizer and binary cross-entropy loss were utilized.

To evaluate model generalizability, training was conducted on two distinct datasets. As shown in Table 5, metrics obtained from separate training on the two datasets showed relatively minor differences, confirming the model can effectively detect crawlers across environments.

## 4.3 Evaluation Metrics

To evaluate the performance of our models, we use terms such as True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). We employ precision, recall, and F1 score.

Precision measures the proportion of correctly predicted positive samples to all samples predicted as positive.

$$Precision(P) = \frac{TP}{TP + FP} \tag{11}$$

**Table 5** Evaluation Performance on Two Datasets

| Datasets | F1 | Precision |
|---|---|---|
| AUTH Log[18] | 0.9228 | 0.9317 |
| NPCASSOC Log[25] | 0.9099 | 0.9174 |

Similarly, recall is the proportion of correctly predicted positive samples to all actual positive samples.

$$Recall(R) = \frac{TP}{TP + FN} \tag{12}$$

The F1 score is a harmonic mean of precision and recall.

$$F1score = 2 \cdot \frac{P \cdot R}{P + R} \tag{13}$$

Since precision and recall affect and constrain each other, we use the F1 score to balance these estimates and evaluate the model's overall performance with a single value.

## 4.4 Performance Comparison Between Federated and Centralized Model

In the field of contemporary network services, the immense volume of user traffic compels service providers to deploy numerous servers to ensure service reliability and enhance the user experience. Each server oriented towards user services accumulates independent user activity data, which encompasses a wealth of user privacy information. Presently, the training of crawlers' detection models predominantly relies on a centralized server training approach, where user access data collected by servers is transmitted to a central server for processing and training.

Under this centralized training architecture, the communication cost in terms of network bandwidth is primarily determined by the data traffic volume incurred when other servers upload data to the central training server. As depicted in Fig. 1, we evaluate these communication costs by comparing the volume of data exchanged between the servers and the central server.

As shown in Fig. 5A, the communication cost during centralized training is directly proportional to the size of the dataset maintained by the detection servers. If the dataset is substantial, it will consume a significant portion of network bandwidth resources, potentially encroaching upon the bandwidth originally allocated for user services. Moreover, frequent data interactions between the central training server and detection servers may pose a risk of sensitive data leakage.

In contrast, the training approach of distributed federated learning is not constrained by the size of the dataset, as it involves only the uploading of model weights, effectively avoiding the issue of user privacy leakage. In our research, we define the upload cost for a single round of federated training as the sum of the total size of all

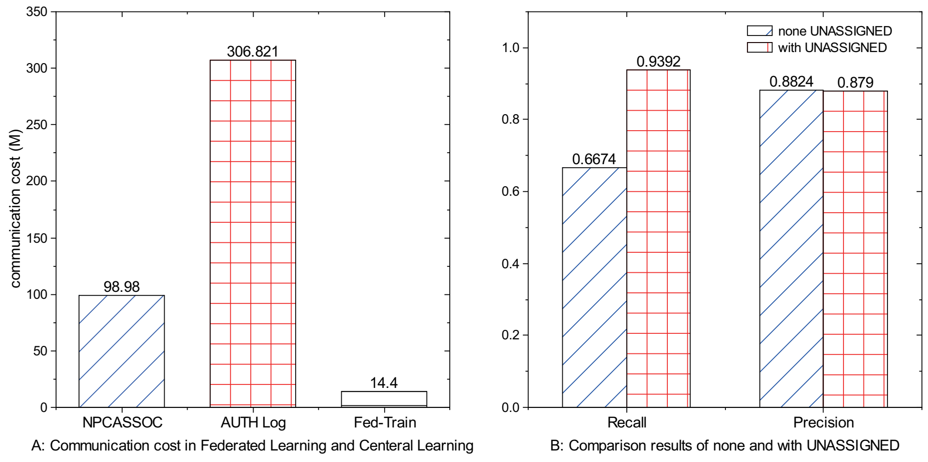


**Fig. 5** Experiments on the communication cost of training for federated learning and decisive weighted fields of the dataset.

local models plus the size of the dataset features that need to be uploaded. Correspondingly, the download cost is set as the cumulative size of all global models received by the detection servers.

## 4.5 Comparisons With The State of The Art

In the initial phase of our empirical investigation, we endeavored to corroborate the assertions posited by Lagopoulos et al. [18], specifically regarding the significance of the *UNASSIGNED* feature within the cited field. This feature, due to its susceptibility to manipulation by crawler entities, may ostensibly lack diagnostic value. The empirical evidence, as reflected by the recall metrics delineated in the corresponding table, intimates that the exclusion of the *UNASSIGNED* feature precipitates a diminution in the model's recall capability. Conversely, its integration engenders a conspicuous enhancement across all evaluative metrics. In light of the comparative analysis undertaken, we submit that the *UNASSIGNED* attribute is both pivotal and susceptible to falsification, as depicted in Fig. 5B. Consequently, it is our recommendation that this attribute be omitted from the formulation of the actual detection model, predicated on the premise that reliance on features that are readily susceptible to counterfeit by sophisticated crawlers can vitiate the model's ability to discern their activity, thereby precipitating a precipitous erosion in model performance. Table 6 furnishes a comprehensive performance comparison with contemporary state-of-the-art models.

Subsequent to this, we meticulously scrutinized and juxtaposed the implementation methodology delineated in the study conducted by Xia et al. [37], which utilizes word vector encoding to represent temporal and user privacy data, including geographic coordinates and temporal stamps. However, given the inherent nature of

**Table 6** Performance Comparison with State-of-the-Art Models

| Model | F1 | Precision | Privacy Protection | Emerging Detection |
|---|---|---|---|---|
| WC3D[12] | 0.9028 | 0.8788 | No | Yes |
| TMSCN[20] | - | - | No | Yes |
| Long-Tail[28] | - | - | No | No |
| PathMarker[36] | 0.918 | - | No | No |
| AANet[37] | 0.5115 | - | No | Yes |
| Content-aware WD[18] | 0.9593 | - | No | No |
| **Proposed TS-Finder** | **0.9228** | **0.9317** | **Yes** | **Yes** |

temporal stamps as continuous and ever-evolving independent variables, the resultant embedding model is potentially unwieldy, challenging both in computation and training. For web services intent on deploying detection algorithms within production milieus, sustaining such a training regimen may be impracticable, especially when juxtaposed against the backdrop of stringent privacy regulations governing the analytical utilization of user data.

In our final comparative analysis, we scrutinized the methodologies employed by Ro et al. [28]. The insights gleaned from our study cast doubt on the sufficiency of the feature data presented in their research for the efficacious identification of crawler activities. The said study relies predominantly on a time series detection approach, as catalogued in the table pertaining to the Temporal Model. When this singular method was integrated into our multi-modular assessment framework, the F1 score was recorded at a modest 0.7735, underscoring the need for a more robust and multifaceted feature set to enhance detection fidelity.

## 4.6 Ablation Study of Modules

To scrutinize the potential redundancy within the constituent modules of the proposed model, a series of ablation studies were meticulously executed. The quantitative results derived from these investigations are systematically tabulated in Table 7. More precisely, the Temporal Model, as illustrated in Fig. 2, is meticulously calibrated to distill temporal features exclusively from the input comprising the user's chronological access data. These features are subsequently harnessed as determinants for decision-making subsequent to a spatial transformation process. Conversely, the MLP module, which integrates a suite of whitening residual networks and a sophisticated multilayer perceptron architecture, is singularly employed in this experimental context to assimilate and process the intricate patterns of user behavior and associated spatial features for the purpose of model training.

The empirical outcomes corresponding to these experiments are encapsulated in Table 7. In a distinct experimental configuration, the Temporal Model is operationalized to isolate temporal features from the user visitation temporal sequence, and

**Table 7** Ablation Study of Modules on Detection Crawlers Model

| Modules | F1 | Recall | Precision | AUC |
|---|---|---|---|---|
| DWRN MLP | 0.8861 | 0.8647 | 0.9084 | 0.9179 |
| Temporal Model | 0.7735 | 0.7024 | 0.8606 | 0.8320 |
| **DWRN MLP + Temporal Model** | **0.9228** | **0.9141** | **0.9317** | **0.9457** |

**Table 8** Exact Experimental Metrics for Different Temporal Models on Detection Crawlers Model

| Modules | F1 | Recall | Precision | AUC |
|---|---|---|---|---|
| LSTM [13] | 0.9099 | 0.9026 | 0.9174 | 0.9377 |
| Bi-LSTM[30] | 0.9017 | 0.8857 | 0.9183 | 0.9296 |
| RNN | 0.8974 | 0.8780 | 0.9177 | 0.9258 |
| **GRU** | **0.9228** | **0.9141** | **0.9317** | **0.9457** |

these distilled features are employed as the foundational elements for model training. In this instance, the DWRN MLP module is deliberately excluded from the process, and the feature vector space is directly transmuted into a vector conducive for crawler detection. The empirical evidence garnered from these experiments unequivocally demonstrates that both user habituation patterns and spatio-temporal behavioral features are integral and salient attributes that substantially enhance the efficacy of crawler detection mechanisms. It is worth noting that the exclusion of either feature module precipitates a tangible decrement in the precision of the crawler detection accuracy.

## 4.7 Comparisons Between Different Models

In this investigation, we have elected to juxtapose the extant temporal model with frequently employed time series models. To achieve this objective, we have chosen an array of temporal models for comparative analysis. The comparative assessment reveals that the performance discrepancies among the selected temporal models are minimal. Nevertheless, the GRU [9] model demonstrates marginally superior training effectiveness relative to other models, as corroborated by the precise experimental data indicators delineated in Table 8. Consequently, we have incorporated the GRU model in the proposed model design and in the appraisal of multiple datasets.

## 4.8 Model Stability Performance

Owing to the opaque nature of deep learning neural network models, it is not uncommon for these models to exhibit disparate performance when executed on different machines. In order to mitigate this issue, we have taken the necessary steps to validate our model on multiple machines with different GPUs, thereby enhancing its robustness. To assess the stability of our design model across diverse GPU servers available

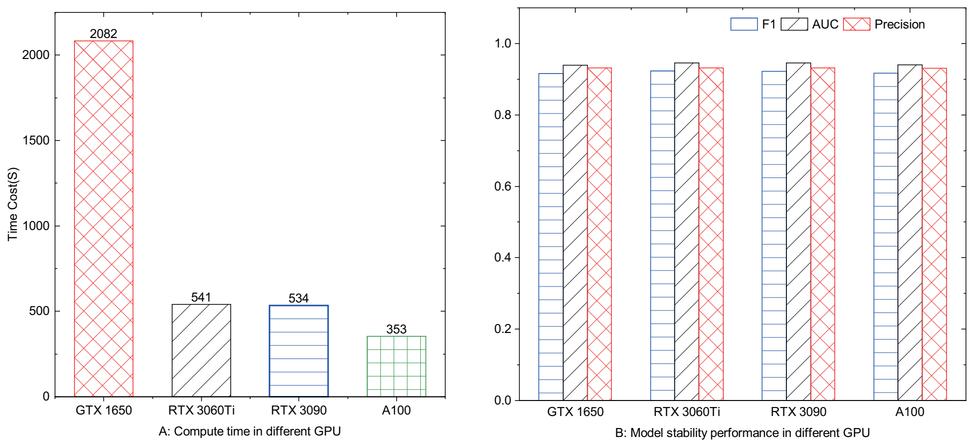


**Fig. 6** Model Stability Performance.

in the market, and as depicted in Fig. 6, we have obtained training metrics for the model on A100, RTX 3090, RTX 3060Ti, and GTX 1650 NVIDIA GPU servers.

Based on the results presented in Fig. 6A, it can be inferred that the time spent on a standard GTX 1650 GPU was the highest among all tested GPUs. Conversely, as depicted in Fig. 6B, the training metrics for various high-performance GPUs showed minimal differences. Notably, the Recall, and Precision metrics for the detection models exhibit only slight variations across different performance GPU servers. This observation suggests that the model presented in this paper delivers comparable performance when trained on both low-cost and high-performance GPU servers.

## 5 CONCLUSION

This paper introduces a state-of-the-art model for crawler detection, TS-Finder, which leverages users' access patterns and spatial-temporal characteristics of visited sites. By extracting these features from log timestamps and addresses that cannot be falsified or tampered with, this approach does not rely on private user data. Instead, we construct user access path trees based on relative access addresses and visit times. We then employ a time series model coupled with an attention mechanism to train and extract user access behaviors, alongside a whitening residual network to learn patterns in user habits and spatial-temporal behavior features - both critical for effective crawler detection. To further enhance our approach's efficacy, we implement federated distributed learning to train our models. This not only mitigates excessive model bandwidth usage caused by exchanging large training data quantities between servers, but also ensures private data leakage is not a concern.

An intended direction for future work is addressing the scarcity of labeled data in this domain. The current model has a strong dependence on labeled datasets, which can be limiting in real-world scenarios. To overcome this constraint, we plan to explore generative adversarial networks and game theory to learn the distributions of positive and negative data from existing datasets. This will help reduce the problem of missing data for detection models and enable the development of more robust and accurate crawler detection.

## Declarations

**Conflict of interest.** The author declares no competing interests.

**Acknowledgments.** We would like to sincerely thank the editors and anonymous reviewers for their helpful comments.

**Data availability.** The implemented code used to support the findings of this study is available from the corresponding author upon request. The datasets used in this paper are publicly available for download.

## References

[1] Acien A, Morales A, Fierrez J, et al (2022) Becaptcha-mouse: Synthetic mouse trajectories and improved bot detection. Pattern Recognition 127:108643

[2] Brown K, Doran D (2018) Contrasting web robot and human behaviors with network models. arXiv preprint arXiv:180109715

[3] Browser Capabilities Project (12, 2022) Browscap project. https://browscap.org/

[4] Chen G, Chen P, Shi Y, et al (2019) Rethinking the usage of batch normalization and dropout in the training of deep neural networks. arxiv 2019. arXiv preprint arXiv:190505928

[5] Cho K, Van Merriënboer B, Gulcehre C, et al (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:14061078

[6] Chu Z, Gianvecchio S, Wang H (2018) Bot or human? a behavior-based online bot detection system. In: From Database to Cyber Security. Springer, p 432–449

[7] COUNTER (11, 2022) Counter-robots. https://github.com/atmire/COUNTER-Robots

[8] Devlin J, Chang MW, Lee K, et al (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:181004805

[9] Dey R, Salem FM (2017) Gate-variants of gated recurrent unit (gru) neural networks. In: 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS), IEEE, pp 1597–1600

[10] Doran D, Gokhale SS (2016) An integrated method for real time and offline web robot detection. Expert Systems 33(6):592–606

[11] Eswaran S, Rani V, Ramakrishnan J, et al (2022) An enhanced network intrusion detection system for malicious crawler detection and security event correlations in ubiquitous banking infrastructure. International Journal of Pervasive Computing and Communications 18(1):59–78

[12] Gao Y, Feng Z, Wang X, et al (2023) Reinforcement learning based web crawler detection for diversity and dynamics. Neurocomputing 520:115–128

[13] Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural computation 9(8):1735–1780

[14] Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, PMLR, pp 448–456

[15] Joulin A, Cissé M, Grangier D, et al (2017) Efficient softmax approximation for gpus. In: International conference on machine learning, PMLR, pp 1302–1310

[16] Kayan H, Nunes M, Rana O, et al (2022) Cybersecurity of industrial cyber-physical systems: A review. ACM Computing Surveys (CSUR) 54(11s):1–35

[17] Kwak N, Choi CH, Choi JY (2001) Feature extraction using ica. In: International Conference on Artificial Neural Networks, Springer, pp 568–573

[18] Lagopoulos A, Tsoumakas G (2020) Content-aware web robot detection. Applied Intelligence 50(11):4017–4028

[19] Lan Z, Chen M, Goodman S, et al (2019) Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:190911942

[20] Li S, Lee CH, Eun DY (2020) Trapping malicious crawlers in social networks. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp 775–784

[21] Li X, Azad BA, Rahmati A, et al (2021) Good bot, bad bot: Characterizing automated browsing activity. In: 2021 IEEE symposium on security and privacy (sp), IEEE, pp 1589–1605

[22] Lu WZ, Yu SZ (2006) Web robot detection based on hidden markov model. In: 2006 International Conference on Communications, Circuits and Systems, IEEE, pp 1806–1810

[23] McMahan B, Moore E, Ramage D, et al (2017) Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, PMLR, pp 1273–1282

[24] Menshchikov A, Komarova A, Gatchin Y, et al (2017) A study of different web-crawler behaviour. In: 2017 20th Conference of Open Innovations Association

(FRUCT), IEEE, pp 268–274

[25] npcassoc access log (2018) npcassoc.org. http://npcassoc.org/log/access.log

[26] Radford A, Wu J, Child R, et al (2019) Language models are unsupervised multitask learners. OpenAI blog 1(8):9

[27] Rahman RU, Tomar DS (2020) New biostatistics features for detecting web bot activity on web applications. Computers & Security 97:102001

[28] Ro I, Han JS, Im EG (2018) Detection method for distributed web-crawlers: A long-tail threshold model. Security and Communication Networks 2018

[29] SayWeee Inc (2023) Security incident. https://www.sayweee.com/en/view/february-2023-data-breach

[30] Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE transactions on Signal Processing 45(11):2673–2681

[31] Shkapenyuk V, Suel T (2002) Design and implementation of a high-performance distributed web crawler. In: Proceedings 18th International Conference on Data Engineering, IEEE, pp 357–368

[32] Srivastava N, Hinton G, Krizhevsky A, et al (2014) Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15(1):1929–1958

[33] Suchacka G, Motyka I (2018) Efficiency analysis of resource request patterns in classification of web robots and humans. In: ECMS, pp 475–481

[34] Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. Advances in neural information processing systems 27

[35] Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. Advances in neural information processing systems 30

[36] Wan S, Li Y, Sun K (2017) Protecting web contents against persistent distributed crawlers. In: 2017 IEEE International Conference on Communications (ICC), IEEE, pp 1–6

[37] Xia W, Zhao F, Wang H, et al (2021) Crawler detection in location-based services using attributed action net. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp 4234–4242

[38] Yu L, Li Y, Zeng Q, et al (2020) Summary of web crawler technology research. In: Journal of Physics: Conference Series, IOP Publishing, p 012036

[39] Zaremba W, Sutskever I, Vinyals O (2014) Recurrent neural network regularization. arXiv preprint arXiv:14092329

[40] Zhuang Z, Kong X, Elke R, et al (2019) Attributed sequence embedding. In: 2019 IEEE International Conference on Big Data (Big Data), IEEE, pp 1723–1728