AI-based user identification method for web services

Ihor Zakutynskyi[1,*,†], Oleksandr Kalishuk[1,†], Maksim Iavich[2,†], Vitalii Nebylytsia[1,†] and Vasyl Yehunko[1,†]

[1] National Aviation University, Liubomyra Huzara Ave. 1, Kyiv, 03058, Ukraine
[2] Caucasus University, Paata Saakadze Str., 1, Tbilisi, 0102, Georgia

Abstract

In our paper, we introduce a universal web service user's identification method. This method is based on analyzing the digital fingerprint of the visitor using a neural network. Within the scope of our research, we performed a comparative analysis between our developed method and the existing fingerprint detection services. The testing results indicate that the accuracy of fingerprint identification using our method surpasses fingerprint.com by 3.1% on desktop platforms and 6.3% on mobile devices. Furthermore, the utilization of our method significantly reduces the number of false positive errors, thereby enhancing the robustness of user identification against variations in browser and device parameters.

Keywords

digital fingerprint, user identification, neural network, LSTM

1. Introduction

In our paper, we present a universal web service user's identification method, which is based on creating a digital fingerprint that is determined using a dataset collected both on the client side (using a JS library) and on the server side (from the HTTP request data from the client) and subsequent analysis by a neural network. The method we have developed for calculating and evaluating a set of parameters using a neural network trained on a test database of users allows for achieving: 1) Greater overall accuracy in user identification, 2) Extended lifespan of the digital fingerprint, 3) Correct cross-browser user identification, 4) Accurate user identification through VPN.

Moreover, the user recognition process requires no significant computational resources, maintains a high identification speed, has a low collision rate, and high accuracy [1].

The neural network helps us identify hidden patterns in parameters and allows us to reveal implicit associations among sets of parameters in the digital fingerprints of visitors [2, 3].

At the same time, our method provides strong protection of privacy and security of user data.

2. Background

Browser fingerprint or device fingerprint, combined into the concept of a digital fingerprint, is information collected about the software and hardware of a remote device for the purpose of its identification.

CH&CMiGIN'24: Third International Conference on Cyber Hygiene & Conflict Management in Global Information Networks, January 24–27, 2024, Kyiv, Ukraine

* Corresponding author.
† These authors contributed equally.
ihor.zakutynskyi@nau.edu.ua (I. Zakutynskyi); akalishuk@gmail.com (O. Kalishuk); miavich@cu.edu.ge (M. Iavich); tet129@gmail.com (V. Nebylytsia); 7253362@stud.nau.edu.ua (V. Yehunko)
0000-0003-2905-3205 (I. Zakutynskyi); 0009-0008-1577-6473 (O. Kalishuk); 0000-0002-3109-7971 (M. Iavich); 0009-0000-0154-9909 (V. Nebylytsia); 0000-0002-5316-8996 (V. Yehunko)

2.1. Fingerprinting techniques

The technique of digital fingerprinting has existed for many years. The first mentions of various techniques for obtaining and analyzing digital fingerprints in scientific literature appeared in 2003 [4], and they have been widely studied since 2009 [5].

Since then, many different techniques for determining the digital fingerprint have been described:

- JavaScript-Based Fingerprints
- CSS-Based Fingerprints
- Canvas-Based Fingerprint
- Hardware and Software-Based Fingerprints
- Fingerprint Based on Audio API
- Plugin-Based Fingerprint
- TLS Fingerprint
- Other Browser Fingerprint Acquisition Technologies (correlation between visitor's gaze and mouse movement; characteristics of HTML parser; font sets (font glyphs); methods based on calculation of JavaScript scripts set execution time; based on user lag time on websites; on the nature of user interaction with touchpad; speed and specificity of typing on keyboard; speed and directions of mouse movement).

In most cases, to identify a digital fingerprint, a scheme is used in which code based on a special JS library is executed on the client side. The code performs a set of tests and checks defined by the library and send the received parameters to the server. Usually, the server is deployed as a separate service (Figure 1).

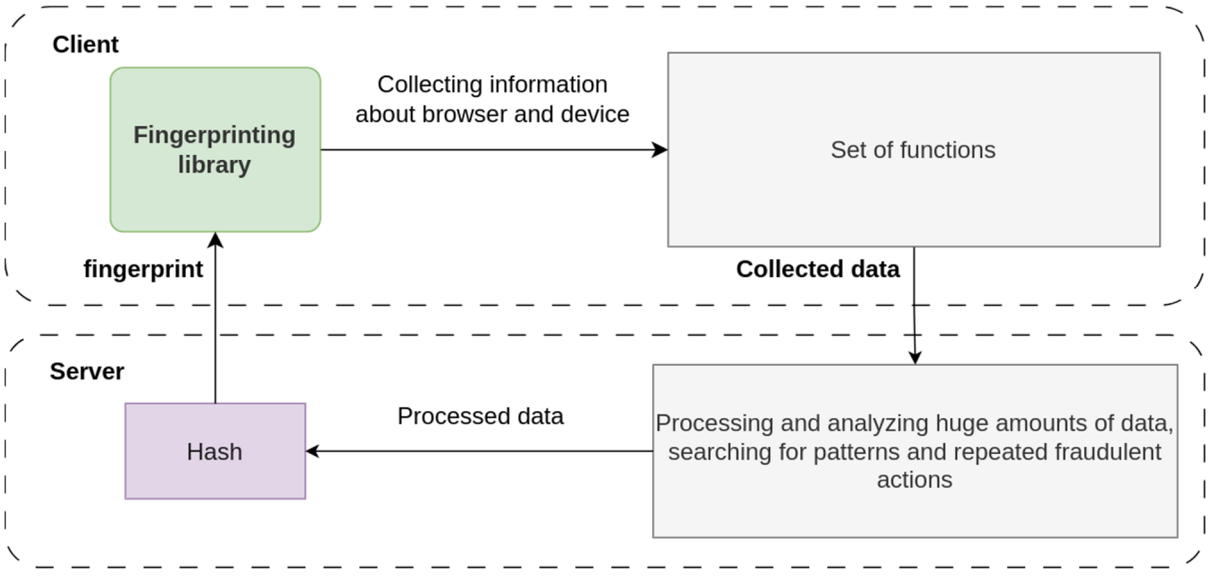

Figure 1: General fingerprinting process.

All modern methods of identifying digital fingerprinting have both advantages and disadvantages.

2.2. Fingerprinting advantages and disadvantages

The main drawbacks of fingerprinting solutions include:

- Low user identification accuracy,
- Computation time for generating a digital fingerprint,
- Time required for matching with previously known digital fingerprints in the system,

- Short lifespan of a specific digital fingerprint,
- High device load on the user's end,
- Dependence on JavaScript,
- Challenges in computing a digital fingerprint in homogeneous environments (computer labs, internet cafes, mobile network environments),
- Cross-browser digital fingerprinting,
- Low accuracy in identifying users operating in incognito mode,
- Matching digital fingerprints over VPN.

In addition to the mentioned drawbacks of existing methods for digital fingerprinting based on open solutions, ready-made commercial services are characterized by additional disadvantages:

- High cost,
- Closed source nature,
- Data stored on third-party servers,
- Dependence on the service provider.

In our assessment, there are currently no effective methods that reliably identify a user based on their digital fingerprint over an extended period, especially when using VPN, incognito mode, or engaging in cross-browser surfing.

2.3. The literature review

We reviewed some research papers that address the problems of fingerprinting and user identification on the Internet.

In [6], the authors reviewed and classified the existing fingerprinting techniques and their applications for user identification on the Internet and analyzed in detail the development of different research directions of browser fingerprinting. Based on the analysis of existing results, the problems faced by different research directions are pointed out. Also, the research achievements in the field of browser fingerprint recognition are summarized and the trend of future development is pointed out. The authors also discussed the privacy issues associated with the use of fingerprinting techniques.

The authors of the paper [7] show that GPU information obtained using WebGL and other technologies can be used to create a unique device fingerprint that can be used for user identification. At the same time, the authors note that changing GPU settings and parameters can change the device fingerprint, which makes identification more difficult.

In the study [8], the authors demonstrate the correlation between gaze and mouse movements and argue that this serves as a valuable source for obtaining browser fingerprints. Simultaneously, the authors point out that collecting data on a person's gaze in the browser has drawbacks, such as inaccuracies when using a webcam and the limitation that users must grant permission for camera access. The study also reveals that, in the case of computers used by multiple users, browser statistics may malfunction and can no longer differentiate between individuals.

In the article [9] authors analyze the popularity of the Transport Layer Security (TLS) protocol on the Internet and its use in censorship circumvention tools. The researchers collected and analyzed a huge volume of real-world TLS traffic to identify the different implementations of TLS clients used on the Internet. Censors can use deep packet inspection (DPI) to identify and block such tools based on their TLS fingerprints. That said, many circumvention tools fail to properly mimic popular TLS implementations, leading to their detection and blocking. To solve the censorship circumvention problem, the authors proposed a solution that allows developers to automatically mimic other popular TLS implementations. Using real-world data, the authors of the paper propose methods to flexibly adapt TLS-fingerprint to the dynamic TLS ecosystem with minimal manual effort.

The authors of the paper [10] propose a new mobile device user's identification method based on the study and analysis of touch dynamics, which has stable patterns of interaction between the user and his mobile device, including factors such as touch force, swipe speed and duration of touch.

This method has shown excellent results, but its scope is limited to only a subset of mobile devices and depends on the availability of APIs for interacting with physical device elements.

In the paper [11], the authors propose a browser fingerprinting defense tool to anonymize users' browsers. The authors show that browser fingerprinting cannot be prevented by the user. Although new methods are constantly being developed that can prevent browser fingerprinting, they cannot prevent it completely.

In the article [12], presents new algorithms for encoding and comparing fingerprints, which focus on the values of parameters with low stability and low entropy.

2.4. Benefits of our method

The method proposed by the authors allows for:

- Improved accuracy in user identification under specified conditions,
- Reduce the percentage of false positives,
- Increased lifespan of the calculated digital fingerprint,
- Maintenance of the speed of digital fingerprint identification at an industry-standard level.

All of these improvements are achieved through the implementation of a novel neural network training algorithm. The results of determining the digital fingerprint of a web service user are a non-linear time series consisting of a set of browser and user device parameters and may vary over time [13, 14]. As the practice of the last 10 years shows, recurrent neural networks (RNN) are the most effective architecture for solving time series problems that cannot be solved by feedforward networks [15]. We performed comparative tests of the two most common RNN architectures LSTM and GRU by the methodology described in [16]. The results of the digital fingerprint accuracy tests are presented in Figure 2.

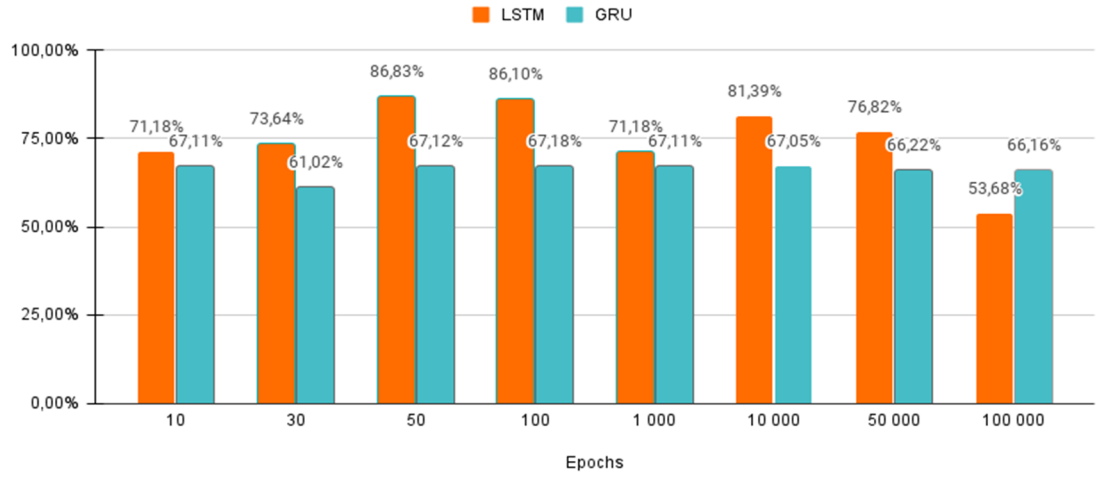

Figure 2: LSTM vs GRU comparison.

For our solution, we utilized the LSTM architecture as it demonstrated significantly better results over a small number of training epochs (50-100 epochs). This implies that, with equal resource consumption, LSTM yields superior results, which can be expressed by:

$$network\ accuracy\ = \left\{ \frac{epochs_{max} \to min}{N_n\ accuracy \to max} \right\}. \quad (1)$$

3. Experiment

3.1. Competitor

Currently, the majorities of systems for obtaining a digital fingerprint are based on the fingerprint.js library or incorporate some of its functions. This library, one of the earliest to emerge, is dynamically evolving and includes prospective developments that emerge periodically. The library is actively developing, and the project repository is frequently updated. As of December 2023, the latest version is 4 [17]. Starting from this version, the developer has changed the distribution terms, and it is now offered under the Business Source License 1.1. Currently, the FingerprintJS service is considered an industry standard.

The service allows for the identification of numerous browser and operating system parameters. The key modules of the fingerprint.js library are outlined in Table 2.

Table 1
Key Modules of the fingerprint.js Library

Parameter	Function	Type of Returned value
Audio fingerprint	getAudioFingerprint()	number or Promise<number>
Fonts	getFonts()	string[]
Plugins	getPlugins()	string[]
Canvas	getCanvasFingerprint()	object
Touchscreen	getTouchSupport()	object
OS CPU	getOsCpu()	string | undefined
Languages	getLanguages()	string[][]
Color depth	getColorDepth()	number
Memory	getDeviceMemory()	number
Resolution	getScreenResolution()	[number | null, number | null]
Screen frame size	getRoundedScreenFrame()	[number | null, number | null, number | null]
Hardware concurrency	getHardwareConcurrency()	number | undefined
Time zone	getTimezone()	string
Session storage	getSessionStorage()	boolean
Local storage	getLocalStorage()	boolean
Indexed DB	getIndexedDB()	boolean | undefined
Open DB	getOpenDatabase()	boolean
CPU class	getCpuClass()	string | undefined
Platform	getPlatform()	string
Vendor	getVendor()	string
Vendor flavors	getVendorFlavors()	string[]
Cookie enabled	areCookiesEnabled()	Boolean
Ad blockers	getDomBlockers()	Promise<string[] | undefined>

Color gamut	getColorGamut()	string | undefined
Color inverted mode	areColorsInverted()	boolean | undefined
Colors forced	areColorsForced()	boolean | undefined
Monochrome depth	getMonochromeDepth()	number | undefined
Contrast	getContrastPreference()	number | undefined
Reduced motion	isMotionReduced()	boolean | undefined
HDR	isHDR()	boolean | undefined
Math calc	getMathFingerprint()	Record<string, number>
Font width	getFontPreferences()	Promise<Record<string, number>>
Video card (WebGL)	getVideoCard()	object | undefined
PDF viewer	isPdfViewerEnabled()	boolean
Architecture	getArchitecture()	number

The general algorithm of operation for the fingerprint.js library is presented in Figure 3.

Algorithm 1 General fingerprinting process

Input: $Web\ browser\ instance$
Output: user fingerprint $fingerprint$
Initialise: array of browser and device parameters F
if $localStorage \ni fingerprint$ **then**
  return $fingerprint$
else
  Get: array of functions as G
  for i=1 to $G.length$ **do**
    Add $G[i]$ execution result to the F
  end for
  Send: $POST\_body\{F\}$ request to server API
  Save: received result to $localStorage$ as $fingerprint$
  return $fingerprint$
end if

Figure 3: General fingerprinting algorithm.

3.2. Neural network training

At the initial stage of preparing data for training the neural network, we have a multidimensional dataset about the user collected in the previous stage. To optimize time and computational resource costs, this multidimensional dataset is transformed into a linear vector. Thus, the neural network receives a one-dimensional vector as input.

Next, after normalization, the data is randomly split into testing and training sets in a 30%/70% ratio.

Based on the testing set, a prediction is made to determine if the visitor is known in our service, and the prediction result is compared with the result obtained based on the predefined parameters of the model. The schematic process of training the neural network is illustrated in Figure 4.

The initial training of the model was conducted using the "Login Data Set for Risk-Based Authentication" dataset from Kaggle [13]. This dataset includes a list of parameters associated with each login attempt.

The structure of the dataset is presented in Table 2.

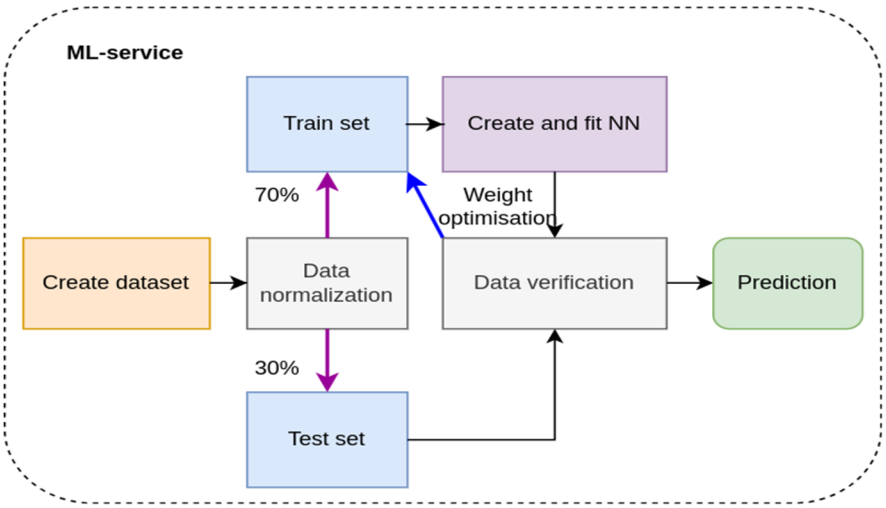

Figure 4: Neural network training algorithm.

Table 2
Dataset Structure

Characteristics	Type	Range or example
User Agent	String	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36.
HTTP Accept Headers	String	Access-Control-Allow-Origin: *, Cache-Control: max-age=604800, Content-Type: multipart/form-data, If-Unmodified-Since: Mon, 27 Nov 2023 12:43:00 EET
Language	String	uk
Screen Resolution	Integer (Width x Height)	2073600
Timezone	String	Europe/Kiev
Browser Plugins	List of Strings []	[PDF Viewer, Chrome PDF Viewer, Chromium PDF Viewer, Microsoft Edge PDF Viewer, WebKit built-in PDF]
Platform (Operating System)	String	Linux x86_64
Browser Version	String	Chrome 119
Device Memory	Integer (in gigabytes)	8
Canvas Fingerprint	String (hashed or raw data)	93a13b9b08d18393f5c731f8f5c58a11
WebGL Vendor and Renderer	String	WebKit WebGL
Cookies Enabled	Boolean	TRUE

Characteristics	Type	Range or example
Do Not Track (DNT) Header	Boolean	FALSE
Fonts List	List of Strings []	["4274,142 default, cursive, fantasy", "4314,143 sans-serif, Arial, Arimo, Helvetica, Liberation Sans", "4249,142 serif", "3780,149 monospace", "4431,143 system-ui, Ubuntu", "4189,143 aakar"]
Audio Fingerprint	String (hashed or raw data)	13b9b08d18393f5c731f8f5c58a116dcb
Hardware Concurrency	Integer	4
Touch Support	Boolean	FALSE
Geolocation	Boolean	FALSE
Connection Speed	String	4g
Ad Blocker Detection	Boolean	FALSE
Local IP Address	String	0.0.0.0 - 255.255.255.255
WebRTC Leak	Boolean	TRUE
Battery Level	Float (percentage)	78.2
CPU Cores	Integer	4
Device Type	String	Desktop
Hash of User Identity Information	String (hashed)	52d84b11737d980aef856699f885ca86

3.3. Experiment conditions

To perform an experiment comparing the effectiveness of the developed method and the method of digital fingerprinting using the FingerprintJS service, a set of parameters from 2134 devices of different types (desktop computers, mobile devices, tablets) and a set of user agents that was generated using the npm package User-Agents [18] were used. User-Agents are a JavaScript package for generating random user agents based on how often they are used in a real environment.

The generated data includes hard-to-find browser fingerprint properties, and powerful filtering capabilities allow the generated user agents to be constrained to fit specific needs.

An experiment to measure the qualitative performance of the developed web service user identification method was performed on the current web service using the algorithm that is shown in Figure 5.

3.4. The results of the experiment

The results of the experiment are summarized in Tables 3 – 5.

Algorithm 2 Software emulator for determining the accuracy of the fingerpring detection method

Input:
N - the number of virtual devices for testing
$browser\_set[]\,(optional)$ - the list of browsers for testing
$user\_agent\_set[]\,(optional)$ - the list of user agents for testing
Output: $method\_accuracy\_n$
Define: $testing\_report[\{$
$device\_id,$
$browser,$
$user\_agent,$
$method\_n\_fingerprint$
$method\_n\_calc\_time$
$\}]$
for $i = 0$ to N **do**
  for $j = 0$ to $browser\_set.length$ **do**
    for $k = 0$ to $user\_agent\_set.length$ **do**
      Run: $browser[browser\_set[j]]$ with $user\_agent[user\_agent\_set[k]]$
      for $p = 0$ to n **do**
        Start: execution time evaluating: $start\_T$ = current time
        Run: pages with $fingerprint$ detection
        $stop\_T$ = current time
        Calculate: execution time evaluating
        $method\_n\_calc\_time$ = $stop\_T$ - $start\_T$
        Write: testing report:

$$testing\_report \leftarrow \begin{Bmatrix} i, \\ browser\_set[j], \\ user\_agent\_set[k], \\ method\_n\_fingerprint, \\ method\_n\_calc\_time[n] \end{Bmatrix}$$

      end for
    end for
  end for
end for
Define: $error\_rates[]$
for $i = 0$ to N **do**
  Group: results by $device\_id$:
  $grouped\_report = testing\_report.filter(val => val.device\_id === i)$
  Find: the most frequent $fingerprint$ and its share from the total:
  $correct\_detections$
  Calculate: device error:
  $Error\_n \leftarrow (100 - (testing\_report.length - correct\_detections))$
end for
Calculate: method accuracy:
$method\_accuracy\_n \leftarrow (100 - \sum(\frac{error\_rates[n]}{error\_rates.length}))$
return $method\_accuracy\_n$

Figure 5: Algorithm of the experiment.

Table 3
Comparison Results: Desktop

Method	Developed	Standard	Developed	Standard	Developed	Standard	Developed	Standard	**Developed**	**Standard**
Platform	MacIntel		Linux		Windows		Android		**Total**	
Total executions	226		188		871		329		**1614**	
Accuracy, %	93,1	91,5	94,4	89,1	93,8	91,3	93,6	89,3	**93,7**	**90,7**
False positive	8	12	5	15	28	60	11	18	**52**	**105**
False negative	8	7	5	6	26	16	10	18	**49**	**47**
Duration, ms	59	78	71	69	54	55	77	82	**61**	**65**

Table 4
Comparison Results: Mobile

Method	Developed	Standard	Developed	Standard	Developed	Standard	Developed	Standard	Developed	Standard	**Developed**	**Standard**
Platform	iPhone		Android type 1		Android type 2		Linux		Android type 3		**Total**	
Total executions	73		114		93		17		122		**419**	
Accuracy, %	98,9	87,1	96,5	88,7	95,2	91,2	95,2	88,3	94,6	91,4	**96,0**	**89,7**
False positive	0	6	2	6	2	6	0	1	4	7	**8**	**26**
False negative	0	3	2	7	2	2	0	1	3	3	**7**	**16**
Duration, ms	156	152	164	168	181	164	160	138	143	146	**160**	**157**

Table 5
Comparison Results: Tablet

Method	Developed	Standard	Developed	Standard	Developed	Standard	**Developed**	**Standard**
Platform	Android type 3		iPad		Android type 1		**Total**	
Total executions	19		28		54		**101**	
Accuracy, %	97,2	88,4	94,3	87,1	93,8	90,6	**94,6**	**89,2**
False positive	0	1	1	2	2	4	**3**	**7**
False negative	0	1	1	2	2	1	**3**	**4**
Duration, ms	92	96	110	121	99	104	**101**	**107**

4. Conclusions

The accuracy comparison data for digital fingerprint identification indicate that for desktop computers, the accuracy of the existing identification method (FingerprintJS) is 90.7%, while the accuracy of our developed method is 93.7%, representing a 3.1% improvement.

For mobile devices, the accuracy of the existing user identification method (FingerprintJS) is 89.7%, whereas the accuracy of our developed method is 96%, showcasing an improvement of 6.3%.

In the case of tablets, the accuracy of the existing identification method (FingerprintJS) is 89.2%, which is 5.4% lower than that of our developed method (94.6%).

The weighted average accuracy of the method developed by us is 3.8% higher than the existing method (94.2% versus 90.4%).

The stability of the algorithm directly depends on reducing the percentage of false positives and false negatives in user identification. The stability of the algorithm can be determined using equation

$$stability = \frac{TP + TN}{TP + TN + FP + FN}, \quad (2)$$

where TP - true positive, TN - true negative, FP - false positive, FN - false negative.

The method developed by us shows a lower number of false positive fingerprint identification results on all investigated platforms:

- Desktop computers: 52 versus 105,
- Mobile devices: 8 versus 26,
- Tablets: 3 versus 7.

The weighted average number of false positive errors for the developed method is 41.0, compared to 84.9 for the existing method.

The number of false negative results in digital fingerprint identification is comparable for both methods on all investigated platforms, with the advantage of the developed method being notably better only on mobile devices:

- Desktop computers: 49 versus 47,
- Mobile devices: 7 versus 16,
- Tablets: 3 versus 4.

The weighted average number of false negative errors for the developed method is 38.6, compared to 38.9 for the existing method. According to formula (2), with a decrease in the number of errors, the overall stability of the method increases. Based on the results obtained, due to a significant reduction in the number of false positive results for the developed method, its stability to changes is higher by 2% compared to the results of the existing method. The number of false negative results is comparable, so it did not significantly impact the final comparison result.

The duration of the identification process using the developed method varies in the ranges of 59-77 ms for desktop computers, 143-181 ms for mobile devices, and 92-110 ms for tablets. Based on the comparison results, it can be concluded that the speed of user identification using the developed method is comparable to the speed of identification using existing modern methods.

The analysis of the obtained results shows that the developed method has higher accuracy on all investigated types of devices and platforms. Additionally, it exhibited a lower overall error rate in the accuracy of identification and comparable speed in the process of digital fingerprint determination.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] J. S. Al-Azzeh, M. Al Hadidi, R. S. Odarchenko, S. Gnatyuk, Z. Shevchuk, Z. Hu, Analysis of self-similar traffic models in computer networks, International Review on Modelling and Simulations 10(5) (2017) 328–336. doi: 10.15866/iremos.v10i5.12009.

[2] M. Zaliskyi, R. Odarchenko, S. Gnatyuk, Y. Petrova, A. Chaplits, Method of traffic monitoring for DDoS attacks detection in e-health systems and networks, CEUR Workshop Proceedings 2255 (2018) 193–204. URL: https://ceur-ws.org/Vol-2255/paper18.pdf.

[3] V. Tkachuk, Y. Yechkalo, S. Semerikov, M. Kislova, Y. Hladyr, Using mobile ICT for online learning during COVID-19 lockdown, Communications in Computer and Information Science, 1308 (2021) 46–67. doi: 10.1007/978-3-030-77592-6_3.

[4] A. Hintz, Fingerprinting websites using traffic analysis, in: R. Dingledine, P. Syverson (Eds.), Privacy Enhancing Technologies. PET 2002, volume 2482 of Lecture Notes in Computer Science, Springer, Berlin, 2003. doi: 10.1007/3-540-36467-6_13.

[5] J. R. Mayer, Any person a pamphleteer: Internet Anonymity in the Age of Web 2.0, Princeton University, 2009, Undergraduate Senior Thesis. URL: http://arks.princeton.edu/ark:/88435/dsp01nc580n467.

[6] D. Zhang, J. Zhang, Y. Bu, B. Chen, C. Sun, T. Wang, A survey of browser fingerprint research and application, Wireless Communications and Mobile Computing, 2022. doi: 10.1155/2022/3363335.
[7] DRAWNAPART: A Device Identification Technique based on Remote GPU Fingerprinting, 2022. URL: https://arxiv.org/abs/2101.03793.
[8] W. Fuhl, N. I. Sanamrad, E. Kasneci, The Gaze and Mouse. Signal as additional Source for User Fingerprints in Browser, 2022. URL: https://arxiv.org/abs/2101.03793.
[9] E. Wustrow, S. Frolov, (University of Colorado Boulder), The use of TLS in Censorship Circumvention, doi:10.14722/ndss.2019.23511.
[10] B. Pelto, M. Vanamala, R. Dave, Your Identity is Your Behavior -- Continuous User Authentication based on Machine Learning and Touch Dynamics, 2022. URL: https://arxiv.org/abs/2305.09482.
[11] D. Moad, V. Sihag, G. Choudhary, Fingerprint defender: Defense against browser-based user tracking. In: I. You, H. Kim, TY., Youn, F. Palmieri, I. Kotenko (Eds.), Mobile Internet Security. MobiSec 2021, volume 1544 of Communications in Computer and Information Science, Springer, Singapore, 2021. doi: 10.1007/978-981-16-9576-6_17.
[12] M. Gabryel, K. Grzanek, Y. Hayashi, Browser Fingerprint Coding Methods Increasing the Effectiveness of User Identification in the Web Traffic, Journal of Artificial Intelligence and Soft Computing Research 10(4) (2020). doi: 10.2478/jaiscr-2020-0016.
[13] Login Data Set for Risk-Based Authentication, 2022. URL: https://www.kaggle.com/datasets/dasgroup/rba-dataset.
[14] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, H. Arshadf, State-of-the-art in artificial neural network applications: A surveyб Heliyon. 4(11):e00938 (2018). doi: 10.1016/j.heliyon.2018.e00938.
[15] A. Lheureux, Feed-forward vs feedback neural networks, 2022. URL: https://blog.paperspace.com/feed-forward-vs-feedback-neural-networks/.
[16] L.V. Sibruk, I.V. Zakutynskyi, Recurrent Neural Networks for Time Series Forecasting. Choosingthe best Architecture for Passenger Traffic Data. Automation and computer-integrated technologies 2(72) (2022) 38–44. doi: 10.18372/1990-5548.72.16941.
[17] GitHub, fingerprintjs/fingerprintjs: Browser fingerprinting library, 2023. URL: https://github.com/fingerprintjs/fingerprintjs.
[18] User Agents, 2022. URL: https://www.npmjs.com/package/user-agents.