





## Article

# Design of Cloud-Based Real-Time Eye-Tracking Monitoring and Storage System

Mustafa Can Gursesli <sup>1,2</sup>, Mehmet Emin Selek <sup>3</sup>, Mustafa Oktay Samur <sup>4</sup>, Mirko Duradoni <sup>2</sup>,  
Kyoungju Park <sup>5</sup>, Andrea Guazzini <sup>2,6</sup> and Antonio Lanata <sup>1,\*</sup>

<sup>1</sup> Department of Information Engineering, University of Florence, 50139 Florence, Italy; mustafacan.gursesli@unifi.it

<sup>2</sup> Department of Education, Literatures, Intercultural Studies, Languages and Psychology, University of Florence, 50135 Florence, Italy; mirko.duradoni@unifi.it (M.D.); andrea.guazzini@unifi.it (A.G.)

<sup>3</sup> Department of Mining Engineering, Istanbul Technical University, Istanbul 34467, Turkey; mehmeteminselek@gmail.com

<sup>4</sup> Department of Electrical and Electronics Engineering, Bilgi University, Istanbul 34060, Turkey; oktaysamr@gmail.com

<sup>5</sup> Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea; kypark@cau.ac.kr

<sup>6</sup> Centre for the Study of Complex Dynamics, University of Florence, 50019 Sesto Fiorentino, Italy

\* Correspondence: antonio.lanata@unifi.it

**Abstract:** The rapid development of technology has led to the implementation of data-driven systems whose performance heavily relies on the amount and type of data. In the latest decades, in the field of bioengineering data management, among others, eye-tracking data have become one of the most interesting and essential components for many medical, psychological, and engineering research applications. However, despite the large usage of eye-tracking data in many studies and applications, a strong gap is still present in the literature regarding real-time data collection and management, which leads to strong constraints for the reliability and accuracy of on-time results. To address this gap, this study aims to introduce a system that enables the collection, processing, real-time streaming, and storage of eye-tracking data. The system was developed using the Java programming language, WebSocket protocol, and Representational State Transfer (REST), improving the efficiency in transferring and managing eye-tracking data. The results were computed in two test conditions, i.e., local and online scenarios, within a time window of 100 seconds. The experiments conducted for this study were carried out by comparing the time delay between two different scenarios, even if preliminary results showed a significantly improved performance of data management systems in managing real-time data transfer. Overall, this system can significantly benefit the research community by providing real-time data transfer and storing the data, enabling more extensive studies using eye-tracking data.



**Citation:** Gursesli, M.C.; Selek, M.E.; Samur, M.O.; Duradoni, M.; Park, K.; Guazzini, A.; Lanata, A. Design of Cloud-Based Real-Time Eye-Tracking Monitoring and Storage System. *Algorithms* **2023**, *16*, 355. <https://doi.org/10.3390/a16070355>

Academic Editor: Frank Werner

Received: 19 June 2023

Revised: 13 July 2023

Accepted: 23 July 2023

Published: 24 July 2023

**Keywords:** data management; cloud computing; RESTful API; eye tracking; web portal



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

In recent decades, technology has become a crucial element of human life, leading to various innovative and convenient advancements across numerous fields, including health [1,2], entertainment [3,4], social media [5,6], physics [7,8], and chemistry [9,10]. While these innovations have positively impacted human life, they also demanded several technological requirements. These requirements, including computational power [11], Internet access [12], electricity [13], data [14], and other factors [15,16], have become increasingly crucial in both academia and industry sectors. Data and data management, in particular, became the center node for solving these technological challenges, and their relevance has been further increased by the growth of machine learning methods and AI

applications [14,17,18]. However, despite the need for huge data, research applications still lack suitable and effective data collection systems. Moreover, since many studies, especially those regarding a large part of the population, moved to mobile applications, real-time data became the strongest constraint to solve [19,20]. These conditions have led researchers to focus on improving and creating novel data collection systems to facilitate the technological advances in the research activity. These data collection systems are widely used in many studies on topics as different as brain signals [21], earthquakes [22], weather conditions [23], etc.

In this context, Representational State Transfer (REST), the most widely used web-based architecture in both the academic literature and industry, was introduced in 2000 as a Ph.D. thesis by Roy Fielding [24] for leading the design and development of the architecture of an Internet-scale distributed hypermedia system. It facilitates the caching of components to reduce user-perceived latency, enforce security, and encapsulate legacy systems [24]. REST employs the Hyper Text Transfer Protocol (HTTP) to enable communication between clients and servers. Its structure provides several advantages, including modifiability and statelessness, which enhance interoperability [25]. These advantages bring substantial benefits to the management of real-time data.

In addition to the solutions and limitations associated with real-time data management, it is widely recognized that the real-time management of diverse data types presents unique challenges due to their high density and rapid flow rates [26]. Eye-tracking data in particular present this complexity due to their highly dynamic and rapidly changing nature [27–29]. Furthermore, an eye-tracking pattern is an indirect measure of the complex biological system behind it, which requires high-cost computational methods for analysis with models, creating a major problem for the smooth real-time streaming of data.

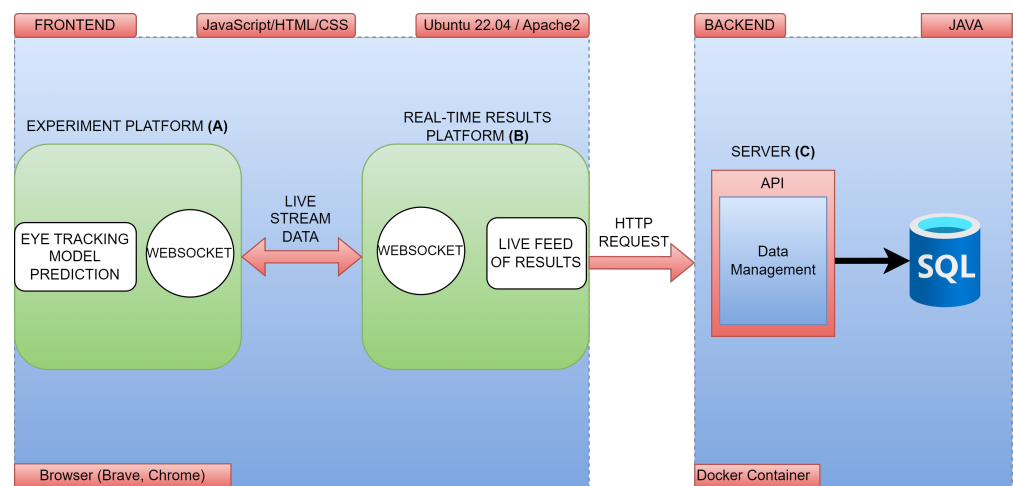
In this regard, eye tracking is a powerful research tool for studying various topics, such as marketing [30], attention [31], perception [32], psychopathology [33], computer vision [34], and decision making [35,36]. Eye tracking provides insight into the neural mechanisms at the base of exploring strategies of visual stimuli [37]. The eye-tracking technology greatly advanced in recent years, achieving greater precision and accuracy, even in real-world environments [34,38,39]. The history of eye tracking can be traced back to the late 1800s, with improvements in terms of comfort, wearability, and performance for a long-time measure of eye movements [36,40,41]. Since then, there have been important advancements that have led to the development of increasingly sophisticated eye-tracking systems [42–44].

Although physical eye-tracking devices have improved and become easier to use, the real-world employment of these devices is still not widespread due to their high cost [42–44]. This practical issue has prompted researchers to find different solutions, and many eye-tracking models using webcams have been developed [34,39,45]. Many of these models can be implemented locally on the user's devices and streamed to different platforms via Internet and web servers. Researchers integrate their models into web platforms to reach larger audiences and collect more data. Unfortunately, there is a gap in the literature regarding web-based streaming and storage systems that can be integrated with real-time eye-tracking models.

This study aims to introduce a system that allows the collection, processing, real-time streaming, and storage of eye-tracking data with REST architecture implementation. The manuscript is structured as follows: In Section 1, data necessity, REST, real-time data, and eye tracking are explained. Section 2 presents the materials and methods, the system design, the Representational State Transfer, the application programming interface, Web-Socket, the database server, Docker, WebGazer.js, the hardware implementation, and the experiment. In Section 3, the experimental results are detailed. Section 4 discusses the achieved experimental results compared with those presented in the literature. Lastly, Section 5 provides a conclusion of the entire study and possible future research directions.

## 2. Materials and Methods

The system's architecture consists of two software modules, i.e., the front-end and the back-end. The former is responsible for interfacing with the user, acquiring information, preprocessing, streaming live data, and transferring to the back-end. The latter is the invisible part and includes applications, servers, and databases. This section will display how our architecture articulates between these two modules. In our structure, three different interconnected platforms are designed to collect, process, stream, and store eye movements during an experimental session. These platforms are the experiment platform, the real-time results platform, and the database management platform (see Figure 1). The experiment and real-time results platforms are located in the front-end while the database management platform is located in the back-end. The front-end and back-end communicate through HTTP. Lastly, data gathered from this study were analyzed using Python Version 3.10.0 with Matplotlib library version 3.5.3. All details of the system and data flow are reported in the following sections.



**Figure 1.** System design components: experiment platform (A), real-time results platform (B), and server (C).

### 2.1. System Design

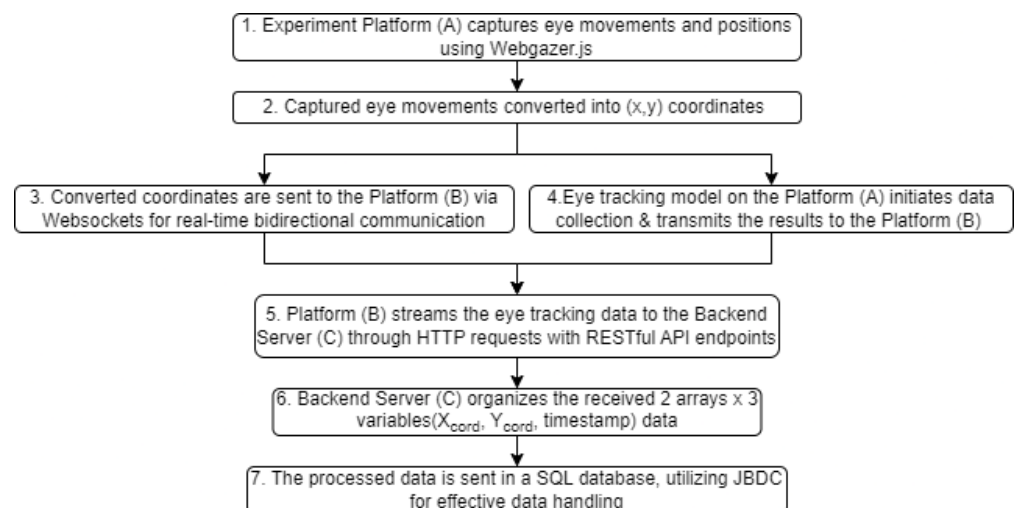
The system is designed with three components under two main modules (see Figure 1). The experimental platform (A) and the real-time results platform (B) are in the front-end module. In the back-end module, there is a server (C) component. The front-end development uses JavaScript, HyperText Markup Language (HTML), and Cascading Style Sheets (CSS), which allows the creation of a user-friendly interface. This interface displays the live data stream and the evaluation report of the eye-tracking system. HTML5 video tags are used to display participant's live video stream.

The front-end is specifically designed to integrate the eye-tracking model and provide a clear presentation of its live results. Moreover, data transfer between the experiment platform (A) and real-time results platform (B) is carried out via WebSocket. Subsequently, these data are transmitted to the back-end server via an HTTP request and stored in the database. The front-end is the preferred implementation location for eye-tracking models to ensure a more robust privacy strategy. Since sensitive data, such as eye-tracking information, is processed in this system, the aim is to perform all computations exclusively on the computer being used, without involving external servers or sources. This strategic approach is motivated by the fact that various models, including eye-tracking models found in the literature, perform computations on snapshots captured by the user's webcam [46, 47]. Processing these images on an external server is considered to introduce potential security vulnerabilities. Therefore, all computations are configured to take place only on the computer running the experiment, and only the results are transferred to other platforms and servers.

The back-end structure was built using Spring Boot, which is a framework of the Java language, providing a robust and scalable data processing and management platform. At the end of data collection on the front-end side, the collected data are sent to the back-end service by HTTP requests. The back-end service aims to process and manage data for database storage. In addition, the back-end provides a data management API that facilitates read and write operations to the SQL database.

Database management involves the use of an SQL database for the efficient storage and management of data. The database provides scalability and ease of retrieval and analysis of stored data. The back-end communicates with the database using Java Database Connectivity (JDBC), a Java API designed to access and manage databases. This seamless integration enables effective data handling within the system. Dockerization plays a major role in encapsulating the various components of the system. It involves separating the front-end, back-end, and database into separate Docker containers. Each container can be deployed independently, allowing for easy scalability based on the application's needs. Dockerization also provides a secure and isolated environment for each component, ensuring the stability and security of the overall system.

The system flow can be briefly described as follows: The experiment platform (A) captures the eye movements and positions of the subject through an embedded model, converts them into coordinates, and sends them to the real-time results platform (B). These two platforms communicate with each other via WebSockets and provide a data flow by constantly listening to exchanged messages from A to B and vice versa. The eye-tracking model placed on the experimental platform initiates data collection and its results are then transmitted to a real-time results platform and streamed to the back-end via HTTP requests. Following the end of the data collection session, the eye-tracking data, streamed instantaneously on the real-time results platform (B), are sent via HTTP request to the Server (C), which constitutes the last stage of the data flow in the back-end (See Figure 2).



**Figure 2.** System flowchart

## 2.2. Representational State Transfer (REST) and Application Programming Interface (API)

Representational State Transfer (REST) is designed to develop web services based on precise standards and limitations to grant an expandable and adaptable cross-data transaction over the Internet [24]. RESTful API (application programming interface) is an interpretation of the REST architecture that provides access to and actions on resources using HTTP. In a RESTful API, the server does not store data about the user between requests; instead, each request has all the data the server needs to process it. RESTful APIs follow a set of constraints, such as client-server architecture and a consistent interface, among others, to ensure that they are reliable, scalable, and easy to maintain [48,49]. REST has become popular among developers due to its simplicity and flexibility. In addition,

RESTful APIs have evolved into a standard for web services development and are actively used by many large companies such as Google, Twitter, etc.

### 2.3. WebSocket

WebSocket is a communication protocol pertaining to the application layer in the Transmission Control Protocol/Internet Protocol model (TCP/IP) [50]. Due to the popularity and prevalence of HTTP, WebSocket uses HTTP constructs for the initial connection between a client and a server [51] and provides persistent communication so that both the client and the server can send messages at any time. Compared to traditional real-time web communication, the WebSocket protocol saves a lot of network bandwidth and server resources, and the real-time performance is significantly improved [52]. It is helpful for real-time applications such as online games, financial trading platforms, eye tracking, and Internet of Things (IoT)-based applications that support server push technology [53,54].

### 2.4. Database Server (SQL)

The SQL (structured query language) is a fourth-generation declarative programming language for relational DBMSs (database management systems) and it is used to communicate with and manipulate databases [55]. The MySQL database stores and retrieves data via the REST API. The stored procedures and functions are designed as a security layer to perform operations that would receive queries from the API for SQL processing in the database [56].

There are many parameters to consider when evaluating database performance. In the next section, we will highlight the qualities that made SQL databases more suitable for this system over NoSQL databases. In particular, NoSQL databases outperform SQL databases regarding writing speed and scalability. NoSQL databases perform better when dealing with large scalability requirements and facilitating rapid data updates [57]. However, SQL databases better manage complex relationships and multiple client scenarios [57]. The characteristics of the SQL, structure, and capability to maintain data integrity make them suitable for scenarios involving relational data tables (such as the study carried out). Due to the anticipated availability of multiple user results and relational data in this system, the SQL database was chosen over NoSQL.

### 2.5. Docker

Docker is a technology that enables container virtualization, which can be compared to a highly efficient virtual machine due to its lightweight nature [58,59]. It is characterized by a modular architecture comprising multiple integral components that interact harmoniously to facilitate the process of “Containerization”. These components are articulated as follows: At the core of Docker is the Docker Engine, which provides the runtime environment for containers [59]. Docker Images, read-only templates that serve as container building blocks, utilize a layered file system and copy-on-write mechanism for efficient image management [59]. When a Docker Image is instantiated, it becomes a Docker Container, which offers a lightweight and secure execution environment [59]. Docker Containers can be easily created, started, stopped, and deleted, providing flexibility in managing application instances [60]. To facilitate image sharing and distribution, Docker Registries, such as Docker Hub, host a vast collection of prebuilt images [61]. Additionally, organizations can establish private registries tailored to their specific image requirements [61]. The modular architecture of Docker, along with its components, enables scalable and flexible application deployment across various environments.

### 2.6. WebGazer.js

WebGazer.js is a JavaScript-based eye-tracking algorithm. This algorithm allows the real-time display of eye-gaze locations on the web using webcams on notebooks and mobile phones [39,62]. This tool aims to utilize eye-tracking systems, which are currently only used in controlled environments and experiments, to enable people to use them in their daily



lives [39,62]. WebGazer.js consists of two core elements. These are a pupil detector and a gaze estimator. The pupil detector detects the position of the eye and pupil through the webcam. At the same time, the gaze estimator uses regression analysis to estimate where the individual is looking on the screen [39,62]. The gaze estimator applies a regression analysis through a calibration based on mouse clicks and mouse movements. Moreover, the pseudocode of WebGazer.js shows the algorithm details (See in Appendix A.1).

### 2.7. Hardware Implementation

In the system, three Docker virtual environments were used to perform experiments, stream real-time eye movements, and store data. In order to carry out online experiments, two separate physical AMD Central Processing Units (CPUs), 1 GB of Random Access Memory (RAM), and 25 GB of Solid State Disk (SSD) hardware were used for the front-end where the eye-tracking model runs and for live feed eye-tracking data. Furthermore, to store the data and manage the back-end, 2 physical Intel CPUs, 2 GB of RAM, and 25 GB of SSD hardware were used. The locations of the servers where the Dockers are used are located in Frankfurt, Germany for online experiments. The local experiments were conducted with Intel i5 8600k CPUs and 16 GB of RAM. Lastly, in both systems, eye-tracking data are collected in X and Y coordinates, while the data for time in seconds are stored in Year:Month:Day:Hour:Minute:Second:Millisecond.

### 2.8. Memory Management

Low-level programming languages, such as C, incorporate manual memory management features like *malloc()* and *free()* [63]. Conversely, JavaScript handles memory allocation automatically during object creation and frees it when those objects are no longer in use, through a process known as garbage collection. “Garbage Collection” in JavaScript plays a crucial role in determining which objects are necessary and which ones can be discarded [64]. It follows a cycle of memory release, where JavaScript identifies and marks objects that are no longer needed [65]. Specifically, within the *predictWebcam* function and the objects created within it, memory is allocated as required during each function call. Once the function produces an output, JavaScript performs the important task of marking and sweeping all the memory that will no longer be utilized, ensuring efficient memory management. In this system, we follow the garbage collection strategy.

### 2.9. Experiments

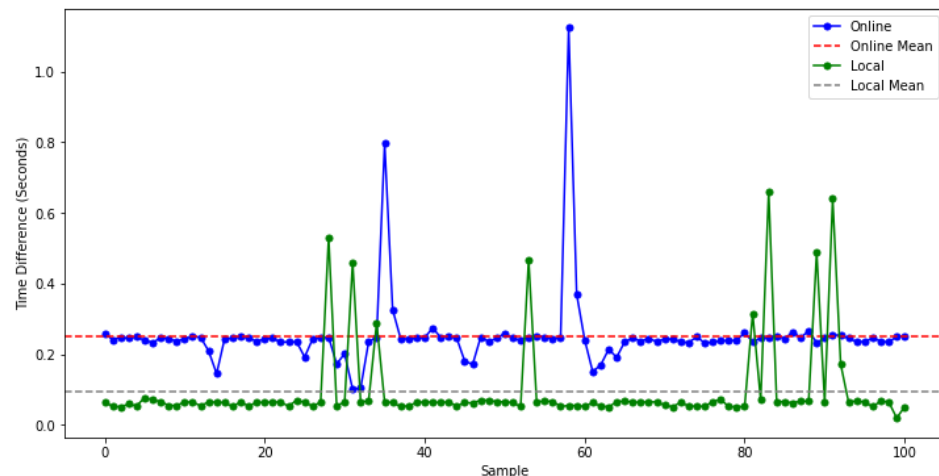
Experimental sessions were carried out to assess the reliability of the proposed system architecture, comparing two different scenarios: local implementation (LI) and online implementation (OI). The local scenario involved configuring the system on the local computer, while the online scenario consisted of configuring the system on the online server.

In order to measure the time delay of both scenarios (LI and OI), the timestamps of each platform were collected during a 100 s time window. The delay was calculated by subtracting the timestamp value of the experiment platform (A) from the timestamp received on the real-time results platform (B) ( $B-A$ ), (i.e., arrival time – starting time). It is of note that platform (A) sends data to platform (B) at the frequency of 1 HZ (see Figure 1). The *Console.log()* function was used to visualize data in the experiment. Specifically, *Console.log()* is a function that allows the data given into the function to be seen outside the code environment. This function allowed us to capture the precise timestamps indicating the arrival and starting time of data effectively.

Moreover, for the second experiment, 15 min of data were collected from the system at one-second intervals to understand how the system affects memory usage and how it changes over time while the eye-tracking model is performing real-life computations in the experiment platform. A *logMemoryUsage()* function was used to record the memory usage measurements in real-time. Specifically, the *logMemoryUsage()* function can be used for several purposes, such as analyzing the change in memory usage over time and detecting memory leaks or performance problems.

### 3. Results

In this study, a series of statistical analyses have been performed to evaluate the difference between the delays of LI and OI. In order to perform the analyses correctly, firstly, the Shapiro–Wilk test was applied to determine whether the delay data were normally distributed. According to the results of the Shapiro–Wilk test, both the LI delay data (Shapiro–Wilk test statistic = 0.370,  $p < 0.05$ ) and the OI delay data (Shapiro–Wilk test statistic = 0.322,  $p < 0.05$ ) did not fit a normal distribution. Time difference distributions are shown in Figure 3.



**Figure 3.** Comparison of time differences between the online and local systems.

Based on these results, it was concluded that parametric statistical tests could not be used and the Mann–Whitney U test, a non-parametric test, was preferred. The results of the Mann–Whitney U test showed a statistically significant difference between local and online latency ( $U = 794.0$ ,  $p < 0.05$ ). Table 1 shows the Mann–Whitney U test results.

**Table 1.** Mann–Whitney U test results regarding the local and online conditions.

Condition	U-Statistic	<i>p</i> -Value
Local vs. online	794.0	$3.317643 \times 10^{-25}$

According to descriptive statistics, the MAD value for the LI delay was 0.004, the median value was 0.064, the minimum value was 0.020, and the maximum value was 0.660. Similarly, the MAD value for the OI delay was 0.006, the median value was 0.244, the minimum value was 0.101, and the maximum value was 1.123. All the results of the descriptive statistics are shown in Table 2.

**Table 2.** Descriptive statistics.

Condition	MAD	Median	Min	Max
Local	0.004	0.064	0.020	0.660
Online	0.006	0.244	0.101	1.123

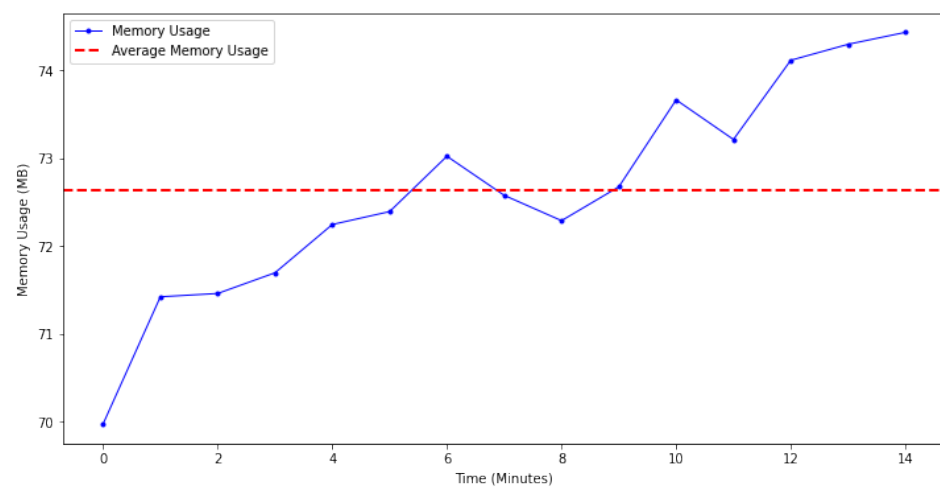
These findings indicate that there is a statistically significant difference between LI and OI delays and that there is a significant difference in their performance.

In addition, analysis of the memory usage data revealed interesting results (see Table 3). The average increase between seconds was measured as 0.0037 MB. The average memory usage during the session was measured as 72.62 MB with a minimum of 63.74 MB, and the maximum memory usage was 83.62. The standard deviation of memory usage was calculated at 3.48 MB.

**Table 3.** Memory usage statistics.

Conditions	Memory (Megabyte (MB))
Average increase between seconds	0.0037
Average memory usage	72.62
Minimum memory usage	63.74
Maximum memory usage	83.62
Standard deviation	3.48

Figure 4 shows an initial low level of memory usage that increases over time, with a steady increase over a period of time. Although there are occasional fluctuations, the average memory usage (red dashed line) is generally above the curve and shows a steadily increasing trend. These results show that the memory usage of the system varies over time and reaches a stable level over a period of time.



**Figure 4.** Diagram of memory usage: blue line, memory usage dynamic; red dotted line, memory usage average. Time axis is expressed in minutes

#### 4. Discussion

The demand for data has seen a substantial increase in recent years due to factors such as rapid technological advancements, growing interest in AI from both the private sector and researchers, and the proliferation of diverse research in the literature [66–69]. However, it is widely acknowledged that data collection systems, expected to keep up with these demands, are facing limitations. This study aims to develop a system that facilitates the data collection process for various studies, particularly in the academic domain, while simultaneously enabling the real-time observation and streaming of the collected data.

Presently, REST is extensively employed in academic research across various fields, including case generation [70], methodologies [71], biological data [72], machine learning [73], etc. Furthermore, prominent companies, like Google, Amazon, Twitter, and Reddit, also utilize this architecture. As part of this study, REST enables the instantaneous streaming of the collected data. However, to avoid restricting researchers solely to Internet-based usage, the system incorporates the Dockerization technique, allowing for local implementation. Consequently, tests were conducted in local and online (server-based) configurations. A significant difference was found between the time it took for the eye-tracking model data to reach the results page in the locally configured system compared to the same system configured online. Numerous performance bottlenecks, such as Internet latency [74], computer configuration [75], and server location [76], present considerable challenges that are difficult to mitigate. Although the latency experienced online is significantly higher than that of the local configuration, it is believed that the experimental online latency is not substantial enough for users to discern [77] (the delay values are shown in Figure 3).



The system presented in this study, which is based on several different techniques, serves the purpose of the real-time streaming and storage of eye-tracking data. However, it is crucial to highlight the flexibility of the proposed system, which can be adapted for collecting and analyzing other data types in different experimental settings. Several studies in the literature use Docker technology to build cloud platforms and integrate them into a variety of experiments, similar to the approach used in this current study. The use of Docker technology allows for the integration of AI and various models in studies. The system demonstrates a well-suited structure for numerous AI models in the literature. In particular, Shanti et al. (2022) successfully implemented facial emotion recognition using Convolutional Neural Networks [78]. In addition, Barillaro et al. (2022) presented a Deep Learning-based ECG signal classification model [79]. Similarly, Vryzas et al. (2020) focused on the task of speech emotion recognition, employing neural networks [80]. All these studies use models that are implemented using Docker technology and have substructures that can work in compatibility with the introduced system. Simultaneously, the system allows the real-time tracking of users' eye movements, enabling streaming over the Internet without being limited to a single task.

The memory consumption of the experimental system should also be highlighted. During the experiment, the memory consumption of the system slowly increased, putting a certain load on the computer used for the experiment. However, it is important to stress that this load is approximately 0.0037 MB per second and therefore does not have a noticeable impact on the overall performance. Nevertheless, in a scenario where the duration of the experiment is significantly longer, the potential load on the system should be carefully considered and the experiment should be structured to take this into account.

Furthermore, future studies need to examine a larger pool of participants and adopt more efficient memory management techniques. These improvements will contribute to a more thorough analysis of the system's capabilities and limitations, helping researchers to gain deeper findings and more reliable systems. Lastly, researchers should test the compatibility of the presented system with other structures and models, not only AI models (e.g., physiological data [81,82], and psychological tests [83,84]). In addition, the proposed architecture fosters strong collaboration between researchers adopting similar platforms, enabling an incredibly flexible data exchange and sharing. Data storage via the Internet is also expected to increase accessibility, thereby encouraging further research and discovery in various fields. However, it is important to recognize that for future implementations of this system, additional actions can be taken to increase the security of data storage. Examples of such actions include the integration of multi-factor authentication, one-time passwords, and other relevant security protocols [85,86].

## 5. Conclusions

This study reported on an approach to data collection and experimentation that demonstrates the intricacies of a multi-purpose system for both online and local applications. This study highlights the fundamental importance of data in scientific endeavors and calls for further exploration of alternative data collection techniques.

**Author Contributions:** Conceptualization, M.E.S., M.O.S., A.L., A.G., M.C.G., K.P., and M.D.; methodology, A.L. and M.C.G.; investigation, M.E.S., M.O.S., A.L., and M.C.G.; data curation, M.E.S., M.O.S., and M.C.G.; writing—original draft preparation, M.C.G., M.O.S., M.E.S., and M.D.; writing—review and editing, A.G., M.C.G., M.O.S., K.P., and A.L.; supervision, A.G., A.L., M.D., and K.P. A.G. and A.L. are equally responsible for this study. All authors have read and agreed to the published version of the manuscript.

**Funding:** Thanks to DigitalOcean (digitalocean.com) and Oliver Mensah for providing us with servers for tests and various other experiments.

**Data Availability Statement:** The data that support the findings of this study are available from the corresponding author, [A.L.], upon reasonable request.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A.

### Appendix A.1. Pseudocode of WebGazer.js

---

#### Algorithm A1: Starting WebGazer

---

```

1 Function startWebGazer:
2   Load WebGazer library;
3   Set up video and canvas elements on the webpage;

```

---



---

#### Algorithm A2: Starting Calibration

---

```

1 Function startCalibration:
2   Call initializeCalibration(), captureCalibrationData(), saveCalibrationData();
3   Display calibration instructions to the user;
4   Wait for the user to initiate the calibration process;
5   Call saveCalibrationData() # Set and save calibration process;

```

---



---

#### Algorithm A3: Capturing Calibration Data

---

```

1 Function captureCalibrationData:
2   Loop for a specified number of calibration points;
3     Display a calibration point on the screen;
4     Wait for the user to focus their gaze on the calibration point;
5     Store the recorded gaze data for the calibration point;

```

---



---

#### Algorithm A4: Saving Calibration Data

---

```

1 Function saveCalibrationData:
2   Save the captured calibration data for future use;
3   Call setCalibrationData();

```

---



---

#### Algorithm A5: Setting Calibration Data

---

```

1 Function setCalibrationData:
2   Set previously saved calibration data;

```

---



---

#### Algorithm A6: Starting Gaze Tracking

---

```

1 Function startGazeTracking:
2   Call initializeWebGazer();
3   Call getGazeData();
4   Begin real-time tracking of the user's gaze;
5   Display the gaze position on the screen;

```

---



---

#### Algorithm A7: Stopping Gaze Tracking

---

```

1 Function stopGazeTracking:
2   Stop tracking the user's gaze;
3   Clear the displayed gaze position on the screen;

```

---



---

#### Algorithm A8: Getting Gaze Data

---

```

1 Function getGazeData:
2   Retrieve the current gaze position data from the WebGazer library;
3   Return the gaze data;

```

---

**Algorithm A9: Example Usage**

```

1 Function ExampleUsage:
2   initializeWebGazer();
3   startCalibration();
4   captureCalibrationData();
5   saveCalibrationData();
6   // Later...;
7   initializeWebGazer();
8   loadCalibrationData();
9   startGazeTracking();
10  // During gaze tracking...;
11  gazeData = getGazeData();
12  // Utilize the gazeData for further processing or interaction;

```

**References**

1. Chaudhry, B.; Wang, J.; Wu, S.; Maglione, M.; Mojica, W.; Roth, E.; Morton, S.C.; Shekelle, P.G. Systematic review: Impact of health information technology on quality, efficiency, and costs of medical care. *Ann. Intern. Med.* **2006**, *144*, 742–752.
2. Buntin, M.B.; Burke, M.F.; Hoaglin, M.C.; Blumenthal, D. The benefits of health information technology: A review of the recent literature shows predominantly positive results. *Health Aff.* **2011**, *30*, 464–471.
3. Martucci, A.; Gursesli, M.C.; Duradoni, M.; Guazzini, A. Overviewing Gaming Motivation and Its Associated Psychological and Sociodemographic Variables: A PRISMA Systematic Review. *Hum. Behav. Emerg. Technol.* **2023**, *2023*, e5640258. <https://doi.org/10.1155/2023/5640258>.
4. Rauterberg, M. Positive Effects of Entertainment Technology on Human Behaviour. In Proceedings of the *Building the Information Society: IFIP International Federation for Information Processing 18th World Computer Congress Topical Sessions, Toulouse, France, 22–27 August 2004*; Jacquart, R., Ed.; Springer: New York, NY, USA, 2004; pp. 51–58. [https://doi.org/10.1007/978-1-4020-8157-6\\_8](https://doi.org/10.1007/978-1-4020-8157-6_8).
5. Duradoni, M.; Spadoni, V.; Gursesli, M.C.; Guazzini, A. Development and Validation of the Need for Online Social Feedback (NfOSF) Scale. *Hum. Behav. Emerg. Technol.* **2023**, *2023*, e5581492. <https://doi.org/10.1155/2023/5581492>.
6. Carr, C.T.; Hayes, R.A. Social media: Defining, developing, and divining. *Atl. J. Commun.* **2015**, *23*, 46–65.
7. Kadish, K.M.; Ruoff, R.S. *Fullerenes: Chemistry, Physics, and Technology*; John Wiley & Sons: New York, NY, USA, 2000; Google-Books-ID: SQRugQM4p9QC.
8. Nicollian, E.H.; Brews, J.R. *MOS (Metal Oxide Semiconductor) Physics and Technology*; John Wiley & Sons: New York, NY, USA, 2002; Google-Books-ID: IgUVEAAAQBAJ.
9. Noll, W. *Chemistry and Technology of Silicones*; Elsevier: Amsterdam, The Netherlands, 2012; Google-Books-ID: 5J3YS3dXA6kC.
10. Whistler, R.L.; BeMiller, J.N.; Paschall, E.F. *Starch: Chemistry and Technology*; Academic Press: New York, NY, USA, 2012; Google-Books-ID: pvAzqk2pAIsC.
11. Hwang, T. Computational power and the social impact of artificial intelligence. *arXiv* **2018**, arXiv:1803.08971.
12. Yaqoob, I.; Ahmed, E.; Hashem, I.A.T.; Ahmed, A.I.A.; Gani, A.; Imran, M.; Guizani, M. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wirel. Commun.* **2017**, *24*, 10–16.
13. SG Andrae, A. New perspectives on internet electricity use in 2030. *Eng. Appl. Sci. Lett.* **2020**, *3*, 19–31.
14. Williams, P.H.; Margules, C.R.; Hilbert, D.W. Data requirements and data sources for biodiversity priority area selection. *J. Biosci.* **2020**, *27*, 327–338. <https://doi.org/10.1007/BF02704963>.
15. Navajas, J.; Barsakcioglu, D.Y.; Eftekhari, A.; Jackson, A.; Constandinou, T.G.; Quiroga, R.Q. Minimum requirements for accurate and efficient real-time on-chip spike sorting. *J. Neurosci. Methods* **2014**, *230*, 51–64.
16. Chaudhary, N.; Weissman, D.; Whitehead, K.A. mRNA vaccines for infectious diseases: Principles, delivery and clinical translation. *Nat. Rev. Drug Discov.* **2021**, *20*, 817–838.
17. Vidgen, B.; Derczynski, L. Directions in abusive language training data, a systematic review: Garbage in, garbage out. *PLoS ONE* **2020**, *15*, e0243300. Publisher: Public Library of Science, <https://doi.org/10.1371/journal.pone.0243300>.
18. Raupach, M.R.; Rayner, P.J.; Barrett, D.J.; DeFries, R.S.; Heimann, M.; Ojima, D.S.; Quegan, S.; Schmutlious, C.C. Model–data synthesis in terrestrial carbon observation: Methods, data requirements and data uncertainty specifications. *Glob. Chang. Biol.* **2005**, *11*, 378–397. Available online: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2486.2005.00917.x> (accessed on 19 June 2023). <https://doi.org/10.1111/j.1365-2486.2005.00917.x>.
19. Farmer, A.; Gibson, O.; Hayton, P.; Bryden, K.; Dudley, C.; Neil, A.; Tarassenko, L. A real-time, mobile phone-based telemedicine system to support young adults with type 1 diabetes. *Inform. Prim. Care* **2005**, *13*, 171–177.
20. Gradl, S.; Kugler, P.; Lohmüller, C.; Eskofier, B. Real-time ECG monitoring and arrhythmia detection using Android-based mobile devices. In Proceedings of the 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, San Diego, CA, USA, 28 August–1 September 2012; IEEE: New York, NY, USA, 2012; pp. 2452–2455.
21. Teplan, M. Fundamentals of EEG measurement. *Meas. Sci. Rev.* **2002**, *2*, 1–11.

22. Boore, D.M.; Smith, C.E. Analysis of earthquake recordings obtained from the Seafloor Earthquake Measurement System (SEMS) instruments deployed off the coast of southern California. *Bull. Seismol. Soc. Am.* **1999**, *89*, 260–274.
23. Xue, M.; Wang, D.; Gao, J.; Brewster, K.; Droegemeier, K.K. The Advanced Regional Prediction System (ARPS), storm-scale numerical weather prediction and data assimilation. *Meteorol. Atmos. Phys.* **2003**, *82*, 139–170.
24. Fielding, R.T. *Architectural Styles and the Design of Network-Based Software Architectures*; University of California: Irvine, CA, USA, 2000.
25. Costa, B.; Pires, P.F.; Delicato, F.C.; Merson, P. Evaluating a Representational State Transfer (REST) architecture: What is the impact of REST in my architecture? In Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture, Sydney, Australia, 7–11 April 2014; IEEE: New York, NY, USA, 2014; pp. 105–114.
26. Cho, G.Y.; Lee, S.J.; Lee, T.R. An optimized compression algorithm for real-time ECG data transmission in wireless network of medical information systems. *J. Med. Syst.* **2015**, *39*, 1–8.
27. Kroner, A.; Senden, M.; Driessens, K.; Goebel, R. Contextual encoder–decoder network for visual saliency prediction. *Neural Netw.* **2020**, *129*, 261–270.
28. Skaramagkas, V.; Giannakakis, G.; Ktistakis, E.; Manousos, D.; Karatzanis, I.; Tachos, N.S.; Tripoliti, E.; Marias, K.; Fotiadis, D.I.; Tsiknakis, M. Review of eye tracking metrics involved in emotional and cognitive processes. *IEEE Rev. Biomed. Eng.* **2021**, *16*, 260–277.
29. Black, M.H.; Chen, N.T.; Iyer, K.K.; Lipp, O.V.; Bölte, S.; Falkmer, M.; Tan, T.; Girdler, S. Mechanisms of facial emotion recognition in autism spectrum disorders: Insights from eye tracking and electroencephalography. *Neurosci. Biobehav. Rev.* **2017**, *80*, 488–515.
30. Wedel, M.; Pieters, R. A review of eye-tracking research in marketing. In *Review of Marketing Research*; Emerald Group Publishing Limited: Bingley, UK, 2017; pp. 123–147.
31. Srivastava, N.; Nawaz, S.; Newn, J.; Lodge, J.; Velloso, E.; M. Erfani, S.; Gasevic, D.; Bailey, J. Are you with me? Measurement of Learners’ Video-Watching Attention with Eye Tracking. In Proceedings of the LAK21: 11th International Learning Analytics and Knowledge Conference, Irvine, CA, USA, 12–16 April 2021; Association for Computing Machinery: New York, NY, USA, 2021; LAK21, pp. 88–98. <https://doi.org/10.1145/3448139.3448148>.
32. Borys, M.; Plechawska-Wójcik, M. Eye-tracking metrics in perception and visual attention research. *EJMT*.
33. Iacono, W.G.; Lykken, D.T. Eye Tracking and Psychopathology: New Procedures Applied to a Sample of Normal Monozygotic Twins. *Arch. Gen. Psychiatry* **1979**, *36*, 1361–1369. <https://doi.org/10.1001/archpsyc.1979.01780120091011>.
34. Krafska, K.; Khosla, A.; Kellnhöfer, P.; Kannan, H.; Bhandarkar, S.; Matusik, W.; Torralba, A. Eye Tracking for Everyone. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2176–2184.
35. Fiedler, S.; Glöckner, A. The Dynamics of Decision Making in Risky Choice: An Eye-Tracking Analysis. *Front. Psychol.* **2012**, *3*, 335.
36. Holmqvist, K.; Nyström, M.; Andersson, R.; Dewhurst, R.; Jarodzka, H.; Van de Weijer, J. *Eye Tracking: A Comprehensive Guide to Methods and Measures*; OUP: Oxford, UK, 2011; Google-Books-ID: 5rIDPV1EoLUC.
37. Pfeiffer, U.J.; Vogeley, K.; Schilbach, L. From gaze cueing to dual eye-tracking: Novel approaches to investigate the neural correlates of gaze in social interaction. *Neurosci. Biobehav. Rev.* **2013**, *37*, 2516–2528. <https://doi.org/10.1016/j.neubiorev.2013.07.017>.
38. Duchowski, A.T. *Eye Tracking Methodology: Theory and Practice*; Springer: New York, NY, USA, 2017.
39. Papoutsaki, A.; Laskey, J.; Huang, J. SearchGazer: Webcam Eye Tracking for Remote Studies of Web Search. In Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval, Oslo, Norway, 7–11 March 2017; p. 26. <https://doi.org/10.1145/3020165.3020170>.
40. Aslin, R.N.; McMurray, B. Automated Corneal-Reflection Eye Tracking in Infancy: Methodological Developments and Applications to Cognition. *Infancy* **2004**, *6*, 155–163. Available online: [https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15327078in0602\\_1](https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15327078in0602_1) (accessed on 19 June 2023). [https://doi.org/10.1207/s15327078in0602\\_1](https://doi.org/10.1207/s15327078in0602_1).
41. Marino, J. Reading Screens: What Eye Tracking Tells Us about the Writing in Digital Longform Journalism. *Lit. J. Stud.* **2016**, *8*.
42. Niehorster, D.C.; Hessels, R.S.; Benjamins, J.S. GlassesViewer: Open-source software for viewing and analyzing data from the Tobii Pro Glasses 2 eye tracker. *Behav. Res. Methods* **2020**, *52*, 1244–1253. <https://doi.org/10.3758/s13428-019-01314-1>.
43. Kortman, B.; Nicholls, K. Assessing for Unilateral Spatial Neglect Using Eye-Tracking Glasses: A Feasibility Study. *Occup. Ther. Health Care* **2016**, *30*, 344–355. <https://doi.org/10.1080/07380577.2016.1208858>.
44. Mele, M.L.; Federici, S. Gaze and eye-tracking solutions for psychological research. *Cogn. Process.* **2012**, *13*, 261–265. <https://doi.org/10.1007/s10339-012-0499-z>.
45. Lu, F.; Sugano, Y.; Okabe, T.; Sato, Y. Head pose-free appearance-based gaze sensing via eye image synthesis. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, 11–15 November 2012; pp. 1008–1011, ISSN: 1051-4651.
46. Xu, P.; Ehinger, K.A.; Zhang, Y.; Finkelstein, A.; Kulkarni, S.R.; Xiao, J. Turkergaze: Crowdsourcing saliency with webcam based eye tracking. *arXiv* **2015**, arXiv:1504.06755.
47. Papoutsaki, A.; Sangkloy, P.; Laskey, J.; Daskalova, N.; Huang, J.; Hays, J. WebGazer: Scalable Webcam Eye Tracking Using User Interactions.

48. Wang, S.; Keivanloo, I.; Zou, Y. How do developers react to restful api evolution? In Proceedings of the Service-Oriented Computing: 12th International Conference, ICSOC 2014, Paris, France, 3–6 November 2014; Proceedings 12; Springer: New York, NY, USA, 2014; pp. 245–259.
49. Richardson, L.; Ruby, S. *RESTful Web Services*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2008. Google-Books-ID: XUaEr-akHsoAC.
50. Berners-Lee, T.J. *Information Management: A proposal*; Technical Report; CERN: Geneva, Switzerland, 1989.
51. Cassetti, O. Websockets and their integration in enterprise networks. *CiteSeer* **2011**.
52. Hu, Y.; Cheng, W. Research and implementation of campus information push system based on WebSocket. In Proceedings of the 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, China, 24–26 November 2017; pp. 1–6. <https://doi.org/10.1109/ISKE.2017.8258720>.
53. Soewito, B.; Christian.; Gunawan, F.E.; Diana.; Kusuma, I.G.P. WebSocket to Support Real Time Smart Home Applications. *Procedia Comput. Sci.* **2019**, *157*, 560–566. <https://doi.org/10.1016/j.procs.2019.09.014>.
54. Hale, M. Eyestream: An Open WebSocket-based Middleware for Serializing and Streaming Eye Tracker Event Data from Gazepoint GP3 HD Research Hardware. *J. Open Source Softw.* **2019**, *4*, 1620. <https://doi.org/10.21105/joss.01620>.
55. Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* **1970**, *13*, 377–387.
56. Kern, C.; Kesavan, A.; Daswani, N. *Foundations of Security: What Every Programmer Needs to Know*; Springer: New York, NY, USA, 2007.
57. Khan, W.; Kumar, T.; Zhang, C.; Raj, K.; Roy, A.M.; Luo, B. SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review. *Big Data Cogn. Comput.* **2023**, *7*, 97.
58. Anderson, C. Docker [Software engineering]. *IEEE Softw.* **2015**, *32*, 102–c3. <https://doi.org/10.1109/MS.2015.62>.
59. Martin, J.P.; Kandasamy, A.; Chandrasekaran, K. Exploring the support for high performance applications in the container runtime environment. *Hum.-Centric Comput. Inf. Sci.* **2018**, *8*, 1–15.
60. De Benedictis, M.; Liroy, A. Integrity verification of Docker containers for a lightweight cloud environment. *Future Gener. Comput. Syst.* **2019**, *97*, 236–246.
61. Chamoli, S. Docker Security: Architecture, Threat Model, and Best Practices. In Proceedings of the Soft Computing: Theories and Applications: Proceedings of SoCTA 2020; Springer: New York, NY, USA, 2021; Volume 2, pp. 253–263.
62. Slim, M.S.; Hartsuiker, R.J. Moving visual world experiments online? A web-based replication of Dijkgraaf, Hartsuiker, and Duyck (2017) using PCIBex and WebGazer.js. *Behav. Res. Methods* **2022**, 1–19. <https://doi.org/10.3758/s13428-022-01989-z>.
63. Chen, X.; Slowinska, A.; Bos, H. Who allocated my memory? Detecting custom memory allocators in C binaries. In Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, Germany, 14–17 October 2013; IEEE: New York, NY, USA, 2013; pp. 22–31.
64. Pienaar, J.A.; Hundt, R. JSWhiz: Static analysis for JavaScript memory leaks. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Montreal, QC, Canada, 25 February–1 March 2013; IEEE: New York, NY, USA, 2013; pp. 1–11.
65. Degenbaev, U.; Eisinger, J.; Hara, K.; Hlopko, M.; Lippautz, M.; Payer, H. Cross-component garbage collection. *Proc. ACM Program. Lang.* **2018**, *2*, 1–24.
66. Das, S.; Nayak, G.K.; Saba, L.; Kalra, M.; Suri, J.S.; Saxena, S. An artificial intelligence framework and its bias for brain tumor segmentation: A narrative review. *Comput. Biol. Med.* **2022**, *143*, 105273.
67. Wilson, C. Public engagement and AI: A values analysis of national strategies. *Gov. Inf. Q.* **2022**, *39*, 101652.
68. Lee, J.C.; Chen, X. Exploring users' adoption intentions in the evolution of artificial intelligence mobile banking applications: The intelligent and anthropomorphic perspectives. *Int. J. Bank Mark.* **2022**, *40*, 631–658.
69. Dogan, M.E.; Goru Dogan, T.; Bozkurt, A. The use of artificial intelligence (AI) in online learning and distance education processes: A systematic review of empirical studies. *Appl. Sci.* **2023**, *13*, 3056.
70. Arcuri, A. RESTful API automated test case generation. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 25–29 July 2017; IEEE: New York, NY, USA, 2017; pp. 9–20.
71. Ehsan, A.; Abuhaliqa, M.A.M.; Catal, C.; Mishra, D. RESTful API testing methodologies: Rationale, challenges, and solution directions. *Appl. Sci.* **2022**, *12*, 4369.
72. Miller, M.A.; Schwartz, T.; Pickett, B.E.; He, S.; Klem, E.B.; Scheuermann, R.H.; Passarotti, M.; Kaufman, S.; O'Leary, M.A. A RESTful API for access to phylogenetic tools via the CIPRES science gateway. *Evol. Bioinform.* **2015**, *11*, EBO-S21501.
73. Gossett, E.; Toher, C.; Oses, C.; Isayev, O.; Legrain, F.; Rose, F.; Zurek, E.; Carrete, J.; Mingo, N.; Tropsha, A.; et al. AFLOW-ML: A RESTful API for machine-learning predictions of materials properties. *Comput. Mater. Sci.* **2018**, *152*, 134–145.
74. Briscoe, B.; Brunstrom, A.; Petlund, A.; Hayes, D.; Ros, D.; Tsang, J.; Gjessing, S.; Fairhurst, G.; Griwodz, C.; Welzl, M. Reducing internet latency: A survey of techniques and their merits. *IEEE Commun. Surv. Tutorials* **2014**, *18*, 2149–2196.
75. Henning, J.L. SPEC CPU2000: Measuring CPU performance in the new millennium. *Computer* **2000**, *33*, 28–35.
76. Charyyev, B.; Arslan, E.; Gunes, M.H. Latency comparison of cloud datacenters and edge servers. In Proceedings of the GLOBECOM 2020—2020 IEEE Global Communications Conference, Taipei, Taiwan, 7–11 December 2020; IEEE: New York, NY, USA, 2020; pp. 1–6.
77. Stetson, C.; Cui, X.; Montague, P.R.; Eagleman, D.M. Motor-sensory recalibration leads to an illusory reversal of action and sensation. *Neuron* **2006**, *51*, 651–659.



78. Shanthi, N.; Stonier, A.A.; Sherine, A.; Devaraju, T.; Abinash, S.; Ajay, R.; Arul Prasath, V.; Ganji, V. An integrated approach for mental health assessment using emotion analysis and scales. *Healthc. Technol. Lett.* **2022**.
79. Barillaro, L.; Agapito, G.; Cannataro, M. Edge-based Deep Learning in Medicine: Classification of ECG signals. In Proceedings of the 2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Las Vegas, NV, USA, 6–8 December 2022; IEEE: New York, NY, USA, 2022; pp. 2169–2174.
80. Vryzas, N.; Vrysis, L.; Matsiola, M.; Kotsakis, R.; Dimoulas, C.; Kalliris, G. Continuous speech emotion recognition with convolutional neural networks. *J. Audio Eng. Soc.* **2020**, *68*, 14–24.
81. Shu, Y.S.; Chen, Z.X.; Lin, Y.H.; Wu, S.H.; Huang, W.H.; Chiou, A.Y.C.; Huang, C.Y.; Hsieh, H.Y.; Liao, F.W.; Zou, T.F.; et al. 26.1 A 4.5 mm<sup>2</sup> Multimodal Biosensing SoC for PPG, ECG, BIOZ and GSR Acquisition in Consumer Wearable Devices. In Proceedings of the 2020 IEEE International Solid-State Circuits Conference-(ISSCC), San Francisco, CA, USA, 16–20 February 2020; IEEE: New York, NY, USA, 2020; pp. 400–402.
82. Soufineyestani, M.; Dowling, D.; Khan, A. Electroencephalography (EEG) technology applications and available devices. *Appl. Sci.* **2020**, *10*, 7453.
83. Li, X.; Liu, Y.; Mao, J.; He, Z.; Zhang, M.; Ma, S. Understanding reading attention distribution during relevance judgement. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Torino, Italy, 22–24 October 2018; pp. 733–742.
84. Cox, W.M.; Fadardi, J.S.; Pothos, E.M. The addiction-stroop test: Theoretical considerations and procedural recommendations. *Psychol. Bull.* **2006**, *132*, 443.
85. Karie, N.M.; Kebande, V.R.; Ikuesan, R.A.; Sookhak, M.; Venter, H.S. Hardening SAML by Integrating SSO and Multi-Factor Authentication (MFA) in the Cloud. In Proceedings of the 3rd International Conference on Networking, Information Systems & Security, Marrakech, Morocco, 31 March–2 April 2020; pp. 1–6.
86. Bruzgiene, R.; Jurgilas, K. Securing remote access to information systems of critical infrastructure using two-factor authentication. *Electronics* **2021**, *10*, 1819.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.