ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

UDC 004.946:004.453.4 DOI: https://doi.org/10.30837/2522-9818.2026.2.052

Heorhii Kuchuk, Mykyta Matvieiev, Nataliia Kosenko, Dmytro Lysytsia, Andrii Levchenko

EVALUATING THE PERFORMANCE OF A WEBAR APPLICATION'S SCENE LOADING ON A MOBILE DEVICE

Subject of the study: the patterns of CPU resource allocation and changes in browser engine timing metrics during the execution of WebAR scene initialization stages. **Object of study**: the process of initializing and loading 3D content in a client-side WebAR application on a mobile device. **The aim of the study** is to develop an approach for comprehensive performance evaluation and to identify patterns in the distribution of computational load during the initialization of WebAR applications using deep browser profiling tools. **Objectives.** Develop a mathematical model of WebAR application operation during the scene initialization phase and decompose the loading process into key stages with an assessment of their time costs. Conduct empirical profiling, identify peak CPU loads and blocking tasks, and formulate well-founded directions for further performance optimization. **Methods.** Performance was investigated through empirical performance testing of the WebAR application during its most resource-intensive stage – the initialization and loading of a 3D scene. Data collection regarding CPU thread load within the context of a web page was performed using the Chrome DevTools developer tools. A Google Pixel 8 mobile device served as the hardware environment for the experiments, and the statistical reliability of the results was ensured by a 10-fold iteration of each test scenario. **Research results.** The developed mathematical model formalizes the process of loading a WebAR application scene and allows for the assessment of CPU load at each stage. Based on the analysis of the time profile obtained using the improved mathematical model and Chrome DevTools, four loading stages were identified: initialization of an empty scene, loading a 3D object from the server, model parsing, and model rendering. A nonlinear nature of the CPU load was established. The parsing stage proved to be the most resource-intensive, during which peak CPU load and the longest blocking task were recorded, caused by the synchronous deserialization of binary data. **Conclusions.** An approach is proposed to improve the performance of loading a WebAR application scene on a mobile device by balancing the CPU load. The study confirmed that the critical phase of initialization is the parsing of the 3D model. Given the simulation results and empirical studies conducted, a promising direction is the implementation of incremental loading and processing of 3D content.

Keywords: web; augmented reality; WebAR application; performance evaluation; parsing; rendering; Angular.

1. Introduction

Digital technologies have become an integral part of modern society, transforming the ways we communicate, learn, work, and consume information [1]. One of the most dynamic and promising technologies is augmented reality (AR), which combines virtual digital content with the physical environment in real time [2]. The integration of virtual objects into real space provides a new level of interactivity and immersion, significantly expanding the possibilities for user interaction with information compared to traditional multimedia tools [3].

WebAR technology has seen particular development; it implements AR functionality without the need to install separate applications, providing access to augmented reality directly through a web browser [4]. This approach lowers barriers to entry for users and simplifies the implementation of AR solutions for businesses [5]. Today, WebAR is actively used in e-commerce, medicine, education, and the entertainment industry, contributing to increased user engagement and the effectiveness of digital services [6]. The growing number of scientific studies and practical implementations in this field indicates the emergence of a trend in which web resources with integrated AR capabilities may become the standard for user interaction with digital content [7].

WebAR has a number of significant advantages, but at the same time is characterized by increased computational load on the web application and the client device [8]. The fundamental contradiction lies in the fact that basic web technologies were created primarily for working with 2D graphics. In contrast, WebAR requires continuous execution of resource-intensive operations: 3D graphics processing, rendering, and real-time spatial tracking [9]. Implementing these processes through standard browser mechanisms leads to reduced performance, manifested in a drop in frame rate, increased latency, blocking of the main execution thread, and overall inefficient use of hardware resources [10]. According to research, such technical and interface barriers account for 64% of all issues preventing users

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

from performing their intended actions. As a result, system responsiveness and the smoothness of interaction with AR content deteriorate, negatively impacting the user experience. The implementation of optimization solutions can significantly reduce these shortcomings; however, developing effective optimization strategies is a complex and multifaceted task [11]. A key prerequisite for solving this problem is an objective and comprehensive performance evaluation that takes into account the characteristics of the browser engine, the limitations of mobile devices, and the specifics of how 3D graphics function in a web environment [12].

Website performance evaluation is based on the analysis of a set of factors that determine its operational efficiency, which are commonly summarized under the concept of web metrics [13]. These include page load time, interface response speed, content display stability, and other indicators of user experience quality. One of the key characteristics is page load time, which encompasses the processes of fetching, processing, and rendering HTML, JavaScript, and CSS resources, as well as graphical elements. Although external factors – such as network latency and connection bandwidth – influence the overall loading time, internal computational processes are of decisive importance for WebAR applications [14]. These include the duration of JavaScript code parsing and execution, 3D scene initialization, shader compilation, loading and processing of 3D models, as well as the efficiency of CPU resource utilization. The most resource-intensive stage is scene initialization, during which the browser simultaneously processes significant amounts of geometric and texture data, configures the graphics pipeline, and synchronizes the video stream from the camera. It is at this stage that there is an increased risk of the main thread blocking and the interface "freezing", which critically affects the user's perception of the service [15]. In this regard, systematic and objective performance evaluation becomes particularly relevant as a necessary condition for improving the efficiency and stability of WebAR applications.

2 Analysis of Recent Studies and Publications

A significant portion of current research focuses primarily on the network and time-based performance characteristics of web resources. For example, in [16], a framework based on machine learning methods is proposed for predicting the performance of web pages. Within this model, quality is assessed based on a set of 16 metrics, dominated by temporal and network indicators: server response time, load duration, page size, and the number of HTTP requests. Despite taking into account indicators such as *Start Render Time* and *Time to Interactive*, the proposed methodology does not provide for in-depth profiling of computational resource consumption, particularly CPU load during the active phase of the application [15]. A similar approach can be seen in the work of *Ghattas and Sartawi* (2020), where the evaluation of web resource quality is based primarily on loading speed and layout stability metrics, without accounting for the computational cost of executing client-side scripts and complex interactive scenarios [6]. *Hussein et al.* (2023) investigated the impact of WebAR on the academic performance of elementary school students. During the technical validation of the AR.js and MyWebAR platforms, the authors used the GTmetrix toolkit, which is primarily focused on evaluating page load speed and network parameters. Thus, in this study, performance was interpreted primarily as a measure of content delivery, while an aspect critical to mobile learning – the CPU load during the execution of an AR scene – was overlooked.

A separate group consists of studies dedicated to functional accuracy. For instance, in [17], an evaluation of the WebXR Device API's performance was conducted, focusing on the accuracy of spatial tracking and the stability of virtual anchor binding through the analysis of trajectory errors and image differences. The only real-time performance metric was frame rate (FPS), which was used to confirm the smoothness of the visualization. However, the study lacks data on the impact on CPU load or device power consumption.

Attempts to quantify hardware costs are presented in [18], which analyzes CPU and RAM usage for different types of markers. Although the authors recorded specific CPU load levels, their methodology relied on the use of the *adb shell top* command. This approach provides only general aggregated data about the process, making it difficult to determine the contribution of individual functions, such as tracking, rendering, or video processing.

The practical aspect of deploying WebAR applications under conditions of limited hardware resources is examined in [19]. The authors implemented a contactless information system for a library based on the ar.js, three.js, and A-Frame libraries. To quantitatively assess performance on budget mobile devices supporting WebGL and WebRTC, four

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

variables were used: frame rate, animation frame request execution time, total load time, and the number of entity objects on the scene. Although this work directly considers the load time metric and analyzes rendering smoothness, the evaluation is conducted at the macro level. The lack of detailed browser profiling tools prevents an assessment of the load distribution across the processor's computational threads during scene initialization, which significantly limits the possibilities for targeted application optimization.

An analysis of the current state of research in the field of web performance evaluation reveals a significant imbalance in the selection of test objects and methods. According to systematic reviews, the vast majority of scientific works focus on the analysis of static content using basic macro-level metrics – page load speed, network latency [20]. However, for WebAR applications, this approach proves incapable of objectively reflecting the actual state of the system. The specific nature of WebAR requires continuous interaction with hardware, video stream processing, and 3D graphics rendering [21]. Existing testing methods do not provide a transparent assessment of the browser engine's internal processes: they do not allow for isolating the computational costs of JavaScript, graphics rendering, and memory management [22]. Although there is a wide range of analytical tools for evaluating web performance, the vast majority of them focus on network and macro-level metrics. Since such utilities are unable to detail the load distribution within the main thread and isolate the execution of JavaScript logic from rendering processes, the most appropriate and justified method for solving this problem is the use of instrumental browser profiling. This confirms the relevance of developing specialized approaches to profiling WebAR applications directly during the scene initialization phase.

3 Problem Statement

The goal of this work is to develop an approach for comprehensive performance evaluation and to identify patterns in the distribution of computational load during the initialization of WebAR applications using deep browser profiling tools. The study aims to quantitatively determine the CPU load on mobile devices, as well as to identify bottlenecks in the processes of loading, processing, and initial rendering of AR content. Only this level of instrumental detail allows for the acquisition of objective micro-level data, which constitutes the key value of the study, as it forms a single reliable foundation for developing effective architectural strategies for optimizing application performance.

The scientific novelty of the study lies in the further development of a mathematical model of the WebAR application scene loading process on a mobile device. Unlike existing macro-level approaches, this work is the first to perform a deep decomposition of browser engine microprocesses. The proposed approach allows for an isolated assessment of the impact of the stages of spatial geometry parsing, direct rendering, and memory management on the blocking of the main execution thread. The level of detail achieved makes it possible to scientifically prove the nature of critical system overloads in a single-threaded environment and to formulate well-founded architectural strategies for optimizing the processing of three-dimensional data. **The object of this study is** the process of initializing and loading 3D content in a client-side WebAR application on a mobile device. **The subject of the study** is the patterns of CPU resource allocation and changes in browser engine timing metrics during the execution of WebAR scene loading stages.

4 Mathematical Model of the Scene Loading Process for a WebAR Application on a Mobile Device

4.1 Formalization of the Process

The mathematical formalization of 3D scene initialization in a WebAR application describes how the system transitions from camera and sensor data to the construction of a virtual scene in real-world coordinates.

In WebAR, several coordinate systems are typically used: the world coordinate system $CS \cdot W$; the camera coordinate system $CS \cdot C$; $CS \cdot O$; $CS \cdot S$. Transformations between them are defined by corresponding matrices.

When constructing the camera model, the position of a scene point in the world coordinate system is defined by a vector

$$P_W = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \quad (1)$$

The conversion to the camera system is performed using the following transformation [23]:

$$P_C = T_{CW} \cdot P_W, \quad (2)$$

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

where $$T_{CW} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}, \quad t = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}, \tag{3}$$

where t is the spatial displacement vector, and R is a 3×3 rotation matrix that satisfies the following requirements:

$$R \in \mathrm{SO}(3), \quad \det R = 1, \quad R^T \cdot R = I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

When generating a projection onto the screen plane, a matrix of the camera's internal parameters is used:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{5}$$

where f_x, f_y – focal lengths; c_x, c_y – image center coordinates.

In that case, the projection is calculated as follows [24]:

$$P = (u, v) = K \cdot (R|t) \cdot P_W, \tag{6}$$

where u, v – the coordinates of the corresponding pixel.

During the initialization phase, the following must be determined: the camera's position; the scene plane; the scale; and the initial anchor point – that is, a point in real space relative to which the AR system places and tracks virtual objects.

Camera pose refers to the camera's position and orientation in space, which are defined by a rotation matrix and a translation vector

$$\Theta = (R, t) \ . \tag{7}$$

The system estimates this pose by analyzing images from the camera and tracking distinctive points or markers in the environment. To do this, it uses Simultaneous Localization and Mapping, Visual Odometry, and marker tracking methods, which help determine the camera's motion and coordinates in real time.

The model matrix describes how a 3D object is positioned in the scene: where it is located, how it is oriented, and what its dimensions are. It is formed as the product of the corresponding matrices [25]:

$$M = T \cdot R \cdot S, \tag{8}$$

where the T matrix controls the object's translation, the R matrix controls its rotation, and the S matrix controls its scaling. First, the object is scaled, then rotated, and finally moved to the desired position in the scene. The resulting matrix is used by the graphics engine to correctly render the model in 3D space.

The final scene in 3D graphics is formed by sequentially applying several matrices to the object's vertex coordinates.

The full transformation is described by the central equation of WebAR visualization:

$$P_{clip} = P \cdot V \cdot M \cdot P_{local}, \tag{9}$$

where the M matrix transforms the vertex from the model's local coordinates to scene coordinates, the V matrix accounts for the camera's position and orientation, transforming the scene into the observer's coordinate system, and the P matrix projects the 3D scene onto a 2D screen, applying either a perspective or orthographic projection. After all these transformations, the screen coordinates of the vertex are obtained, which are used to render the final image.

Note that in markerless WebAR, the system first analyzes the surrounding space and attempts to find flat surfaces, such as a floor, table, or wall [26]. A plane is described by the equation

$$ax + by + cz + d = 0, \tag{10}$$

where the coefficients determine its position and orientation in space, and the normal to the surface

$$\vec{n} = (a, b, c) \tag{11}$$

shows the direction perpendicular to this plane.

Thanks to this normal vector, the system understands how to correctly orient the WebAR object relative to the surface. For example, a virtual chair can be automatically placed flat on the floor rather than at an angle. This method of surface detection is used in modern WebAR systems without special markers.

In marker-based WebAR, the process begins with the device's camera detecting a special marker [27]. After that, the algorithm finds the marker's key points or corners in the image. Based on the correspondence between the marker's points and their real-world geometry, the homography matrix is calculated using the following formula:

$$H = K \cdot \left(R + \frac{t \cdot n^T}{d} \right), \tag{12}$$

where d is the distance to the plane.

The resulting homography matrix describes the transformation between the marker plane and the camera image plane, allowing the system to determine the camera's pose and the marker's exact position in space. After calculating the homography, AR objects

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

are stably tied to the marker and move correctly with it in real time.

During the operation of AR systems, object coordinates may "jitter" slightly due to camera noise and tracking errors. To make the motion more stable, temporal filtering or data smoothing is used.

The mathematical model under development uses the exponential smoothing method [28]:

$$\hat{x}_k = \beta \cdot x_k + (1-\beta) \cdot \hat{x}_{k-1}, \quad (13)$$

where x_k – a new position or angle measurement, and $\hat{x}_k$ –updated smoothed value. The coefficient β determines how much the system trusts the new data compared to previous values. As a result, AR objects move more smoothly and stably, without sudden jumps or flickering.

After determining the camera position, constructing the transformation matrices, and stabilizing the tracking, the system generates a complete WebAR scene. Formally, such a scene can be described as a set

$$\aleph = \left\{ Q_i, C, L, T, \tilde{I} \right\}. \quad (14)$$

The component Q_i represents all 3D objects displayed in augmented reality. These can include furniture models, animations, text, or interactive elements. The component C describes the user's camera and its position in space. It is the camera that defines the viewpoint from which the scene is rendered. The element L is responsible for scene lighting and affects the realism of object rendering. For example, the system can adjust brightness or shadows according to the lighting of the real environment. The set T contains all object transformations, including translation, rotation, and scaling. These transformations are implemented through the model, view, and projection matrices used during scene rendering. The component $\tilde{I}$ describes user interactions with the WebAR environment. Such interactions include screen touches, gestures, device movement, or selecting AR objects in real time.

4.2 Performance Analysis

To analyze scene loading performance, it is advisable to examine the main stages of loading and preparing a WebAR application, as these stages determine the startup speed and smoothness of the scene. Since the maximum load on the application occurs during initialization and scene loading, the study focuses exclusively on four key stages: environment initialization, resource loading, data parsing, and scene rendering.

The initialization stage includes launching the WebAR engine, creating the scene, configuring the camera, and connecting tracking modules. During loading, the system retrieves 3D models, textures, scripts, and other necessary resources.

During the parsing stage, the received data is processed and prepared for use by the graphics engine. This is followed by rendering, i.e., the construction of the final image and its display on the user's screen. For each of these stages, time characteristics, memory usage, and the load on the processor or graphics adapter can be determined. Thus, the performance of a WebAR application can be formalized as a set of key hardware and software parameters:

$$W_{perf} = (CPU, GPU, NET, MEM, FPS). \quad (15)$$

The *CPU* component characterizes the processor load that arises during the execution of computer vision algorithms, tracking, sensor data processing, and application logic. It is the central processing unit that performs most of the operations related to determining the camera's position, analyzing frames, finding keypoints, and calculating scene transformations. In markerless WebAR, the CPU load increases significantly due to the constant real-time execution of Simultaneous Localization and Mapping (SLAM) and Visual Odometry algorithms. High CPU load can lead to frame processing delays, overheating of the mobile device, and reduced stability of the AR scene.

The *GPU* component describes the load on the graphics processor during the rendering of 3D objects, lighting, textures, and animations. This component is responsible for constructing the final scene image and ensuring the visual quality of augmented reality. The more complex the models and effects used in the scene, the higher the load on the graphics adapter [29]. However, in most WebAR applications, the *GPU* primarily performs standard graphics operations, while the main tracking computations are performed by the *CPU* component.

The *NET* component characterizes the network overhead associated with loading models, textures, JavaScript libraries, and multimedia content. The volume of network traffic directly affects the launch speed of the WebAR application and the scene initialization time. This parameter is particularly critical for mobile devices with unstable Internet connections [30].

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

The *MEM* parameter describes RAM usage during application operation. Memory is used to store 3D models, textures, frame buffers, tracking results, and utility data structures. Excessive memory usage can cause performance degradation or force the WebAR scene to terminate on mobile devices.

The *FPS* (Frames Per Second) component characterizes the smoothness of scene rendering and the number of frames the system generates per second. For a comfortable augmented reality experience, it is usually necessary to maintain at least 30 FPS, while 60 FPS is considered optimal. A drop in FPS leads to image stuttering and instability of AR objects.

4.3 Assessing the CPU load of a WebAR application

Although WebAR performance depends on a combination of several parameters, the key factor in most scenarios is the *CPU* load, since the main algorithms for computer vision, camera pose estimation, environment analysis, and tracking are executed by the central processor in real time. Furthermore, unlike native AR applications, WebAR has an additional layer of browser abstraction, which increases the load specifically on the CPU. Additionally, even a slight CPU overload can cause both the accumulation of delays throughout the entire AR pipeline and disrupt the overall synchronization of the scene. Furthermore, on mobile devices, the CPU often becomes the primary source of power consumption and heat generation during AR operation [31]. Under sustained load, the system may automatically reduce the processor frequency (thermal throttling), which directly reduces FPS and tracking stability.

The CPU load of a WebAR application is determined by the volume of computations the system performs during scene analysis and augmented reality rendering. To estimate the load, the number of operations, their execution time, the processor clock frequency, the level of parallelism, and the intensity of data processing are taken into account. To calculate the load in the mathematical model, the following formula is used

$$CPU_{stage} = \frac{N_{ops}}{f_{cpu} \cdot \eta}, \qquad (16)$$

where N_{ops} characterizes the number of arithmetic and logical operations performed at a specific stage of the WebAR system's operation, the parameter f_{cpu} determines the processor clock frequency and affects the speed of computation, while the coefficient η describes the efficiency of CPU resource utilization, taking into account code optimization, parallel execution, and performance losses. As the number of operations or the complexity of computer vision algorithms increases, the processor load rises significantly [32]. Therefore, to ensure the stable operation of WebAR applications, it is important to optimize algorithms, reduce the computational load, and make effective use of multithreaded data processing.

Let's calculate the processor load step by step.

4.3.1 The initialization stage is the first and key step in preparing a 3D application for operation. At this stage, a WebGL context is created, which enables interaction with graphics hardware and sensors. Engine initialization includes loading the necessary libraries and settings, as well as preparing internal data structures. An important component is the creation of the scene graph – a structure that describes the hierarchy of scene objects and their relationships. Additionally, sensors are connected that are responsible for collecting information from the environment or the user.

The CPU load at this stage can be calculated using the following formula:

$$CPU_{init} = \frac{N_{ob} \cdot C_{ob} + N_{comp} \cdot C_{comp}}{f_{cpu} \cdot \eta_{init}}, \qquad (17)$$

where N_{ob} – quantity of stage objects; C_{ob} – specific number of computational units (CU) per object; N_{comp} – number of components; C_{comp} – the specific number of CU for their initialization; coefficient η_{init} takes into account additional overhead costs associated with organizing the process.

This formula allows you to estimate how much CPU resources will be used during system startup. It is important for optimizing startup, especially in cases involving large scenes or complex components. The initialization phase is often critical in terms of startup time, so the efficiency of this step directly affects the overall performance of the application. Proper organization of this phase helps reduce delays.

4.3.2 The resource loading phase includes several important processes, such as HTTP request processing, data decompression, thread management, and caching. These operations are performed to ensure the correct loading and processing of information before it is displayed. HTTP processing involves analyzing incoming requests and responses and generating

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

appropriate responses. Decompression is required to unpack compressed data transmitted over the network to reduce traffic volume. Thread management allows for the efficient allocation of CPU resources among different tasks, ensuring parallelism and speed [33]. Caching helps reduce the number of repeated requests to the same resources, which improves performance.

To quantitatively estimate the CPU load during this stage, the following formula is used:

$$CPU_{load} = \frac{S_{dec} \cdot C_{data} + N_{res} \cdot C_{res}}{f_{cpu} \cdot \eta_{load}}, \quad (18)$$

where S_{dec} – the amount of data to be decompressed; C_{data} – he specific amount of decompression CU per byte; N_{res} – the number of requests; C_{res} – the specific amount of CU per request; η_{load} – CPU load efficiency.

The formula defines the total load as the sum of the resources spent on decompressing and processing each resource, taking into account their quantity and cost. This helps estimate the CPU power that will be required to perform all necessary operations during the loading phase. Balanced CPU utilization at this stage helps avoid delays and ensures fast content loading [34].

4.3.3 The parsing stage is the most CPU-intensive when processing 3D scenes. During this stage, structured data is parsed and interpreted, 3D models are decoded, shader programs are compiled, and the scene is constructed for subsequent rendering. JSON parsing involves converting text data into an internal format that the system can understand. Decoding 3D models is necessary to obtain geometry, textures, and animations. The formation of the scene graph organizes scene objects into a hierarchical structure, which facilitates the management of their positions and properties. These operations require significant computational resources, which is why the CPU load is at its peak at this point. To estimate this load, the following formula is used:

$$CPU_{purse} = \frac{V_{sc} \cdot C_{\text{vertex}} + F_{pol} \cdot C_{pol} + S_{shader} \cdot C_{shader}}{f_{cpu} \cdot \eta_{purse}}, \quad (19)$$

where V_{sc} – the number of vertices in the scene; C_{vertex} – the specific number of CUs required to process a single vertex; F_{pol} – the number of polygons; C_{pol} – the specific number of CUs required for a single polygon; S_{shader} – the number of shader programs; C_{shader} – the specific number of CUs required to compile a single shader; η_{purse} – the processor utilization efficiency at this stage.

Parsing large models with a high number of vertices and polygons, as well as a large number of shader programs, significantly increases the demands on computational resources. This necessitates optimizing data structures and shader code to reduce processing time. Effective resource management at this stage allows you to prepare the scene for smooth and fast rendering. At the same time, it is important to monitor CPU utilization to avoid overloading the system. Proper balancing and optimization of the parsing stage are critical for WebAR performance.

It should be noted that in WebAR (augmented reality in a web browser), the time requirements for data parsing, frame processing, and script execution are extremely strict. Any delay causes visual stuttering, desynchronization of 3D objects from the real world, and can lead to motion sickness in the user. Here are the key timing requirements and constraints that the entire process is broken down into [35, 36]:

1. For AR to appear smooth, the frame rate must be 60 FPS (frames per second), so the total time per frame is 16.67 ms, or 1000 ms / 60 frames.

2. Any model parsing that takes longer than 2–3 ms will cause noticeable jank.

3. Motion-to-Photon Latency – the time from when the user physically rotates the phone to when the virtual object on the screen shifts to match the new position – should not exceed 20–30 ms. If computer vision algorithms (parsing of facial points, planes, or markers) run slower than 15 ms per frame, the virtual object will "lag" behind the real world during rapid movements.

4.3.4 The rendering stage is a key stage in the graphics process, where the CPU prepares all the information for transfer to the GPU. Several important operations are performed at this stage, such as preparing draw calls, updating transformation matrices, performing culling (removing unnecessary objects from the scene), and transferring commands directly to the GPU. Draw calls are commands that tell the GPU which objects to render and in what order. Updating matrices involves changing the transformations of objects or the camera, which determines how the scene will be rendered in the current frame. Culling saves resources by excluding objects that are not visible to the camera through visibility or collision checks. This reduces the load on the GPU by decreasing the amount of graphical

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

data being processed. The CPU load at this stage is determined by the following formula:

$$CPU_{render} = \frac{N_{draw} \cdot C_{draw} + N_{\text{Renewal}} \cdot C_{\text{Renewal}} + N_{cont} \cdot C_{cont}}{f_{cpu} \cdot \eta_{render}}, \quad (20)$$

where N_{draw} – number of draw calls; C_{draw} – cost of preparing one draw call; N_{Renewal} – number of matrix updates; C_{Renewal} – cost of renewal one matrix; N_{cont} – number of collision or visibility checks; C_{cont} – cost per check; η_{render} – processor efficiency.

The number of draw calls significantly impacts performance, as each call takes time to process. Matrix updates are essential for animation and the movement of objects in the scene. Visibility checks help avoid unnecessary rendering of invisible elements, which significantly improves efficiency [8]. This entire process must be optimized to achieve a high frame rate and smooth application performance.

4.4 CPU Load Balancing

Let the execution time for each of the considered stages of loading a WebAR application scene on a mobile device be T_{init}, T_{load}, T_{parse}, T_{render} accordingly, and after performing the calculations using equations (17) through (20), the following values for the required CPU time were obtained: CPU_{init}, CPU_{load}, CPU_{parse}, CPU_{render}. For each stage $(i = 1\ldots4)$, we will calculate the load index using the formula:

$$Ind_i = \frac{CPU_i}{T_i} \cdot 100\%, \quad i = \overline{1,4}, \quad (21)$$

where i corresponds to the sequential number of the WebAR scene loading stage in the application.

We will define the current CPU load index as a vector $\mathbf{Ind} = (Ind_1, Ind_2, Ind_3, Ind_4)$

Let's define the CPU load balance metric as

$$\gamma(\mathbf{Ind}) = \frac{\sum_{i=1}^{4} \left| Ind_i - \sum_{i=1}^{4} Ind_i / 4 \right|}{\sum_{i=1}^{4} Ind_i}. \quad (22)$$

After decomposing the stages, it becomes possible to relieve the overloaded stages by performing certain actions on the less-loaded preceding stages. As a result of this procedure θ_ξ, we obtain a new load index

$$\theta_\xi(\mathbf{Ind}) = (\theta_\xi(Ind_1), \theta_\xi(Ind_2), \theta_\xi(Ind_3), \theta_\xi(Ind_4)). \quad (23)$$

In that case, the problem of finding the optimal CPU load balance can be formulated as follows:

$$\theta_\xi^*(\mathbf{Ind}) = \arg\left(\gamma(\theta_\xi(\mathbf{Ind})) \middle| \gamma(\theta_\xi(\mathbf{Ind})) > \gamma(\theta_\xi^*(\mathbf{Ind})) \quad \forall \theta_\xi \in \Theta, \theta_\xi \neq \theta_\xi^* \right), \quad (24)$$

where $\Theta = \{\theta_\xi\}$ – the set of all possible stage-override procedures that satisfy all the timing requirements and constraints 1–3 for the parsing stage.

5 Study of the scene loading process for a WebAR app on a Google Pixel 8 smartphone

5.1 Experimental conditions

This study evaluates the performance of a WebAR application developed using *Angular 19* and the *A-Frame 1.7.1* library. A Location AR scene containing a single AR object with a size of 16.4 MB was selected for testing. Since the maximum load on the application occurs during initialization and scene loading, the study focuses exclusively on this stage. CPU load in the context of the web page was selected as the primary metric for evaluating performance.

Performance metrics were collected using the *Chrome DevTools* developer tools. The test mobile device was connected to the computer via a USB interface. The Performance tab was used to measure CPU parameters. It details the activity of the WebAR application, showing the time spent on Scripting, Rendering, and other categories. It is important to note that this tool does not reflect the overall CPU load of the entire device; therefore, the data obtained should be interpreted exclusively as indicators of the web page's load. Since the process of using *Chrome DevTools* itself creates an additional load, each test scenario was run 10 times. For further analysis, the run with the average values was selected.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

Testing was conducted on a Google Pixel 8 smartphone (October 2023 model). The test device is equipped with a Google Tensor G3 processor, has 8 GB of RAM, and runs on the Android 16 operating system.

5.2 Experiment Description

To ensure the reproducibility of results and minimize the impact of uncontrolled background processes, a device preparation procedure was implemented. Before testing began, the mobile device's battery was charged to a level above 90%. After that, a full system reboot was performed to clear the RAM and force-close background processes. The display brightness was set to 80%. All third-party apps were disabled; only the website with the WebAR app remained active in the web browser.

To ensure consistent lighting conditions and stabilize the camera angle, the device was securely mounted in a static position using a tripod. At the software level, developer mode and USB debugging were enabled (Fig. 1). The physical connection to the host computer was established using high-speed cables. The connection was verified using the *Chrome DevTools*, via the *chrome://inspect* page, which confirmed the successful detection of the remote device and the availability of the target browser tab for inspection (Fig. 2).

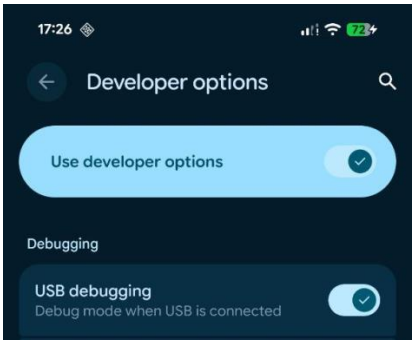

Fig. 1. Developer mode and USB debugging enabled

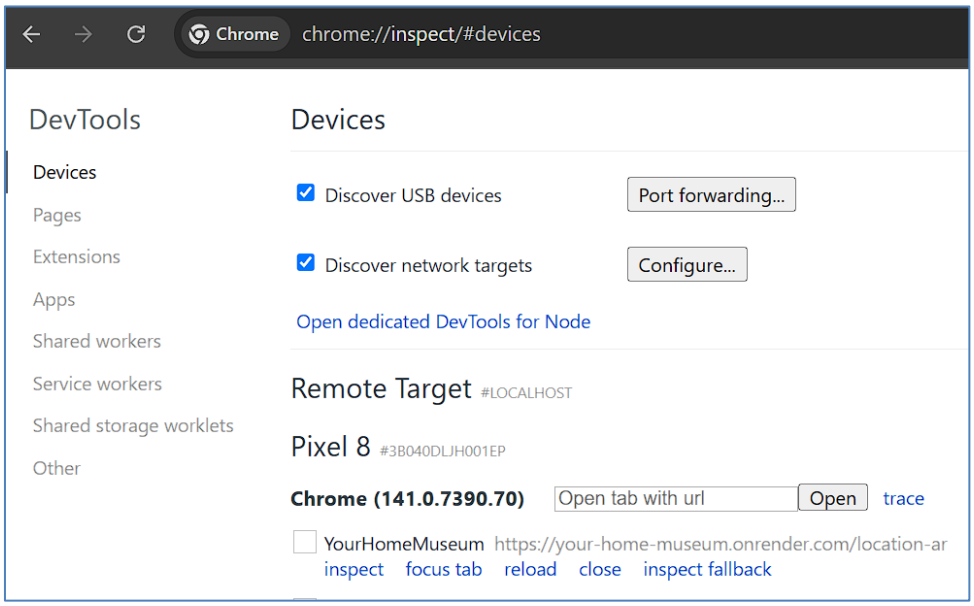

Fig. 2. Displaying the mobile tab on a computer

After completing the preparatory phase, the data collection procedure was implemented. It involved performing ten consecutive iterations of performance profiling using the Performance tab in *Chrome DevTools*. The experimental scenario covered the full cycle of loading and initializing a geolocated WebAR scene.

Upon completion of each measurement iteration, trace files containing detailed performance metrics were exported. Specifically, the time spent on Scripting, System processes, Rendering, and Painting was analyzed. Additionally, the total script execution time (Total), average frame rate (FPS), and JavaScript heap growth were recorded.

To select the most representative sample for further detailed analysis, a comparison was made of the peak values of JS Heap memory usage and the duration of key processing stages. It is important to note that the Total metric also accounts for periods of system idle time. The resulting quantitative data has been organized and presented in Table 1.

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

Table 1. *Test results on the Pixel 8 device*

Test	JS Heap (пік), mb	Scripting, ms	System, ms	Rendering, ms	Painting, ms	Total, ms
1	60	2,209	558	186	145	6,743
2	60,3	2,289	573	166	127	6,401
3	41,1	1,649	438	232	153	6,181
4	42,8	2,359	543	260	161	7,238
5	45	1,705	462	214	158	5,929
6	26	2,352	580	183	140	5,898
7	42,1	2,502	568	196	156	6,262
8	42,6	2,534	845	203	180	6,898
9	42,7	2,454	574	140	135	5,904
10	60,1	2,820	668	209	165	6,765

5.3 Choosing a representative sample

To identify a representative test among the ten results obtained, the *normalized deviation method* was applied. This approach makes it possible to objectively identify the test whose scores are closest to the average performance conditions, minimizing the impact of random fluctuations and outliers.

For each test, the normalized deviation was calculated using the corresponding formula:

$$Score_i = \sum_j \frac{|X_{ij} - Me_j|}{IQR_j}. \quad (25)$$

where X_{ij} – the value of the j-th metric for the i-th test; Me_j – the median of the corresponding metric; IQR_j – the interquartile range of this metric.

The interquartile range is calculated as the difference between the third and first quartiles of the sample and is determined using the formula:

$$IQR = Q_3 - Q_1. \quad (26)$$

where Q_1 – 25th percentile (lower quartile); Q_3 – 75th percentile (upper quartile).

For a sample of size $n = 10$, the calculation was performed using Tukey's method: the sorted data set was divided into two equal parts of 5 elements each. In this case, Q_1 is defined as the median of the first half of the set, the 3rd element in the sorted list, and Q_3 as the median of the second half, the 8th element.

Based on Tukey's method, basic statistical indicators were determined for each metric. The results of determining the median, Q_1, Q_3 and calculating the interquartile range *IQR* using equation (26) for each column are presented in Table 2.

The results of the calculation of the standardized deviation for each test using Equation (1) are presented in Table 3.

Table 2. *Values of the indicators based on the Tukey method*

Metric	Median	Q_1 (3-rd element)	Q_3 (8-th element)	*IQR*
JS Heap	42,75	42,1	60,0	17,9
Scripting	2,356	2,209	2,502	0,293
System	570,5	543	580	37,0
Rendering	199,5	183	214	31,0
Painting	154,5	140	161	21,0
Total	6,332	5,929	6,765	0,836

Table 3. *Normalized test deviations on Pixel 8 device*

Test	Score
1	3,18
2	3,75
3	7,39
4	4,10
5	6,40
6	2,95
7	0,87
8	10,04
9	3,79
10	6,51

The results of the calculations show that Test No. 7 has the lowest normalized value, indicating the smallest deviation of its parameters from the median characteristics of the sample. This indicates the greatest correspondence of the results obtained in this test to the typical conditions of the experiment. Thus, the specified recording was identified as representative and used for further comparative analysis between the devices under study.

5.4 Analysis of the representative recording

The representative recording obtained in the *Performance* tab of the *Chrome DevTools* toolkit is shown in Fig. 3. The analysis focuses on the application initialization period; therefore, the intervals at the beginning and end of the recording are not included. The main visual metrics are: the CPU activity graph at the top of the figure; the Main thread, which records function calls; the JS Heap memory usage graph; and the Summary section.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

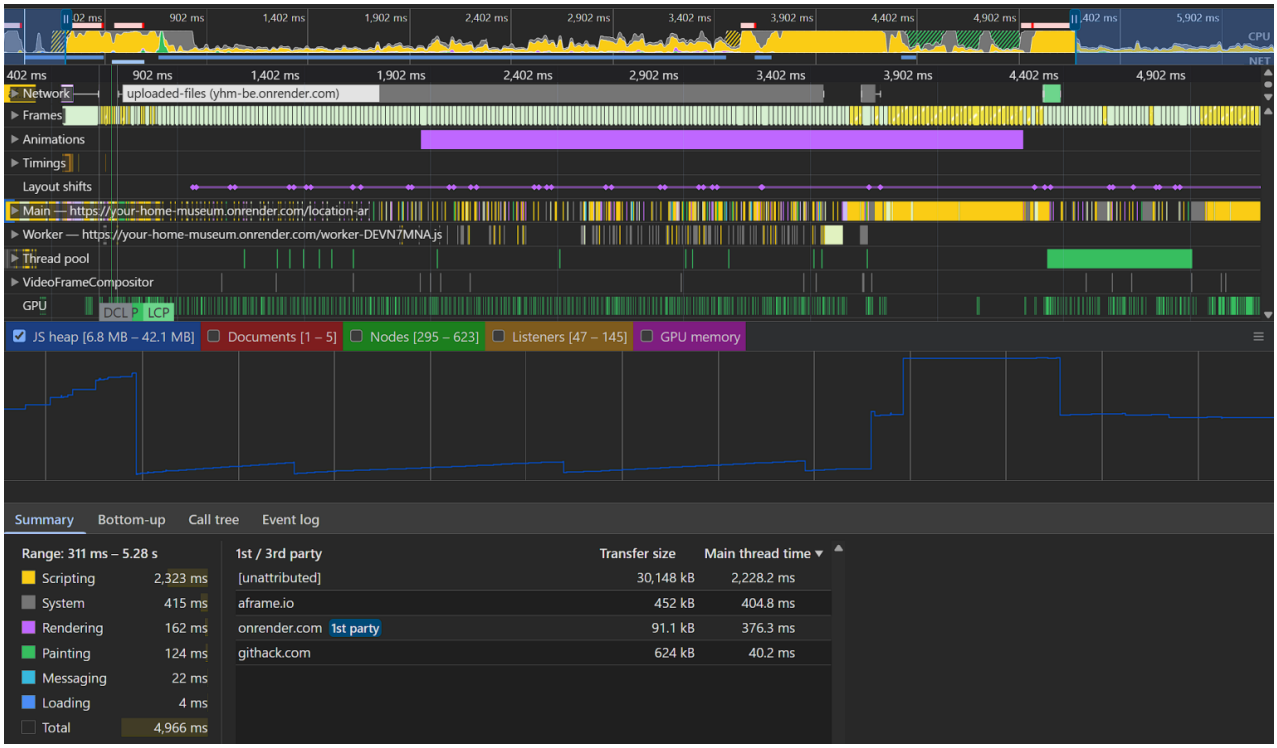

Fig. 3. Representative recording

The Summary section indicates that the total runtime of the main thread is 4.96 seconds, of which: 2.32 seconds are spent on scripts, 1.91 seconds on idle time, 0.41 seconds on system overhead, 0.16 seconds on page rendering, and 0.12 seconds on content rendering. This data is presented in a chart in Fig. 4. Inter-process communication (Messaging) and page loading are not included, as they account for less than 1% of the total time. Thus, the Scripting process accounts for the largest share (47%), followed closely by Idle (39%).

The CPU graph is divided into four intervals (Fig. 5), three of which show high CPU load, as indicated by the sections filled entirely in yellow. The unhighlighted intervals are not included because they are not directly related to the application's initialization.

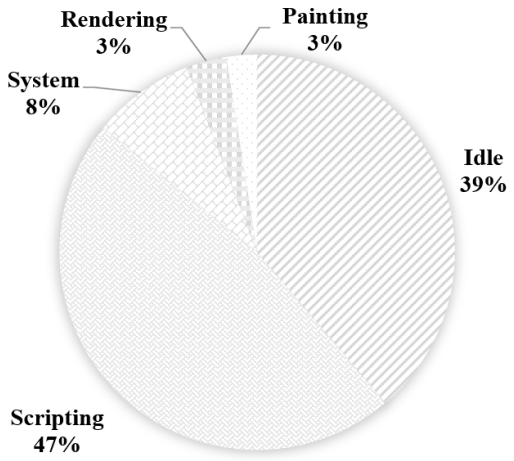

Fig. 4. Flowchart of the application loading process

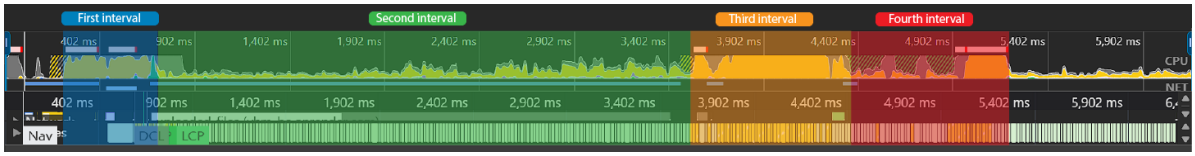

Fig. 5. Division of the recording into intervals

The first interval of the profiling covers the time range from 0.311 to 0.795 seconds. This stage is characterized by a high computational load due to the initialization of the client application. The Scripting process accounts for the dominant share of time, over 50% of the entire interval. This computational activity is primarily driven by two key operations. The Unattributed category had the longest duration among them, at

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

108 milliseconds. A detailed trace analysis revealed that this category involves the execution of native code by the JavaScript engine. Due to the specific nature of code optimization in the Angular framework, the profiler was unable to correlate these operations with specific function calls, which is typical for such architectures. The second significant component of script execution was the initialization of the A-Frame library, which took 89.1 milliseconds. An additional indicator of the active initialization phase is the rapid increase in JS Heap memory consumption from 26 to 36 megabytes. This dynamic indicates the mass creation of virtual DOM objects and the preparation of a three-dimensional scene. Overall, during this interval, the system accumulates resources for processing logic and data structures, and at the end of the stage, it performs the initial page rendering process, which takes only 28 milliseconds.

Upon completion of the initialization phase, a period of *moderate computational intensity* (0.796–3.62 s) is observed. During this period, the dominant state of the main thread is the waiting mode – 1.6 s. The application requests and waits for the transfer of a 3D model from the server, which keeps the CPU load low. Computational activity does not include blocking tasks and is primarily driven by operations such as reading the input stream, data transformation, and background processes. An important characteristic of this stage is memory optimization: instead of data accumulation, the Garbage Collector was triggered, reducing the size of the JS Heap from 37 MB to 7 MB. This indicates effective resource release after initialization and before loading the 3D model.

The third stage (3.62 s – 4.45 s) is the heavy data processing phase. After the AR object finished loading from the server, the main thread entered a state of peak load – scripting took up over 82% of the time. The longest blocking task, lasting 550 ms, was recorded during this interval. Analysis shows that this is the process of parsing the 3D model's binary data and converting it into Three.js scene structures. This is confirmed by a sharp increase in memory usage: from 8.5 MB at the start of the interval to 42.1 MB. This spike corresponds to the decompression of the compressed model file into full-fledged objects in RAM.

The final phase (4.45 s – 5.3 s) is the 3D model rendering period. In the first half of this time interval, the browser utilizes a pool of background threads – Thread Pool Workers – which enables the parallel execution of decoding operations and the preparation of graphical resources. In the recorded trace, three workers are observed, each processing separate fragments of the model or associated texture data; this activity can be classified as the preparatory phase of rendering. While the main thread's activity is minimal in the first half of the interval, the CPU load becomes dominant in the second half. It contains calls to WebGLRenderer.render, renderScene, renderObjects, renderBufferDirect, as well as WebGLUniforms.upload and safeSetTexture2D operations. This indicates the compilation of shader programs, the loading of textures, and the direct rendering of the 3D model in the graphics scene.

The results of the analysis of a representative recording indicate four distinct stages of application loading: an intensive initialization phase of the initial empty scene lasting from 0.311 s to 0.795 s, which took 0.48 s; a moderate period of loading the 3D object from the server-side from 0.796 s to 3.62 s, which took 2.82 s; a very resource-intensive period of 3D object parsing from 3.62 s to 4.45 s, which lasted 0.83 s; and an intensive phase of 3D object rendering from 4.45 s to 5.3 s, which lasted 0.85 s.

5.5 Results

To determine the most resource-intensive stage of the WebAR application's loading process, four intervals were considered: initialization, resource loading, parsing, and rendering. For each interval, the loading index was calculated using formula (21). The results of the calculations are presented in Table 4.

Table 4. *Processor performance metrics at key stages of 3D application development*

№	Stage	Interval, s	T_i, ms	CPU_i, ms	Dwell time, ms	Ind_i, %
1	Initialization	0.311 – 0.795	484	311.9	172.1	64.4
2	Loading	0.796 – 3.62	2824	1318.7	1505.3	46.7
3	Parsing	3.62 – 4.45	830	732.4	97.6	88.2
4	Rendering	4.45 – 5.30	850	540.9	309.1	63.6

The most critical stage turned out to be the third *parsing* interval, during which the peak CPU load of 88.2% was recorded (Fig. 6). Despite its relatively short duration (0.83 s), the density of computational operations during this

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

interval is the highest. This is due to the synchronous nature of the 3D model binary data deserialization process and the massive creation of objects in memory, which leaves the processor almost no idle time.

The *initialization* (64.4%) and *rendering* (63.6%) stages show similar average load metrics but differ in the nature of the operations. While in the first stage the CPU is primarily occupied with JIT compilation and executing JavaScript library code, in the fourth stage interaction with the GPU and frame preparation play a significant role.

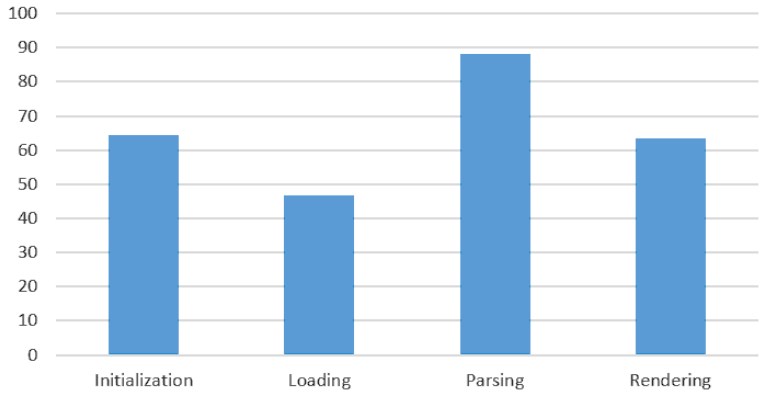

Fig. 6. Percentage load graph of intervals

The least loaded stage is *resource loading* (46.7%), which is an expected result, since network bandwidth is the limiting factor in this interval, and the main processor thread spends more than half the time waiting for data.

Thus, the highest computational load is observed during the third *Parsing* interval, while the *Loading* stage is the least loaded and has the highest idle time.

Further performance optimization should be based on resource reallocation: reducing idle time in the *Loading* stage by transferring some computational tasks from the *Parsing* stage to it. This approach will ensure the maximum increase in application performance.

6 Practical Significance

The practical significance of the research results lies in the possibility of their direct implementation by companies specializing in the development of commercial augmented reality applications. The modern e-commerce, interactive marketing, and digital education industries are actively adopting augmented reality technologies to improve customer engagement. However, the success of such solutions critically depends on the smooth operation of the software on consumers' end devices. The approach to deep profiling of browser engine microprocesses proposed in this work provides the industry with an effective tool for identifying hidden technical flaws that cannot be detected using standard network analytics tools. The refined mathematical model of the process and the empirical data obtained form a solid scientific foundation for the transition from intuitive development to engineering-based design of client application architecture.

The key issue hindering the widespread and effective implementation of browser-based augmented reality in a business environment is the loss of control over the interface during the loading of volumetric spatial content. When a user initiates the viewing of a three-dimensional product model, there is intensive consumption of the mobile device's computational resources. The peak load on the central processing unit during geometry deserialization, as identified in the study, causes a prolonged blockage of the main execution thread. For the end user, this appears as a complete system freeze: the screen does not respond to touch, animations stop, and overall performance degrades sharply. Such a negative technical experience destroys the sense of immersion and leads to disappointment with the quality of the digital service provided.

The economic efficiency of any digital product is directly correlated with user experience metrics. In the highly competitive e-commerce market, companies think in terms of financial return and customer satisfaction levels. Technical glitches caused by inefficient allocation of processor resources have direct financial consequences. Users tend to immediately leave a webpage if its interface becomes unresponsive, even for a fraction of a second. Accordingly, eliminating architectural overloads – the nature of which was demonstrated in this study – will allow businesses to

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

significantly reduce user drop-off rates. Maintaining interface responsiveness ensures that a potential buyer will complete the intended action, whether it is viewing a product from all angles or placing an order. Consequently, optimizing the 3D scene initialization process becomes not merely a technical task but a strategic tool for increasing conversion rates in commercial augmented reality services.

To achieve these economic and technical benefits, developers of commercial platforms must implement new engineering standards for working with spatial data. The study's findings indicate the need to move away from monolithic synchronous loading of 3D models in the browser's main thread. Instead, the industry is advised to transition to methods of incremental geometry loading and offloading resource-intensive tasks to background computing processes. The use of modern 3D data compression formats combined with asynchronous logic processing will help balance the load on smartphone hardware. Implementing such architectural solutions will ensure a stable frame rate and high system responsiveness regardless of the cost and technical specifications of the consumer's mobile device.

Thus, this work goes beyond purely theoretical analysis and offers the industry a ready-made analytical framework. The application of the outlined principles for evaluating and optimizing performance opens the way for companies to create high-quality, stable, and cost-effective products. Meeting technical performance requirements is inextricably linked to meeting consumer needs, making the results of this study an integral part of the process of improving commercial augmented reality ecosystems.

7 Conclusions

As a result of this study, an approach was implemented for the comprehensive evaluation of performance and the identification of patterns in the distribution of computational load during the initialization of WebAR applications. To this end, an improved mathematical model of the WebAR application scene loading process on a mobile device was applied. Through the use of deep browser profiling tools, the microprocesses of the browser engine were decomposed, and the nonlinear nature of mobile device hardware resource consumption was empirically confirmed.

Analysis of isolated time profiles of a client application based on the Angular 19 component architecture and the A-Frame 1.7.1 declarative framework allowed us to identify the key phases of scene initialization. It has been scientifically proven that the critical factor in performance degradation is not the stage of direct graphics rendering, which demonstrates a stable load of 63.6 percent, but the process of parsing spatial geometry. It is during the parsing stage that the peak CPU load of 88.2 percent was recorded, along with the longest blocking task lasting 550 milliseconds, caused by the synchronous deserialization of the model's binary data. At the same time, the effectiveness of memory management in the studied stack was confirmed: the garbage collection mechanism was observed to function correctly, ensuring a rapid reduction of the peak JS Heap size from 40 megabytes to a stable operating level of 22 megabytes.

The obtained objective micro-level data form a reliable foundation for developing well-founded architectural strategies for optimizing the performance of three-dimensional web applications. Since processing a monolithic model in the main thread causes critical blocking of the main execution thread in a single-threaded environment, traditional approaches to initialization require a fundamental reevaluation. A promising and practical approach to eliminating the identified bottlenecks is the use of asynchronous and incremental geometry loading methods, as well as the mandatory delegation of resource-intensive deserialization processes to background computational threads. Implementing such engineering solutions will allow developers of commercial WebAR services to avoid critical interface delays, ensuring high user retention and improving the overall economic efficiency of digital products.

Conflict of interest

The authors declare that they have no conflict of interest with respect to this research, including financial, personal, authorship, or other conflicts that could influence the research and its results presented in this article.

Funding

The study was conducted without financial support.

Data availability

The manuscript has no associated data.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technology in the creation of this work.

Acknowledgments

The study is being conducted within the framework of the research project 0126U003139 of the National Fund of Ukraine on the topic "Scientific Foundations for the Formation and Management of Human Capital in a Multi-Project Environment to Support the Sustainable Development of Ukraine's Recovery Programs".

References

1. Kayser, I. (2026), "Technology, Digital Transformation and Society: A Closing Editorial", *Social Sciences*, Vol. 15(4), 251 p. DOI: https://doi.org/10.3390/socsci15040251
2. Kalyoncu, F., Karal, H. (2025), "Design and development of augmented reality application for basic concepts of computer systems", *Education and Information Technologies*, Vol. 30(1), pp. 347-375. DOI: https://doi.org/10.1007/s10639-024-12971-x
3. Nadeem, M., Lal, M., Cen, J., Sharsheer, M. (2022), "AR4FSM: Mobile Augmented Reality Application in Engineering Education for Finite-State Machine Understanding", *Education Sciences*, Vol. 12(8), 555 p. DOI: https://doi.org/10.3390/educsci12080555
4. Butt, A., Ahmad, H., Muzaffar, A., Ali, F., Shafique, N. (2022), "WOW, the make-up AR app is impressive: a comparative study between China and South Korea", *Journal of Services Marketing*, Vol. 36, No. 1 pp. 73–88. DOI: https://doi.org/10.1108/JSM-12-2020-0508
5. Yin, C. Z. Y., Jung, T., tom Dieck, M. C., Lee, M. Y. (2021), "Mobile Augmented Reality Heritage Applications: Meeting the Needs of Heritage Tourists", *Sustainability*, Vol. 13(5), 2523 p. DOI: https://doi.org/10.3390/su13052523
6. Parekh, P., Patel, S., Patel, N., Shah M. (2020), "Systematic review and meta-analysis of augmented reality in medicine, retail, and games", *Visual Computing for Industry, Biomedicine, and Art*, Vol. 3, No. 1, 21 p. DOI: https://doi.org/10.1186/s42492-020-00057-7
7. Udayan, J. D., Kataria, G., Yadav, R., Kothari, S. (2020), "Augmented Reality in Brand Building and Marketing – Valves Industry", *2020 International Conference on Emerging Trends in Information Technology and Engineering* (ic-ETITE), pp. 1–6. DOI: https://doi.org/10.1109/ic-ETITE47903.2020.425
8. Boutsi, A.M., Ioannidis, C., Verykokou, S. (2023), "Multi-Resolution 3D Rendering for High-Performance Web AR", *Sensors*, Vol. 23 (15), 6885 p. DOI: https://doi.org/10.3390/s23156885
9. Alsulami, M. H., Khayyat, M. M., Aboulola, O. I., Alsaqer, M. S. (2021), "Development of an Approach to Evaluate Website Effectiveness", *Sustainability*, Vol. 13(23), 13304 p. DOI: https://doi.org/10.3390/su132313304
10. Morales-Vargas, A., Pedraza-Jimenez, R., Codina, L. (2022), "Website quality in digital media: literature review on general evaluation methods and indicators and reliability attributes", *Revista Latina de Comunicacion Social*, Vol. 80, pp. 39–63. DOI: https://doi.org/10.4185/RLCS-2022-1515
11. Kuchuk, G., Kharchenko, V., Kovalenko, A., Ruchkov, E. (2016), "Approaches to selection of combinatorial algorithm for optimization in network traffic control of safety-critical systems", *Proceedings of 2016 IEEE East-West Design and Test Symposium*, EWDTS 2016, 7807655 p. DOI: https://doi.org/10.1109/EWDTS.2016.7807655
12. Matvieiev, M. (2024), "Analysis of Optimization Methods for Augmented Reality Web Applications", *Control, Navigation and Communication Systems*, No. 4(78), pp. 106-108. DOI: https://doi.org/10.26906/SUNZ.2024.4.106
13. Ghattas, M. M., Sartawi, P. D. B. (2020), "Performance Evaluation of Websites Using Machine Learning", *EIMJ*, Vol. 51, pp. 36–41, available at: https://www.researchgate.net/publication/345975296_Performance_Evaluation_of_Websites_Using_Machine_Learning
14. Tuah, N.M., Ahmad, W.N.W., Andrias, R.M., Dg. S. Ajor, Sura, S., Rodzuan, A.R.A. (2025), "Assessing the user experience of marker-based 3D WebAR applications using user experience questionnaire", *International Journal of Informatics and Communication Technology*, Vol. 14(1), pp. 31–41. DOI: https://doi.org/10.11591/ijict.v14i1.pp31-41
15. Ghattas, M., Mora, A. M., Odeh, S. (2025), "A Novel Approach for Evaluating Web Page Performance Based on Machine Learning Algorithms and Optimization Algorithms", *AI*, Vol. 6., No. 2., 19 p. DOI: https://doi.org/10.3390/ai6020019
16. Hussein, H. A., Ali, M. H., Al-Hashimi, M., Majeed, N. T., Hameed, Q. A., Ismael, R. D. (2023), "The Effect of Web Augmented Reality on Primary Pupils' Achievement in English", *Applied System Innovation*, Vol. 6(1), 18 p. DOI: https://doi.org/10.3390/asi6010018
17. Lee, D., Shim, W., Lee, M., Lee, S., Jung, K.-D., Kwon, S. (2021), "Performance Evaluation of Ground AR Anchor with WebXR Device API", *Applied Sciences*, Vol. 11(17), 7877 p. DOI: https://doi.org/10.3390/app11177877

18. Angeleri, C., Marini, F., Alvarez, D., Leale, G., Curras, D. (2021), "CPU and RAM Performance Assessment for Different Marker Types in Augmented Reality Applications", *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pp. 271–277. DOI: https://doi.org/10.5220/0010302302710277
19. Roy, S. G., Kanjilal U. (2021), "Web-based Augmented Reality for Information Delivery Services: A Performance Study", *DESIDOC Journal of Library & Information Technology*, Vol. 41, No. 3, pp. 167–174. DOI: https://doi.org/10.14429/djlit.41.03.16428
20. Ghattas, M., Odeh, S., Mora, A. M. (2025), "Predicting Website Performance: A Systematic Review of Metrics, Methods, and Research Gaps (2010–2024)", *Computers*, Vol. 14. No. 10, 446 p. DOI: https://doi.org/10.3390/computers14100446
21. Najadat, H., Al-Badarneh, A., Alodibat, S. (2021), "A review of website evaluation using web diagnostic tools and data envelopment analysis", *Bulletin of Electrical Eng. and Informatics*, Vol. 10, pp. 258–265. DOI: https://doi.org/10.11591/eei.v10i1.1755
22. Niazi, M. G., Kamran, M. K. A., Ghaebi, A. (2020), "Presenting a proposed framework for evaluating university websites", *Electronic Library*, Vol. 38, No. 5–6, pp. 881–904. DOI: https://doi.org/10.1108/EL-06-2020-0141
23. Romanenkov, Yu., Mukhin, V., Kosenko, V., Revenko, D., Lobach, O., Kosenko, N., Yakovleva, A. (2024), "Criterion for Ranking Interval Alternatives in a Decision-Making Task", *International Journal of Modern Education and Computer Science*, Vol. 16(2), pp. 72–82. DOI: https://doi.org/10.5815/ijmecs.2024.02.06
24. Kovalenko, A., Kuchuk, H., Kuchuk, N., Kostolny, J. (2021), "Horizontal scaling method for a hyperconverged network", *2021 International Conference on Information and Digital Technologies* (IDT), Zilina, Slovakia, DOI: https://doi.org/10.1109/IDT52577.2021.9497534
25. Kuchuk, H., Matvieiev, M. (2025), "Modeling the process of loading 3D models in a client application", *Advanced Information Systems*, Vol. 9(4), pp. 11–16. DOI: https://doi.org/10.20998/2522-9052.2025.4.02
26. Kumar, A., Arora, A. (2019), "A Filter-Wrapper based Feature Selection for Optimized Website Quality Prediction", *Proceedings 2019 Amity International Conference on Artificial Intelligence* (AICAI), pp. 284–291. DOI: https://doi.org/10.1109/aicai.2019.8701362
27. Allison, R., Hayes, C., McNulty, C. A. M., Young, V. (2019), "A Comprehensive Framework to Evaluate Websites: Literature Review and Development of GoodWeb", *JMIR Formative Research*, Vol. 3(4). DOI: https://doi.org/10.2196/14372
28. Semenov, S., Mozhaiev, O., Kuchuk, N., Mozhaiev, M., Tiulieniev, S., Gnusov, Yu., Yevstrat, D.,Chyrva, Y., Kuchuk, H. (2022), "Devising a procedure for defining the general criteria of abnormal behavior of a computer system based on the improved criterion of uniformity of input data samples", *Eastern-European Journal of Enterprise Technologies*, Vol. 6(4–120), pp. 40–49. DOI: https://doi.org/10.15587/1729-4061.2022.269128
29. Kuchuk, H., Kovalenko, A., Ibrahim, B.F. Ruban, I. (2019), "Adaptive compression method for video information", *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8(1), pp. 66–69. DOI: http://dx.doi.org/10.30534/ijatcse/2019/1181.22019
30. Kuchuk, H., Kalinin, Y., Dotsenko, N., Chumachenko, I., Pakhomov, Y. (2024), "A Decomposition of integrated high-density IoT data flow"A, *Advanced Information Systems*, Vol. 8, No. 3, pp. 77–84. DOI: https://doi.org/10.20998/2522-9052.2024.3.09
31. Kuchuk, H., Mozhaiev, O., Tiulieniev, S., Mozhaiev, M., Kuchuk, N., Lubentsov, A., Onishchenko, Yu., Gnusov, Yu., Brendel, O., Roh, V. (2025), "Devising a method for energy-efficient control over a data transmission process across the mobile high-density Internet of Things", *Eastern European Journal of Enterprise Technologies*, Vol. 4(4(136)), pp. 46–57, DOI: https://doi.org/10.15587/1729-4061.2025.336111
32. Kuchuk, N., Kashkevich, S., Radchenko, V., Andrusenko, Y. Kuchuk, H. (2024), "Applying edge computing in the execution IoT operative transactions", *Advanced Information Systems*, Vol. 8, No. 4, pp. 49–59. DOI: https://doi.org/10.20998/2522-9052.2024.4.07
33. Kuchuk, H., Kosenko, N., Kuchuk, N., Gopejenko, V., Kosenko, V. (2026), "Adaptive Resource Allocation Method for the Mobile Fog Layer of High-Density Industrial Internet of Things in Industry 5.0 Networks", *Innovative technologies and scientific solutions for industries*, Vol. (1(35), pp. 65–78. DOI: https://doi.org/10.30837/2522-9818.2026.1.065
34. Kuchuk, H., Husieva, Y., Novoselov, S., Lysytsia, D., Krykhovetskyi, H. (2025), "Load Balancing of the layers Iot Fog-Cloud support network", *Advanced Information Systems*, Vol. 9, No. 1, pp. 91–98. DOI: https://doi.org/10.20998/2522-9052.2025.1.11
35. Muff, F., Borcard, D., Fill, HG. (2026), "MM-AR: a web-based open-source metamodeling platform for spatial conceptual modelling", Software and Systems Modeling. DOI: https://doi.org/10.1007/s10270-025-01351-9
36. Boutsi, A.-M., Ioannidis, C., Verykokou, S. (2023), "Multi-Resolution 3D Rendering for High-Performance Web AR", *Sensors*, Vol. 23(15), 6885 p. DOI: https://doi.org/10.3390/s23156885

Received (Надійшла) 24.02.2026
Accepted for publication (Прийнята до друку) 21.05.2026
Publication date (Дата публікації) 27.06.2026

ISSN 2522-9818 (print) Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2026. No. 2 (36)

Відомості про авторів / About the Authors

Кучук Георгій Анатолійович – доктор технічних наук, професор, Національний технічний університет "Харківський політехнічний інститут", професор кафедри комп'ютерної інженерії та програмування, Харків, Україна;
Heorhii Kuchuk – Doctor of Technical Sciences, Professor, National Technical University "Kharkiv Polytechnic Institute", Professor of the Computer Engineering and Programming Department, Kharkiv, Ukraine;
e-mail: kuchuk56@ukr.net
ORCID ID: http://orcid.org/0000-0002-2862-438X
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57057781300

Матвєєв Микита Іванович – Національний технічний університет "Харківський політехнічний інститут", здобувач вищої освіти ступеня доктора філософії кафедри комп'ютерної інженерії та програмування, Харків, Україна;
Mykyta Matvieiev – National Technical University "Kharkiv Polytechnic Institute", Postgraduate Student of the Computer Engineering and Programming Department, Kharkiv, Ukraine;
e-mail: Mykyta.Matvieiev@cit.khpi.edu.ua
ORCID ID: https://orcid.org/0009-0000-8773-6640
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=60141774400

Косенко Наталія Вікторівна – кандидат технічних наук, доцент, Харківський національний університет міського господарства ім. О.М. Бекетова, доцент кафедри управління проектами у міському господарстві і будівництві, Харків, Україна;
Nataliia Kosenko – Candidate of Technical Sciences, Associate Professor, Beketov National University of Urban Economy in Kharkiv, Associate Professor of the Project Management in Urban Economy and Construction Department, Kharkiv, Ukraine;
e-mail: kosnatalja@gmail.com;
ORCID ID: https://orcid.org/0000-0002-5942-3150;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57196219605

Лисиця Дмитро Олександрович – кандидат технічних наук, доцент, Національний технічний університет "Харківський політехнічний інститут", доцент кафедри комп'ютерної інженерії та програмування, Харків, Україна;
Dmytro Lysytsia – Candidate of Technical Sciences, Associate Professor, National Technical University "Kharkiv Polytechnic Institute", Associate Professor of the Computer Engineering and Programming Department, Kharkiv, Ukraine;
e-mail: Dmytro.Lysytsia@khpi.edu.ua
ORCID ID: https://orcid.org/0000-0003-1778-4676
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57220049627

Левченко Андрій Олександрович – кандидат технічних наук, доцент, Військова академія, професор кафедри застосування підрозділів військової розвідки та Сил спеціальних операцій, Одеса, Україна;
Andrii Levchenko – Candidate of Technical Sciences, Associate Professor, Odessa Military Academy, Professor of the Training of Intelligence Units and Special Operations Forces Department, Odesa, Ukraine;
e-mail: katyaandreylev@gmail.com
ORCID ID: https://orcid.org/0000-0001-5550-0027
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57220805603

ОЦІНЮВАННЯ ПРОДУКТИВНОСТІ ЗАВАНТАЖЕННЯ СЦЕНИ WEBAR ЗАСТОСУНКУ НА МОБІЛЬНОМУ ПРИСТРОЇ

Предметом дослідження є закономірності розподілу обчислювальних ресурсів центрального процесора та зміни часових метрик браузерного рушія під час виконання етапів ініціалізації WebAR-сцени. **Об'єктом даного дослідження** є процес ініціалізації та завантаження 3D-контенту у клієнтському WebAR-застосунку на мобільному пристрої. **Метою роботи** є розробка підходу до комплексного оцінювання продуктивності та встановлення закономірностей розподілу обчислювального навантаження під час ініціалізації WebAR-застосунків з використанням інструментів глибокого браузерного профайлінгу. **Завдання**. Розробити математичну модель функціонування WebAR-застосунку на етапі

Сучасний стан наукових досліджень та технологій в промисловості. 2026. № 2 (36) *ISSN 2522-9818 (print)*
Innovative technologies and scientific solutions for industries. 2026. No. 2 (36) *ISSN 2524-2296 (online)*

ініціалізації сцени та декомпозувати процес завантаження на ключові етапи з оцінкою їхніх часових витрат. Провести емпіричне профілювання, ідентифікувати пікові навантаження на CPU та блокуючі задачі, а також сформувати обґрунтовані напрями подальшої оптимізації продуктивності. **Методи**. Дослідження продуктивності проведено шляхом емпіричного тестування продуктивності WebAR-застосунку під час найбільш ресурсоємного етапу – ініціалізації та завантаження тривимірної сцени. Збір даних щодо навантаження на потоки CPU в контексті веб-сторінки здійснювався за допомогою інструментів розробника Chrome DevTools. Апаратним середовищем для експериментів слугував мобільний пристрій Google Pixel 8, а статистична достовірність результатів забезпечувалася 10-кратною ітерацією кожного тестового сценарію. **Результат дослідження**. Розроблена математична модель формалізує процес завантаження сцени WebAR-застосунку та дозволяє провести оцінку завантаженості CPU на кожному етапі. За результатом аналізу часового профілю, отриманого з використанням удосконаленої математичної моделі та засобів Chrome DevTools виокремлено чотири етапи завантаження: ініціалізація порожньої сцени, завантаження 3D-об'єкта із сервера, парсинг моделі та її рендеринг. Встановлено нелінійний характер навантаження на процесор. Найбільш ресурсоємним виявився етап парсингу, під час якого зафіксовано пікове завантаження CPU і найдовша блокуюча задача, спричинена синхронною десеріалізацією бінарних даних. **Висновки**. Запропонований підхід до підвищення продуктивності завантаження сцени WebAR-застосунку на мобільному пристрої шляхом балансування навантаження CPU. Також дослідження підтвердило, що критичною фазою ініціалізації є парсинг 3D-моделі. З огляду на результати моделювання та проведених емпіричних досліджень, перспективним напрямом є впровадження поетапного завантаження та обробки 3D-контенту.

Ключові слова: web; augmented reality; WebAR-застосунок; performance evaluation; парсинг; рендеринг; Angular.

Бібліографічні описи / Bibliographic descriptions

Кучук Г. А., Матвєєв М. І., Косенко Н. В., Лисиця Д. О., Левченко А. О. Оцінювання продуктивності завантаження сцени WebAR-застосунку на мобільному пристрої. *Сучасний стан наукових досліджень та технологій в промисловості*. 2026. № 2 (36). С. 52–69. DOI: https://doi.org/10.30837/2522-9818.2026.2.052

Kuchuk, H., Matvieiev, M., Kosenko, N., Lysytsia, D., Levchenko, A. (2026), "Evaluating the performance of a webar application's scene loading on a mobile device", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (36), P. 52–69. DOI: https://doi.org/10.30837/2522-9818.2026.2.052