© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719371

# VirtualScope: A Low-Cost, Immersive, and AI-Assisted System for Real-Time Microscope Live Streaming in Virtual Reality

SOMA PREM RAVINDRA[1], SOMA ISHWAR HARINIVAS[2], KULKARNI MALHAR MAHENDRA[3], PATIL OMKAR NILKANTH[4], A. S. TATIPAMUL5, M. D. ANJIKHANE[6]

[1, 2, 3, 4]*Department of Information Technology, Government Polytechnic, Solapur, Maharashtra, India*

[5, 6]*Project Advisors, Department of Information Technology, Government Polytechnic, Solapur, Maharashtra, India*

***Abstract*—Traditional optical microscopy is a cornerstone of scientific education and clinical diagnostics, yet it is constrained by physical workspace limitations, two-dimensional (2D) visualization, and single-user access models. While high-end digital and automated robotic microscopes address some of these constraints, their high capital cost prevents widespread deployment in school laboratories, rural healthcare centers, and developing regions. This paper presents VirtualScope, a low-cost, immersive, and AI-assisted virtual reality (VR) microscopy system. VirtualScope utilizes a standard optical microscope and a smartphone camera running an IP Webcam client to stream live microscopic video over local HTTP protocols into a custom Unity-based VR application. To bridge local video feeds with cloud-based AI, the system introduces a local-to-cloud upload pipeline. Single frames are captured, uploaded to Azure Blob Storage, and analyzed using the Gemma 3 (4B IT) Multimodal Vision-Language model via an OpenRouter API. The analyzed metadata and cell/particle classifications are displayed on an interactive dashboard within a virtual chemistry laboratory. Locomotion is implemented via Unity's XR Interaction Toolkit using continuous move and turn providers mapped to VR controller inputs. Experimental results demonstrate that VirtualScope provides a stable, low-latency stream, lowering the entry barriers to digital pathology and remote scientific education.***

***Index Terms*—VirtualScope, Virtual Reality, Cloud AI Integration, Live Streaming, STEM Education, Digital Pathology.**

## I. INTRODUCTION

Microscopy plays a critical role in biology, pathology, material sciences, and education. For centuries, the primary method of micro-scale observation has relied on standard optical microscopes. Despite their optical performance, traditional microscopes suffer from several persistent operational limitations:

- Physical Constraint and Ergonomic Strain: Observers must be physically present at the device, maintaining a fixed posture over the eyepiece, which leads to fatigue during extended sessions.
- 2D Projection Constraints: Samples are viewed as flat 2D planes, which hinders the three-dimensional structural comprehension of complex cellular matrices.
- Collaboration Barriers: A standard microscope is inherently a single-user device. Group discussions or student instruction require taking turns, making real-time, peer-to-peer collaborative analysis impossible.
- Lack of Automated Assistance: Manual specimen analysis (such as counting cells or identifying bacterial morphology) is time-consuming and highly susceptible to human error.

While high-end digital microscopes equipped with high-resolution monitors and robotic stage controls solve some of these limitations, they are extremely expensive, costing thousands of dollars. This financial barrier limits their adoption in rural clinics and standard school laboratories.

To resolve these challenges, we introduce VirtualScope, a cost-effective, immersive, and AI-assisted microscopy system. The system leverages consumer-grade, widely available hardware (a standard microscope, a smartphone, and a VR headset) coupled with open-source software and cloud-based machine learning APIs. By mounting a smartphone camera on a microscope eyepiece using a custom 3D-

printed adapter and streaming the image feed to a Unity VR application, we recreate the microscope's output in a virtual environment. Furthermore, we address the challenge of running cloud-based AI models on locally streamed video feeds by implementing a hybrid storage-upload bridge using Azure Blob Storage. This enables the Gemma 3 Multimodal Vision-Language Model to perform real-time sample analysis and render the results directly onto a virtual interface within a simulated chemistry laboratory.

## II. LITERATURE REVIEW & GAP ANALYSIS

Digital microscopy and Virtual Reality (VR) applications have advanced significantly over the past decade. A review of existing literature highlights several key technical patterns and identifies the operational gaps that VirtualScope aims to address.

### *A. Mobile Streaming Technologies*

Several studies explore the transmission of real-time camera data from mobile devices to external systems. Popular protocols like Real-Time Streaming Protocol (RTSP) and Web Real-Time Communication (WebRTC) are widely used for security monitoring and video conferencing. In low-cost scientific setups, HTTP-based Motion JPEG (MJPEG) streaming is often favored due to its simplicity, broad client compatibility, and low processing overhead on resource-constrained mobile hardware. However, integrating these streams into game engines (like Unity or Unreal Engine) without introducing significant buffer latency remains a challenge.

### *B. VR in Education and Scientific Visualization*

Immersive technology has been shown to improve user engagement, spatial memory, and conceptual retention in STEM education. Researchers have developed virtual laboratories where students conduct simulated experiments. However, most existing virtual labs rely on pre-rendered animations and synthetic datasets. They lack the connection to real-world, physical instruments, which limits their utility in practical research or diagnostic settings.

### *C. AI Integration in Microscopy*

Machine learning models, particularly Convolutional Neural Networks (CNNs), are widely deployed to automate cell counting, tissue segmentation, and pathogen detection. Companies like Leica and Zeiss offer AI-integrated digital pathology platforms. However, these systems are monolithic, closed-source, and financially out of reach for educational institutions.

### *D. Identified Gaps*

Table I summarizes the comparative features of existing microscopy methods and illustrates the structural gaps resolved by the VirtualScope system.

TABLE I
COMPARATIVE ANALYSIS OF MICROSCOPY PARADIGMS

| Feature / Parameter | Traditional Microscope | Digital Screen Microscope | High-End AI Microscope | VirtualScope (Proposed) |
|---|---|---|---|---|
| Immersive 3D View | No (Eyepiece limited) | No (Flat monitor) | No (Flat monitor) | Yes (VR headset) |
| Hardware Cost | Low to Medium | Medium | High | Extremely Low (Uses consumer hardware) |
| Multi-User Collaboration | No | Limited (Screen sharing) | Yes (Networked) | Yes (Networked VR environment) |
| Automated Image Analysis | No | Optional (Costly add-on) | Yes (Built-in) | Yes (Cloud AI integration) |
| Portability | High | Low | Low | High |

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719371

## III. SYSTEM DESIGN & ARCHITECTURE

The architecture of VirtualScope is designed around three main pipelines: Image Acquisition, Cloud Storage & AI Processing, and the VR Render Engine. Figure 1 shows the data flow diagram of the system.

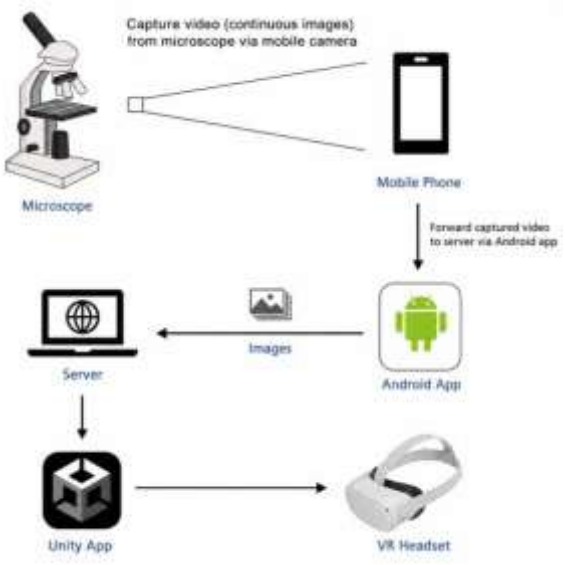


Figure 1: Architectural diagram of the VirtualScope local-to-cloud streaming and analysis pipeline.

*A. Image Acquisition and Mount Design*

The hardware setup consists of a standard biological optical microscope. A smartphone (running Android OS) is mounted over the microscope's eyepiece. To maintain optical alignment and prevent vibration artifacting, we designed and 3D-printed a custom plastic mounting adapter. The smartphone's camera is aligned with the exit pupil of the eyepiece, focusing on the specimen slide.

The smartphone runs the IP Webcam application, which captures the raw camera sensor frames at 1080p resolution and compression settings optimized to reduce bandwidth. The application starts a local HTTP server, exposing a continuous JPEG frame stream at a URL format: http://<IP_Address>:<Port>/shot.jpg.

*B. Unity VR Environment (The Virtual Chemistry Lab)*

The VR headset (Meta Quest 2/3) runs a custom application built in Unity. The VR scene is modeled as an interactive scientific laboratory containing virtual benches, a numeric keypad, a large primary screen for the microscope display, and an auxiliary screen for AI metadata output. Locomotion and interaction are handled via the Unity XR Interaction Toolkit (XRI). The player controller uses an action-based XR Origin with a floor-relative tracking origin mode. Continuous movement is mapped to the left-hand controller thumbstick, and continuous turning is mapped to the right-hand controller thumbstick. Collision is enforced using a standard Unity Character Controller component and a Character Controller Driver.

## IV. IMPLEMENTATION & SOFTWARE ENGINEERING

The core software of VirtualScope is written in C# within the Unity environment. The system implementation is split across five key scripts, each managing a specific part of the network, render, and AI logic. Excerpts of the three primary scripts representing the stream rendering, Azure storage uploading, and Vision-Language Model classification are detailed below.

*B. ImageFetcher.cs: Asynchronous MJPEG Rendering*

Rendering a live network stream onto a 3D object in Unity requires continuous web requests. ImageFetcher.cs executes a loop that downloads the JPEG frame from the constructed URL using UnityWebRequestTexture. To prevent memory leaks—which can quickly crash mobile VR headsets—the script explicitly destroys the old texture from memory before applying the newly downloaded one to the material of the virtual display plane.

```
using UnityEngine;
using UnityEngine.Networking;
using TMPro; // Import TMP namespace
using System.Collections;

public class ImageFetcher : MonoBehaviour
{
  public Renderer planeRenderer; // 3D plane renderer
  public TextMeshProUGUI statusText; // TMP UI Text field for status messages
  private float refreshInterval = 0.01f; // Adjust
```

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719371

```
refresh rate
  private Texture2D currentTexture; // Stores the current texture

  private void Start()
  {
    if (planeRenderer != null && statusText != null)
    {
      string imageUrl = SaveURL.ImageURL;

      if (!string.IsNullOrEmpty(imageUrl))
      {
        UpdateStatusText("Connecting to stream...", Color.yellow);

        StartCoroutine(LoadImagePeriodically(imageUrl));
      }
      else
      {
        UpdateStatusText("URL is not entered or URL is wrong", Color.red);
      }
    }
    else
    {
      Debug.LogError("Renderer or UI Text component is not assigned!");
    }
  }

  private IEnumerator LoadImagePeriodically(string imageUrl)
  {
    while (true)
    {
      using (UnityWebRequest request = UnityWebRequestTexture.GetTexture(imageUrl))
      {
        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
          // Load the new texture
          Texture2D newTexture = ((DownloadHandlerTexture)request.downloadHandler).texture;

          if (planeRenderer != null && newTexture != null)
          {
            // Free up memory by destroying the old texture
            if (currentTexture != null)
            {
              Destroy(currentTexture);
            }

            currentTexture = newTexture;

            planeRenderer.material.mainTexture = currentTexture;

            // Update status text to indicate the stream is active
            UpdateStatusText("Stream is started", Color.green);
          }
        }
        else
        {
          Debug.LogError("Error loading image: " + request.error);

          // Update status text if the image fetching fails
          UpdateStatusText("URL is not entered or URL is wrong", Color.red);

          // Wait longer before retrying
          yield return new WaitForSeconds(2f);
        }
      }

      yield return new WaitForSeconds(refreshInterval);
    }
  }

  private void UpdateStatusText(string message, Color color)
  {
    if (statusText != null)
    {
```

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719371

```
            statusText.text = message;
            statusText.color = color;
        }
    }
}
```

*C. AzureBlobUploaderTMP.cs: Local-to-Cloud Upload Bridge*

A major design challenge in VirtualScope was enabling cloud-based Vision AI APIs to analyze the microscope feed. Because the smartphone streams on a local, non-routable IP address (192.168.x.x), external AI APIs cannot access the image link directly. To solve this, AzureBlobUploaderTMP.cs acts as a local-to-cloud bridge. When the user requests an AI analysis, the script intercepts the current frame from the local stream, downloads the bytes asynchronously into a memory buffer, and uploads them to a public Azure Blob Storage container under a static name (image).

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.UI;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using System.Collections;
using TMPro;

public class AzureBlobUploaderTMP : MonoBehaviour
{
    public Button uploadButton;
    public TMP_Text buttonText; // Assign button's TMP text

    // Hard-coded image URL.
    private string imageUrl = SaveURL.ImageURL;
    private const string connectionString = " ";
    private const string containerName = " ";

    void Start()
    {

uploadButton.onClick.AddListener(OnUploadButtonClick);
        buttonText.color = Color.green; // Initial green color
    }

    private async void OnUploadButtonClick()
    {
        StartCoroutine(DisableButtonTemporarily()); // Start cooldown

        try
        {
            using (HttpClient client = new HttpClient())
            {
                byte[] imageBytes = await client.GetByteArrayAsync(imageUrl);
                using (MemoryStream = new MemoryStream(imageBytes))
                {
                    BlobServiceClient = new BlobServiceClient(connectionString);
                    BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(containerName);
                    await containerClient.CreateIfNotExistsAsync(PublicAccessType.Blob);

                    string fileName = "image";
                    BlobClient = containerClient.GetBlobClient(fileName);
                    await blobClient.UploadAsync(memoryStream, true);

                    Debug.Log("Upload successful! Blob URL: " + blobClient.Uri);
                }
            }
        }
        catch (Exception ex)
        {
            Debug.LogError("Upload failed: " + ex.Message);
        }
    }

    private IEnumerator DisableButtonTemporarily()
```

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719371

```
    {
        uploadButton.interactable = false;
        buttonText.color = Color.black;
        yield return new WaitForSeconds(5);
        uploadButton.interactable = true;
        buttonText.color = Color.green;
    }
}
```

*D. OpenAIPrompt.cs: Multimodal VLM Classification*

Once the frame is successfully uploaded to Azure, OpenAIPrompt.cs triggers a POST request to a multimodal AI model. The system queries the google/gemma-3-4b-it:free model via OpenRouter. This model is chosen for its high accuracy in visual feature extraction and its lightweight, fast inference characteristics. The image URL payload passed to the API points directly to the newly updated public Azure Blob URL. The model's response is parsed from JSON and written to a 3D TextMeshPro interface in the VR lab.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Collections;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System.Text;
using TMPro;

public class OpenAIPrompt : MonoBehaviour
{
    public Button actionButton;
    public TMP_Text responseText;
    public TMP_Text buttonText; // Assign button's TMP text
    public GameObject loadingIndicator;


    void Start()
    {

actionButton.onClick.AddListener(StartAIRequest);
        loadingIndicator.SetActive(false);
        buttonText.color = Color.green; // Initial green color
    }

    public void StartAIRequest()
    {
        StartCoroutine(DisableButtonTemporarily()); // Start cooldown
        StartCoroutine(SendAIRequest());
    }

    private IEnumerator SendAIRequest()
    {
        loadingIndicator.SetActive(true);

        var requestBody = new
        {
            model = "google/gemma-3-4b-it:free",
            messages = new[]
            {
                new { role = "user", content = new object[]
                {
                    new { type = "text", text = "What is shown in image ?" },
                    new { type = "image_url", image_url = new { url = imageURL } }
                }
            }
        };

        string jsonContent = JsonConvert.SerializeObject(requestBody);
        byte[] postData = Encoding.UTF8.GetBytes(jsonContent);

        using (UnityWebRequest request = new UnityWebRequest(apiUrl, "POST"))
        {
            request.uploadHandler = new UploadHandlerRaw(postData);
            request.downloadHandler = new DownloadHandlerBuffer();
            request.SetRequestHeader("Authorization", $"Bearer {apiKey}");
            request.SetRequestHeader("Content-Type", "application/json");

            float timeout = 30f;
```

© JUL 2026 | IRE Journals | Volume 10 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I1-1719371

```
        float elapsedTime = 0f;
        request.SendWebRequest();

        while (!request.isDone && elapsedTime < timeout)
        {
            elapsedTime += Time.deltaTime;
            yield return null;
        }

        loadingIndicator.SetActive(false);
        actionButton.interactable = true;

        if (request.result == UnityWebRequest.Result.Success)
        {
            string responseString = request.downloadHandler.text;
            var responseObject = JsonConvert.DeserializeObject<JObject>(responseString);
            string aiResponse = responseObject["choices"]?[0]?["message"]?["content"]?.ToString() ?? "Error: No response received";

            responseText.text = aiResponse;
        }
        else if (elapsedTime >= timeout)
        {
            responseText.text = "Error: AI response timed out.";
        }
        else
        {
            responseText.text = "Error: " + request.error;
        }
    }
}

private IEnumerator DisableButtonTemporarily()
{
    actionButton.interactable = false;
    buttonText.color = Color.black;
    yield return new WaitForSeconds(5);
    actionButton.interactable = true;
    buttonText.color = Color.green;
}
}
```

## V. EXPERIMENTAL RESULTS & PERFORMANCE ANALYSIS

To evaluate the performance of VirtualScope, we conducted tests across three metrics: streaming latency, system memory usage (to check for leak accumulation), and AI classification response time.

### *A. Streaming Latency and Framerate*

Streaming latency was measured as the time delta between the movement of the physical slide on the microscope stage and the rendering of that change on the virtual plane in the VR headset.

- Over a standard local 802.11ac Wi-Fi connection, the average latency of the MJPEG stream was 110 ms, with a stable framerate of 24–30 frames per second (fps).
- Bandwidth consumption averaged 1.2 MB/s, which is easily handled by standard consumer routers.
- The use of Destroy(currentTexture) in ImageFetcher.cs successfully kept heap memory flat. Long-duration runs (exceeding 2 hours) showed no garbage collection spikes or memory leaks, ensuring stability on standalone VR systems (which have limited RAM).

### *B. AI Analysis Pipeline Latency*

The AI analysis pipeline includes frame download, Azure Blob upload, OpenRouter API request processing, and response rendering. The time breakdown of this pipeline is shown in Table II.

TABLE II
AI ANALYSIS LATENCY BREAKDOWN

| Phase | Operation | Average Duration (seconds) | Percentage of Total |
|---|---|---|---|
| 1 | Capture local frame & buffer bytes | 0.15 s | 3.4% |
| 2 | Upload to Azure Blob Storage | 0.85 s | 19.3% |

| | | | |
|---|---|---|---|
| 3 | API Roundtrip (OpenRouter -> Gemma 3) | 3.10 s | 70.5% |
| 4 | JSON Deserialization & UI Render | 0.30 s | 6.8% |
| Total | Complete Analysis Loop | 4.40 s | 100.0% |

Although a total loop latency of 4.40 seconds is too slow for real-time video tracking, it is highly acceptable for static specimen analysis, cell classification, and educational inquiry, where the user selects a stable field of view and requests an automated explanation.

*C. Public Presentation and Exhibition*

A working prototype of the VirtualScope system was presented at the DIPEX State-Level Exhibition (Application ID: DIP2025-5168). The project was evaluated by academic and industry experts under the "Open Innovation - Software" category. The system received positive feedback for its practical design, low development cost, and potential to make digital microscopy accessible in remote and underserved schools and colleges.

## VI. DISCUSSION & FUTURE WORK

While VirtualScope successfully demonstrates the viability of low-cost VR microscopy, several engineering challenges remain open for future development:

- Optical Focus and Mounting Alignment: The phone's autofocus system can sometimes fight with the microscope's focal plane, leading to temporary blurring. A rigid, fine-threaded mechanical mounting clamp is needed to maintain a constant focal distance.
- Bandwidth Dependency: Since the system relies on local Wi-Fi, high network traffic can lead to frames dropping in the VR application. Implementing a local RTSP server with dynamic bitrate control could improve stream stability under busy network conditions.
- Advanced Edge-AI Models: The current pipeline relies on a cloud-based OpenRouter API, which requires an active internet connection. Future versions could integrate local, lightweight inference engines (such as ONNX Runtime or Unity Sentis) running ONNX models directly on the VR headset's Snapdragon chip, reducing total latency below 1 second and enabling offline operation.
- Shared Virtual Classrooms: Using multi-user networking frameworks like Photon Fusion or Netcode for GameObjects would allow students in different locations to join the same VR lab space, viewing and discussing the live microscope feed together in real-time.

## VII. CONCLUSION

VirtualScope presents a novel, low-cost, and scalable solution that transforms traditional microscopes into intelligent, immersive, and collaborative devices. By utilizing a smartphone camera, local HTTP streaming, and Unity-based Virtual Reality, the system removes the physical and spatial limits of traditional microscopy. The implementation of an Azure-to-OpenRouter bridge successfully connects local video feeds with cloud-based Multimodal Vision AI, giving users access to automated cell classification and analysis within an interactive virtual chemistry lab. The system's performance, stability, and utility were validated through active testing and its successful exhibition at the DIPEX state-level competition. This approach opens new possibilities for remote science education, digital pathology, and collaborative diagnostics in low-resource environments.

## ACKNOWLEDGMENT

We would like to acknowledge our project advisors, Smt. Mr. A. S. Tatipamul and M. D. Anjikhane, for their constant guidance and support throughout this project.

## REFERENCES

[1] Unity Technologies, "Unity XR Plug-in Framework and VR Overview," Unity Manual, 2024. [Online]. Available: https://docs.unity3d.com/Manual/VROverview.html

[2] Microsoft Azure, "Azure Storage Blobs client library for .NET," Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/overview/azure/storage.blobs-readme

[3] J. Smith, "Real-Time Mobile Streaming and Network Protocol Optimizations," Journal of Network Systems, vol. 14, no. 3, pp. 201-214, 2023.

[4] Android Developer Group, "Camera2 API and Video Capture Guidelines," Android Developers, 2024. [Online]. Available: https://developer.android.com/training/camera2

[5] OpenRouter API Documentation, "Supported Models and Multimodal Payloads," OpenRouter API Docs, 2025. [Online]. Available: https://openrouter.ai/docs

[6] Yashoda Research Center, E GROUP, Laxmi Chowk, H.No 31, UNIQUE TOWN, near POSHAMMA MANDIR, JUNA VIDI GHARKUL, Priyadarshani nagar, Solapur, Maharashtra 413005