big data and cognitive computing

MDPI

Article

An Approach to Enable Human–3D Object Interaction Through Voice Commands in an Immersive Virtual Environment

Alessio Catalfamo [1,*], Antonio Celesti [1,*], Maria Fazio [1], A. F. M. Saifuddin Saif [2], Yu-Sheng Lin [3], Edelberto Franco Silva [4] and Massimo Villari [1]

[1] MIFT Department, University of Messina, 98166 Messina, Italy; mfazio@unime.it (M.F.); mvillari@unime.it (M.V.)
[2] Department of Computing, Information and Mathematical Sciences, and Technology (CIMST), Chicago State University, Chicago, IL 60628, USA; asaif@csu.edu
[3] Department of Mechanical Engineering, Southern Taiwan University of Science and Technology, Tainan 71005, Taiwan; yushenglin@stust.edu.tw
[4] Department of Computer Science, Federal University of Juiz de Fora (UFJF), Juiz de Fora 36036-330, MG, Brazil; edelberto.franco@ufjf.br
* Correspondence: alecatalfamo@unime.it (A.C.); acelesti@unime.it (A.C.)

Abstract

Nowadays, the Metaverse is facing many challenges. In this context, Virtual Reality (VR) applications allowing voice-based human–3D object interactions are limited due to the current hardware/software limitations. In fact, adopting Automated Speech Recognition (ASR) systems to interact with 3D objects in VR applications through users' voice commands presents significant challenges due to the hardware and software limitations of headset devices. This paper aims to bridge this gap by proposing a methodology to address these issues. In particular, starting from a Mel-Frequency Cepstral Coefficient (MFCC) extraction algorithm able to capture the unique characteristics of the user's voice, we pass it as input to a Convolutional Neural Network (CNN) model. After that, in order to integrate the CNN model with a VR application running on a standalone headset, such as Oculus Quest, we converted it into an Open Neural Network Exchange (ONNX) format, i.e., a Machine Learning (ML) interoperability open standard format. The proposed system demonstrates good performance and represents a foundation for the development of user-centric, effective computing systems, enhancing accessibility to VR environments through voice-based commands. Experiments demonstrate that a native CNN model developed through TensorFlow presents comparable performances with respect to the corresponding CNN model converted into the ONNX format, paving the way towards the development of VR applications running in headsets controlled through the user's voice.

Keywords: virtual reality; automated speech recognition; convolutional neural networks; ONNX

check for updates

Academic Editor: Moulay A. Akhloufi

Received: 18 April 2025
Revised: 20 June 2025
Accepted: 27 June 2025
Published: 17 July 2025

Citation: Catalfamo, A.; Celesti, A.; Fazio, M.; Saif, A.F.M.S.; Lin, Y.-S.; Silva, E.F.; Villari, M. An Approach to Enable Human–3D Object Interaction Through Voice Commands in an Immersive Virtual Environment. *Big Data Cogn. Comput.* **2025**, *9*, 188. https://doi.org/10.3390/bdcc9070188

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Virtual Reality (VR) has been one of the most explored technologies in recent years because it represents essential technology for implementing the Metaverse, which is rapidly emerging as a new technological trend. It can be seen as a collective virtual shared environment where people can interact, socialise, work, and play. The main purpose that the Metaverse aims to achieve is the creation of an immersive experience for users in a VR

environment. For this reason, recently, lots of research initiatives have tried to improve all possible methodologies that allow a better immersive experience for end users.

Currently, the VR market includes different kinds of headset devices to access VR environments and applications. Some of these devices are tethered to a Personal Computer (PC), which allows for deploying loosely coupled VR applications compliant with the device's operating system. In other cases, tightly coupled VR applications are strongly dependent on the software/hardware configuration of the device. In fact, developing and deploying a VR application in a headset integrated with an Automatic Speech Recognition (ASR) system and allowing interaction with the user through their voice commands, as depicted in Figure 1, is often strongly constrained to the hardware/software of the device. This paper is motivated by the fact that current scientific literature lacks a comprehensive methodology to develop and deploy a VR application integrated with an ASR system into hardware/software-constrained headsets, such as, for example, Oculus Quest [1]. In fact, a general methodology to control objects in a VR environment through users' voice commands has not been properly investigated by the scientific community. The ASR improves the user's experience in VR applications for different reasons. By enabling hands-free interactions, the ASR system can improve accessibility, which is particularly beneficial in VR environments, where traditional hand-based controls are not usable or convenient. Adaptive ASR systems enhance the user's immersion, inclusivity, and accessibility, benefiting individuals across age groups and ability levels. Furthermore, it is also fundamental for people with motor and/or cognitive impairments.



Figure 1. Reference scenario including a user accessing a VR application with a headset device and controlling 3D objects through voice commands.

Thanks to our methodology, the VR application running on the headset can infer the environmental audio, e.g., the user's voice, and use it to apply any given command to control 3D objects in the VR environment, exploiting a Convolutional Neural Network (CNN) model. In particular, we start from a CNN model, enabling an ASR system that can be used in a VR application; then, it is necessary to transform it into an Open Neural Network Exchange (ONNX) model, the open standard format for Machine Learning (ML) interoperability, which is commonly supported by standalone headset devices such as Oculus Quest. Although a more complicated model (e.g., Transformer) can face the ASR problem more efficiently when considering complex audio [2], in this paper, we considered a CNN model that is more lightweight to deploy within the headset with limited hardware resources, but that is, at the same time, suitable for recognizing simple user voice commands.

This work starts from our previous work in which we developed a VR application integrated with an ASR system compatible with the Oculus Quest headset device [3]. Since headset devices are typically software/hardware constrained, in order to run any kind of CNN model within them, in this paper, we discuss an alternative methodology consisting of converting an existing CNN model for ASR into the ONNX format, which is compatible with most commercial headset devices. The contributions of this research work are twofold:

- Design of a methodology to integrate ASR systems in VR applications deployable in any headset device supporting ONNX;
- Discussing a possible prototype developed by using Unity 2023.2.17, ONNX 1.19.0, and PyTorch 2.7.1 technologies.

The remainder of this paper is organised as follows: Related works are discussed in Section 2. The method adopted to design our solution and a prototype implementation are discussed, respectively, in Sections 3 and 4. A case study in which 3D objects are moved in a VR application through the user's voice is discussed in Section 5. Experiments proving the effectiveness of our solution are discussed in Section 6. In the end, final considerations and future works are described in Section 7.

2. Background and Related Work

The speech recognition problem is one of the most investigated problems in AI research trends. Several technologies have been explored and implemented to find a more optimal solution. Starting from the classical Artificial Neural Networks (ANN) [4] until innovative and complex models like Transformers [2,5], several strategies have been exploited to treat speech recognition problems with state-of-the-art methods. In this paper, we propose a methodology for implementing speech recognition in virtual reality that involves a specific machine learning model (CNN). Hence, this section will delve into the work in speech recognition within virtual reality. Several works have tried to exploit speech recognition in a VR environment, approaching the problem in different ways and suggesting several solutions. In [6], the author proposes a medical speech recognition use case to simulate an eye examination simulation in a VR context. Speech recognition, in this work, is exploited to instruct the virtual patient. The work mainly focuses on face validity in VR, and the conclusions are related to it.

Medical use cases are frequent in VR topics, and speech recognition can be a considerable tool to facilitate the final user, i.e., the patient. For example, in the work proposed in [7], the VR is exploited to design a solution for the social interactions in children with autism spectrum disorder. The solution exploits both face and speech recognition to provide a virtual environment in which a training system for enriching social interactions is carried out. Another similar use case for using VR for social skills is described in [8]. In particular, the work carried out here faced the limitations of conversational AI in the existing training tools for social interactions. The proposed work reflects on enriching a specific social context, considering a given organisational element in a particular culture. The use of speech recognition in this specific case is minimal. In [9], another solution regarding the social science field is explored. Even in this case, VR is exploited to improve social skills and to overcome issues related to social anxiety. The research in the paper faced the problem of social phobia, and the proposed solution exploits voice and heart rates to examine the emotional and physical symptoms of social anxiety. The work proposed in [10] deals with a mobile VR application and speech recognition, focusing on the problem of children's phonological dyslexia. The application of VR and speech recognition to the issues related to racial inequities in police use of force is discussed in [11]. In particular, VR is exploited to set experimental settings that better mimic officers' experiences. Here, speech recognition, according to the paper, is the victim of an intense accuracy degradation. The solution designed in [12] does not exploit VR but uses Augmented Reality (AR) to create an Android-based application that can recognise voice commands to execute different actions in a car environment. Unlike the other solutions considered here, the work implements an augmented reality, but it is essential to believe it because the implementation technologies are analogous. The scientific literature introduced speech recognition in the Metaverse context, which is also carried out thanks to VR technologies. The work proposed in [13]

proposes a speech recognition solution in the Metaverse that combines neural networks and traditional symbolic reasoning. The solution attempts to create an aircraft environment, and the speech recognition is exploited to understand users' requests and replies based on context and aircraft-specific knowledge. Meanwhile, the research carried out in [14] uses machine learning and, in particular, speech recognition inside the Metaverse and the real world to recognise speech emotions. Another interesting aspect, very related to the solution proposed in our work, is how, in virtual reality, speech recognition can be used as the source of specific commands and instructions. The research discussed in [15] designs an algorithm for the interaction of humans with machines in which the gestures in driving a virtual hand are estimated through speech recognition.

All of the solutions presented and described here implement a specific approach for applying speech recognition in virtual reality. However, these works typically focus on narrow use cases, often lack technical depth in the implementation details of the speech recognition pipeline, or do not offer generalizable methodologies suitable for broader applications. In contrast, the solution proposed in this paper introduces a more structured, systematic approach that integrates speech recognition into VR using a dedicated machine learning framework (CNN-based), with clear stages for data acquisition, training, inference, and evaluation. Our methodology is designed not only to demonstrate feasibility in a single context but also to be adaptable and reproducible across different VR applications. This makes our contribution more robust and scalable compared to previous works, which often remain at a proof-of-concept level without offering replicable design guidelines.

3. Method

In this Section, we discuss our methodology to develop and deploy a CNN for an ASR system into hardware/software-constrained headsets, as well as the Oculus Quest device. Specifically, starting from the description of a system architecture allowing software developers to create a VR application with which users can interact through their voice commands, we focus on voice preprocessing and voice recognition mechanisms running in the headset device. Moreover, we also examine the CNN, which is involved in the flow of speech processing and recognition.

3.1. System Architecture

Figure 2 shows our system architecture enabling speech recognition in VR applications. In particular, we generally acquire the user's voice through the microphone of a headset device to control the VR application. The system architecture consists of two main layers: the System Application Layer and the Edge Computing Layer.

- **System Application (Layer 1)**. It includes all the components that allow for processing the user's voice and the software module for controlling 3D objects in the VR application. Specifically, layer 1 is composed of two key software modules:
  - *Keyword Spotting System.* It is the component that processes the captured user's voice and the consequent speech recognition. In particular, it includes the following:
    - *Speech Processing module*. It is designed to capture the user's voice, extract its characteristics, and convert it into a format suitable for the input of a classification model based, in our case, on CNN. This module is split into two logical components: *Audio Feature Extraction* and *Speech Decoder*. The first one is responsible for preprocessing the audio signal and extracting the features used to train the model and infer the audio signal. The second one maps the extracted features with a specific model that can exploit the acoustics of the pronounced word, its linguistics, or its statistical information within a context. In our case,

we will consider a specific model that exploits the acoustic characteristics of the input audio signal.

* *Speech Recognition Module*. It is the logical component that combines the algorithms for identifying keywords from the audio by applying speech recognition algorithms and models. In particular, the Text Parser and Word Matching modules allow text separation and the selection of algorithms for comparing words within the text.

– *Virtual Reality module*. It is the software component that allows the integration of the CNN model for ASR in the VR application, enabling the interaction between the user's voice acquired from the real world and the VR environment. Furthermore, with the *Scene Module*, it is possible to define VR scenes with 3D objects with which to interact. The *Controller Module* allows for controlling the actions based on the input specified by the user's voice.

– *Interaction Module*. It is the component that provides a high-level interface enabling the use of the ASR system in the VR application.

- **Edge device (Layer 0)**. It is part of the architecture close to the user, and it consists of a headset device equipped with a microphone that allows the acquisition and processing of the user's voice. The user's voice is continuously captured in real-time through the headset's microphone. This raw audio data is then optionally preprocessed (e.g., by denoising, normalisation, feature extraction tasks) before being forwarded for training or inference.

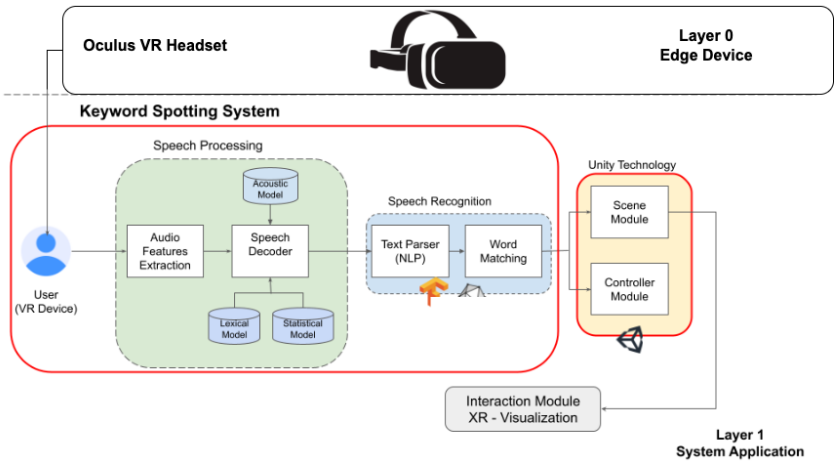


Figure 2. System architecture enabling user control of 3D objects with their voice in a VR application deployed in a headset device.

3.2. VR Application Voice Command Preprocessing in the Oculus Headset

Preprocessing the user's voice commands involves extracting the MFCC coefficients [16]. Generally, an acquired audio signal must be split into samples of the same size and subsequently processed by the MFCC extraction algorithm. Moreover, these coefficients are robust and reliable for variations in speakers and recording conditions. Figure 3 shows the algorithmic steps used for the proposed system:

- **Pre-emphasis**. It is a signal processing technique in which the weakest and highest frequencies are modulated before transmission to improve the signal-to-noise ratio.
- **Frame Blocking** This processing step allows us to keep the information by overlapping the different frames of the audio signal.
- **Windowing**. It is the step in which the window function is applied to the different segments of the audio signal to reduce leakage effects. A window function is a

symmetric function whose value is zero outside a defined range and reaches its maximum near the middle.

- **Discrete Fourier Transform (DFT)**. It is the transform used to analyse signals to convert them from the time domain to the frequency domain. It is possible to filter the signal from the spectrum of frequencies by removing the noise and analysing its single frequencies. The DFT operates on continuous functions and produces continuous outputs.
- **Mel Frequency Filter Bank**. In this processing phase, the signal is separated into frequency bands in the MEL Frequency space, where the perception of non-linear human sound is simulated.
- **Cepstrum**. It is processed by applying the Fourier transform to the decibel spectrum of a signal. This parameter is generally used to analyse the rates of change in the spectral content.
- **Logged Energy**. This parameter represents the average log energy of the audio signal in the input to the function.
- **Delta**. This function block allows for extracting and identifying differences in signal features to consider the sequence of transitions between phonemes.

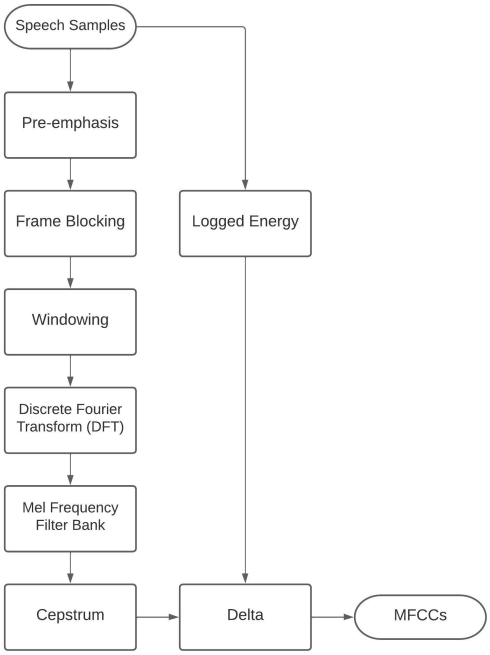


Figure 3. Block diagram of MFCC extraction algorithm.

The MFCC coefficients are used for the processing and recognition phases. In particular, as we will discuss below, they will be the input of a CNN used to process and recognise the user's voice. The preprocessing phase and the subsequent training process are performed offline during a preliminary phase. In this phase, the CNN model is trained using a labelled dataset of speech audio, where MFCC features are extracted and used to learn meaningful patterns. Once the training is completed, the resulting trained model is optimised and deployed in the headset device. During the operational (a posteriori) phase, the model is used for inference: the headset device captures real-time user voices, extracts MFCC features, and feeds them into the pre-trained CNN to perform on-device ASR efficiently.

3.3. VR Application's Voice Recognition Model in the Oculus Headset

The *Speech Decoder* and the *Speech Recognition* components are completed through the design and implementation of a CNN that is trained on a dataset that includes different voice signals labelled according to brief, simple voice commands [17]. This trained model is used to classify and identify the user's voice commands to interact with 3D objects in the VR application. Specifically, we adopted the optimised CNN model for self-speaker command recognition already discussed in our previous work [3].

Table 1 shows the CNN's configuration for each involved layer.

Table 1. CNN model configuration.

Layer	Input Shape	Output Shape	Activation
Input Layer	(44, 13, 1)	(44, 13, 1)	None
Conv2D	(44, 13, 1)	(42, 11, 64)	ReLU
BatchNormalization	(42, 11, 64)	(42, 11, 64)	None
MaxPooling2D	(42, 11, 64)	(21, 6, 64)	None
Conv2D	(21, 6, 64)	(19, 4, 32)	ReLU
BatchNormalization	(19, 4, 32)	(19, 4, 32)	None
MaxPooling2D	(19, 4, 32)	(10, 2, 32)	None
Conv2D	(10, 2, 32)	(9, 1, 32)	ReLU
BatchNormalization	(9, 1, 32)	(9, 1, 32)	None
MaxPooling2D	(9, 1, 32)	(5, 1, 32)	None
Flatten	(5, 1, 32)	(160)	None
Fully Connected	(160)	(64)	ReLU
Dropout	(64)	(64)	None
Fully Connected	(64)	(5)	Softmax

The CNN has been previously trained considering the following parameters:

- Epochs: 28;
- Batch Size: 32;
- Optimizer: Adam function [18];
- Learning Rate: 0.0001.

The parameters described in Table 1 are *input_shape*, which defines each layer's input, *output shape,* and the activation function.

We adopted the Google speech dataset https://research.google/blog/launching-the-speech-commands-dataset (accessed on 24 August 2017). The dataset was split into a training set (70%), a test set (20%), and a validation set (10%) to train the model.

4. System Prototype

This section describes a possible prototype implementation of a VR application controlled by the user's voice command, considering the Oculus Quest device and the Unity real-time development platform. Specifically, we firstly focus on the implementation and conversion of the CNN model for ASR into the Open Neural Network Exchange (ONNX), i.e., the only possible format for integrating a CNN model for ASR with a VR application developed through Unity. Secondly, we focus on the VR application development with particular attention to the integration of the ASR system based on an ONNX-converted CNN model.

4.1. CNN Model Conversion in the ONNX Format

The user's voice interacting with 3D objects of the VR application can be carried out through the integrated directional microphone of the headset.

After the implementation and training of the CNN model were completed, in our case, using the Keras tool library, it was necessary to convert it to the ONNX format in order to be compatible with the Unity environment used for the development of the VR application.

The ONNX standard was chosen as it supports many operators and implements different accelerators based on the choice of headset device.

Model Inference in Unity

Figure 4 shows a Netron graph representation, where each node indicates an operation performed by the forward step of the described CNN. Moreover, the properties of each node of the graph are depicted, including the attributes, the input, and the output of each layer. This visualisation allows us to have a clearer view of the convolutional layers that characterise the network.

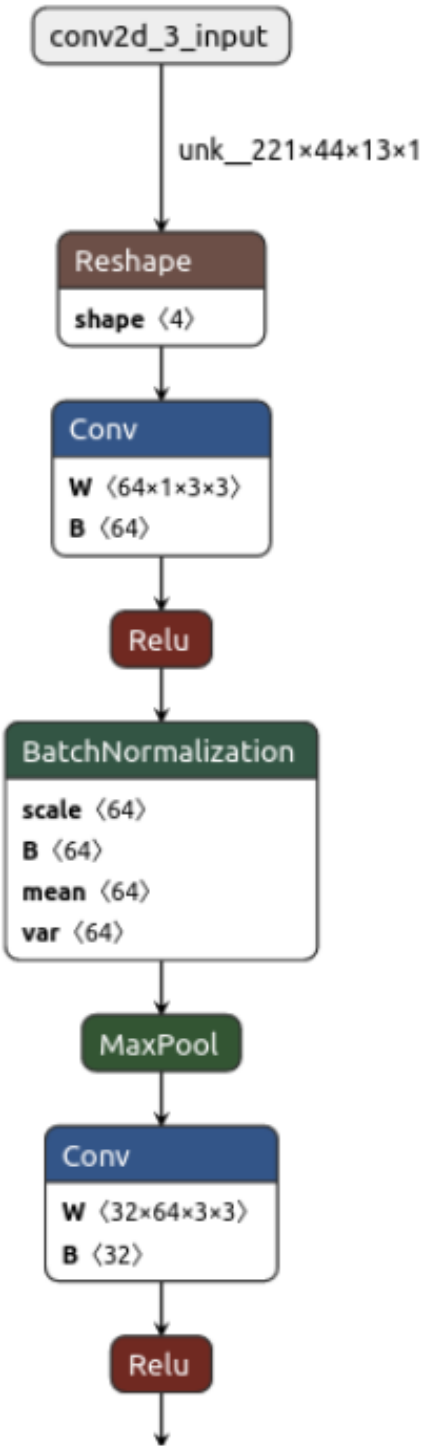


Figure 4. Netron Graph model representation.

For the implementation of the CNN model in the Unity Engine, it is necessary to use different frameworks and tools such as *Barracuda* [19] and *Accord.NET* [20]. Once the VR application has been created through Unity and the 3D objects have been imported, the Barracuda 1.0.4 module must also be imported by inserting a manifest in the project's Package folder. The ASR system based on the ONNX-converted CNN has to be included in the project assets. Particular attention has been paid to the *SharpDX5* [21] library, which has been imported to obtain high computing performance.

4.2. Audio Signal Processing

The *Audio Features Extraction* submodule, described in Section 3, consists of extracting MFCCs [22] used to infer the input processing signal. Listing 1 describes the implementation of the user's voice processing in the Unity environment.

Listing 1. Convertion of the MFCC list into tensor for the ONNX model inference.

```
    public class MelCoeff : MonoBehaviour
    {
        # Getters and Setters
        public int frame { get; set; }
        public double[] mfccs { get; set; }
    }

    # Convert extracted MFCCs into Tensor for ONNX Model Inference
        public Tensor convertMFCCs(List<object> mfccs, int n_frames = 44, int
     n_mfccs = 13) {
        # Define a new tensor
        Tensor tempTensor = new Tensor(new[] { 1, n_frames, n_mfccs, 1 });
        # Get generic obj type for extracting attributes
        Type myType = mfccs[0].GetType();
        # Get generic obj properties
        IList<PropertyInfo> props = new List<PropertyInfo>(myType.
    GetProperties());
        # Populate tensor
        for(int i = 0; i < n_frames; i++){
            # Get each frame
            foreach (PropertyInfo prop in props){
                object propValue = prop.GetValue(mfccs[i], null);
                # Get list of double[] associated to MFCCs
                if (propValue.GetType() == typeof(double[])){
                    double[] mfccs_values = (double[])propValue;
                    # Populate tensor
                    for (int j = 0; j < n_mfccs; j++){
                        tempTensor[0, i, j, 0] = (float)mfccs_values[j];
                    }
                }
            }
        }
        return tempTensor;
    }
```

The *MELCoeff* class has been defined to access objects from the *Accord.Audio"* and *Accord.Audio.DirectSound* modules. The method of the Features Extraction class, defined in Listing 1, is used for processing audio features and converting them into a tensor format suitable for making inferences. The *IList interface* allows accessing a generic list of fixed size; in this way, it is possible to manage the samples of the recorded audio file. Each audio sample is composed of 44 frames and 13 MFCCs since the input layer of the ONNX convolutional network accepts as input a (44, 13) matrix.

The code related to audio processing is shown in Listing 2.

Listing 2. FeatureExtraction class for user's voice processing.

```
public class FeatureExtraction : MonoBehaviour
{
    #  Audio processing
    private Signal audio_file, splitted_audio;
    private MelFrequencyCepstrumCoefficient mfcc;
    private string audio_file_path;
    private IEnumerable mfccs_output;
    public List<object> extractFromAudio(){
        # Get the audio file name
        audio_file = signal.FromFile("Assets/Recordings/temp_rec.wav");
        # Get first audio second - 22050 sr
        if (audio_file.Length >= 22050){
            audio_file = audio_file.Split(windowSize: 22050, step: 22050)[0];
        }

        # Extract MFCCs

        mfcc = new MelFrequencyCepstrumCoefficient(samplingRate: 22050,
    cepstrumCount: 13, frameRate: 44);
        mfccs_output = mfcc.Transform(audio_file);
        List<object> mfccs_list = mfccs_output.Cast<object>().ToList();

        return mfccs_list;
}
```

The *FeaturesExtraction* class was used for processing the user's voice signal, which is loaded using the *Signal* class provided by the Accord.NET Framework. The audio file is then split according to the defined sample rate, and the MFCCs are extracted for each sample.

4.3. ONNX Model Interference

For executing a CNN within Unity, it is necessary to use the *IWorker Interface* provided by the Barracuda module. The model must be loaded, and a Worker instance must be created to run it in Barracuda. Furthermore, to start the CNN in Unity, it is necessary to create a worker object of the *WorkerFactory* class. The *ONNXModel* class, whose code is highlighted in Listing 3, provides methods and variables for processing and classifying the user's voice signal. This class's constructor takes the Keras neural network model converted to the ONNX format as input.

Listing 3. Method to create the CNN model in ONNX format.

```
public class ONNXModel : MonoBehaviour
{
    # ONNX Model
    [SerializeField]
    private NNModel KerasModel;
    private Model runtimeModel;
    private IWorker worker;
    private string outputLayerName;

    public ONNXModel(NNModel KerasModel){

        this.KerasModel = KerasModel;
    }
}
    #Start is called before the first frame update
    public void CreateModel(){
        // NN Model and Worker Interface
        this.runtimeModel = ModelLoader.Load(this.KerasModel);
```

```
        this.worker = WorkerFactory.CreateWorker(WorkerFactory.Type.Auto,
    this.runtimeModel);

        # Get output layer name, last index
        this.outputLayerName = this.runtimeModel.outputs[this.runtimeModel.
    outputs.Count - 1];
    }
```

The ONNX-converted CNN model is loaded in the memory graph representation. The worker can break down the neural network into small tasks, as described previously. The *outputLayerName* variable maps the prediction result to the correct word (e.g., yes, no, up, down, ...). The *createModel* method is invoked to initialise the parameters to make an inference.

4.4. VR Application and 3D Objects

To develop an immersive virtual scene, we used Unity and the Oculus Quest device. Moreover, we use the Oculus Developer HUB [23] for configuring the Oculus Quest and monitoring performance. In particular, this software helps measure application performance, update headset drivers, and so on. For the creation of the VR application to be deployed on Oculus Quest, the following packages have been implemented:

- Oculus Integration SDK v43.0;
- XR Plug-in Framework.

The XR Plug-in enables XR hardware and software integrations for multiple platforms. XR initialisation on startup has been enabled for the project configuration on both Android and Windows systems using Oculus as the provided plug-in.

Unity Audio Processing System

To develop the audio processing system, we used the *Recorder* class, whose code is shown in Listing 4, which provides the required variables and methods. In our system prototype, we used the microphone integrated in the Oculus Quest headset. The recordings are saved in memory and processed in real-time to activate the ASR system inference through the ONNX-converted CNN model.

Listing 4. Recorder class for capture audio stream from input device.

```
public class Recorder  : MonoBehaviour
{
    # Capture audio signal
    private WaveIn inputSignal;
    private WaveFileWriter fileWriter;
    private String dir_file_path;
    private String file_name;
    private int capture_device_index;

    #Constructor
    public Recorder(int inputDeviceIndex, String filePath, String fileName){
        this.file_name = fileName;
        this.dir_file_path = filePath;
        this.capture_device_index = inputDeviceIndex;
    }
}
```

The constructor of this class takes as input parameters the index of the audio device, the name of the audio file, and the path of the directory where it will be saved in memory. The *Application.persistentDataPath* is a constant used to indicate the persistent storage path

on Android devices. The *WaveIn* inputSignal is defined for acquiring audio input, and, finally, the *fileWriter* variable saves the data stream in memory.

5. Case Study: Moving 3D Objects With User's Voice in a VR Application

This Section presents a case study in which we implemented a VR application where 3D objects are controlled with the user's hand or voice.

In the first case, 3D objects can be grabbed by the user's hand movement, whereas in the second case, leveraging the ONNX-converted CNN model, the user is able to grab 3D objects with their voice command. In particular, we developed different common 3D objects within a VR scene and implemented different actions triggered by the user's voice.

In particular, as shown in Figure 5, we created a scene in the VR application that included a cup and a torch. The 3D objects are placed on a desk and can be grabbed and moved using the user's hands or voice. It is also possible to specify the grip types that are supported. For example, it can be set to pinch, palm, all, or none.

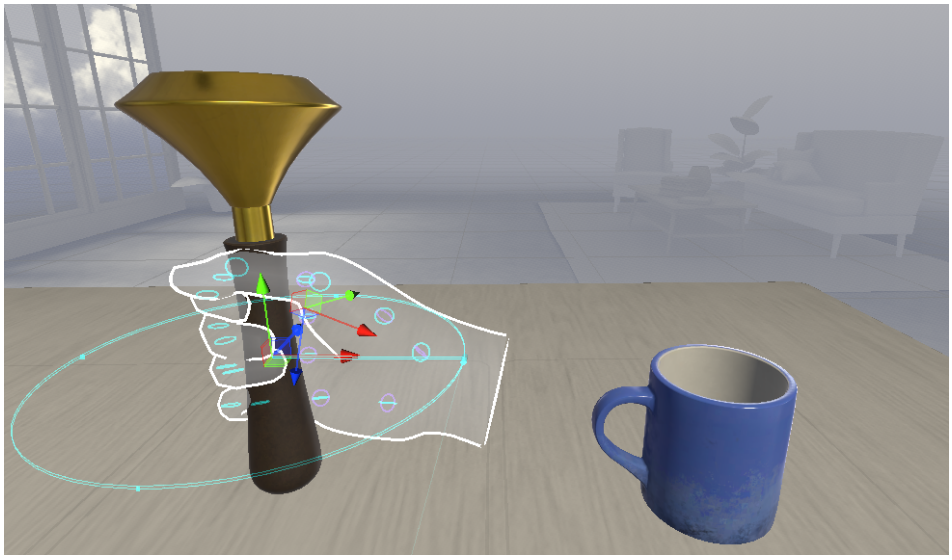


Figure 5. Example of a scene in the VR application in which the user can grab a cup or a torch either with their hand or voice.

Interaction Through Voice Command

In our use case, the user can experience the simulation in room-scale mode within a diagonal area of up to 4 m. Room-scale VR is a design paradigm that allows users to map real-life motion with movement within the VR environment. For 3D object interactions, it is possible to enable touch controllers and hand tracking using the headset's external cameras. The 3D object used for the basic interactions is the controller hands. Specifically, the child game object "Handgrab Point" captures these 3D objects at specific points.

The tree represented in Figure 6 shows the prefab controller hands hierarchy in the Unity real-time development platform. The touch controllers/hands are divided into the left controller/hand and the right controller/hand, and both are composed of DataSource and Interactors Unity objects. DataSource objects acquire motion-tracking data or button inputs; the Interactor objects define the different points of contact with the objects of the scene.

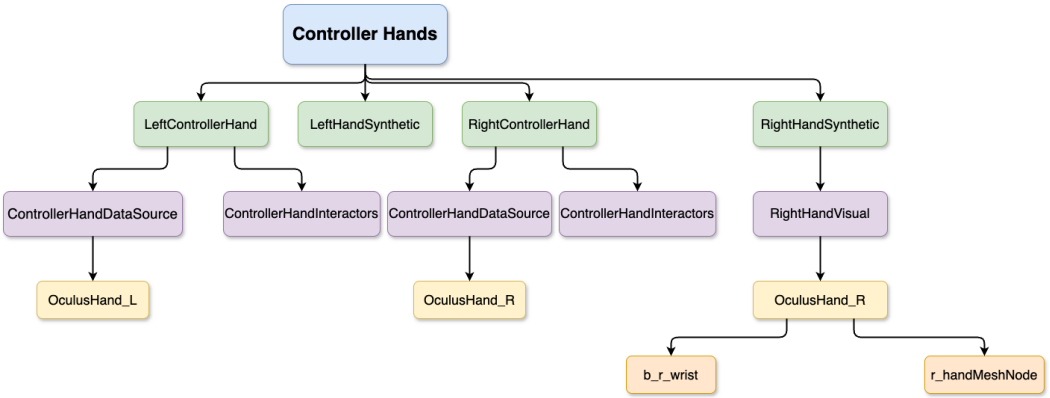


Figure 6. Hierarchy of hand controllers in Unity.

Figure 7 shows the hand-tracking models that can be rendered in the 3D environment. This feature allows using hands as input devices. To enable hand tracking, the OVRCameraRig was imported into the project, and the hand prefabs were added to its TrackingSpace, in particular, the right and left-hand anchors. After integrating the script for interactions, the feature "auto switch between hands and controllers" was enabled. This feature allows one to select the use of hands when the touch controllers are placed down. There are many ways in which the 3D object can be grabbed: these are defined based on the data obtained by the tracker at the time of contact with the 3D object.

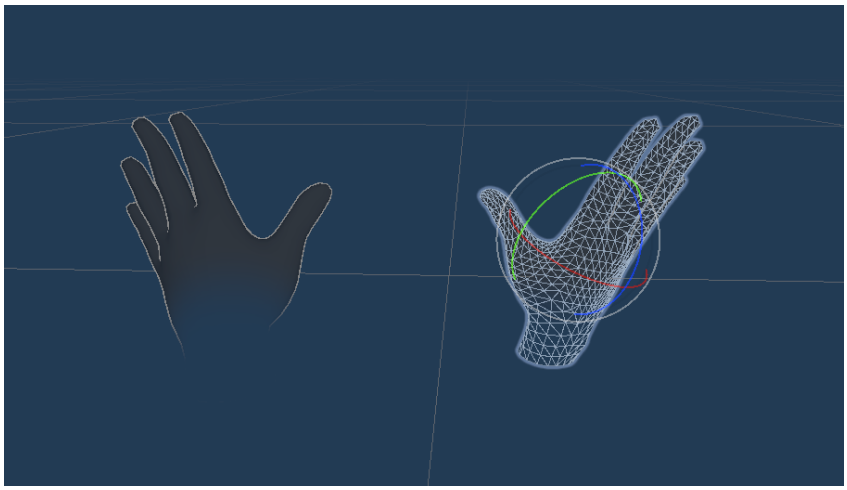


Figure 7. Oculus Quest hand tracking interaction model.

Figure 8 shows the joints of the VR hand. This information is stored in variables of type Transform. In Unity, these variables are used to manipulate the scale, position, and rotation of the different game objects of the scene.

In the case of the user's voice command, the user's hand autonomously moves through the 3D object and grabs it. In this case, the hand movement and 3D object grabbing were triggered by the user's voice command recognised by the ONNX-converted CNN model of the ASR system integrated in the VR application. The complete sequence of tasks includes:

1. The user presses the Oculus Quest headset controller button to trigger audio recording;
2. The system captures the user's voice input through the headset microphone;
3. The recorded audio is processed by the ONNX-converted CNN model of the ASR system;
4. The recognised voice command is parsed and interpreted by the scene controller;
5. The automatic hand movement action toward the 3D object is executed;
6. The automatic 3D object grabbing action is performed.

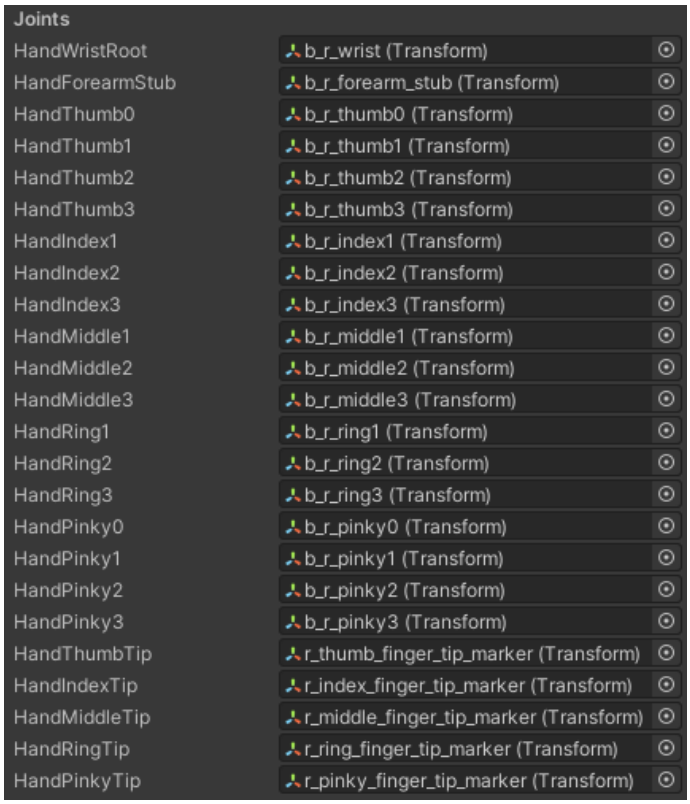


Figure 8. VR hand joints configuration.

6. Experiments

This Section presents an experiment that was carried out to validate the proposed system architecture. As described, the optimal CNN model was converted into an ONNX model and integrated into the implemented immersive VR application. For this reason, we analysed the inference phase considering the two frameworks involved in our implementation (i.e., Tensorflow and ONNX) in a simulated environment, and we evaluated the final implementation in a real environment.

6.1. Testbed Setup

As mentioned, we exploit a real environment for the experimental evaluation, not a simulated one. Indeed, we deploy our implementation and test it in an Oculus Quest device with the following features:

- **Panel Type:** Dual OLED (1600 × 1440);
- **Refresh Rate:** 72–60 Hz;
- **CPU:** Qualcomm Snapdragon 835;
- **GPU:** Qualcomm Adreno 540;
- **Memory:** 4 GB;
- **Motion Tracking:** Inside-out technology.

For the comparison evaluation between Tensorflow and ONNX technologies, we used a specific machine where it was possible to deploy the model through the two frameworks. It is configured as follows:

- **CPU**: Intel-Core i5-9600K 3.7 GHz;
- **GPU**: NVIDIA GeForce RTX 2080 Ti;
- **RAM**: HyperX DDR4 (4 x 4GB);
- **Disk**: SSD AORUS NVMe (1TB);
- **Motherboard**: Z390 AORUS PRO.

6.2. Performance Comparison—ONNX vs. Keras

In our performance evaluations, we consider it necessary to manually analyse the model inference to compare the two frameworks used (i.e., TensorFlow Keras and ONNX) for model implementation and inference.

Figure 9 shows the inference tests performed on the ONNX and Keras models on a sample of 100, 250, 500, 750, 1000, 1250, and 1500 instances. The Y-axis reports the execution time of the models in seconds. The ONNX model performs better than the Keras model when inference is conducted for more than 650 instances. Therefore, ONNX has faster execution times with inference on large amounts of data.

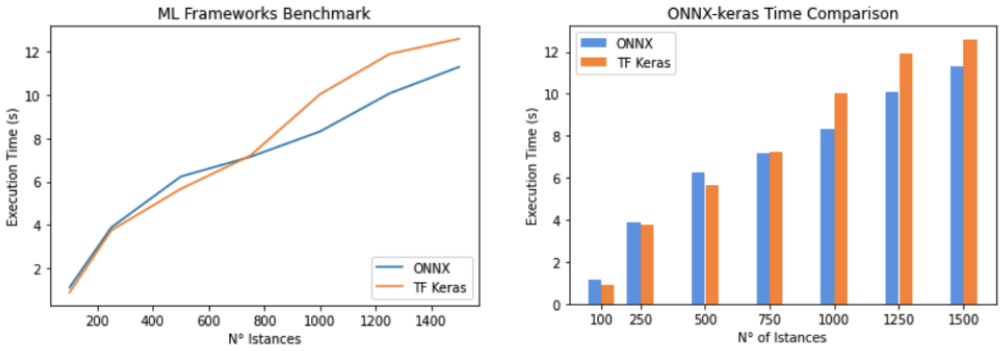


Figure 9. ONNX-Keras ML framework benchmark and time comparison.

Although no formal hypothesis testing (e.g., *t*-test or ANOVA) was performed, the standard deviation across repeated runs was consistently low, suggesting stable measurements. For large input sizes (above 650 instances), the reduction in inference time for ONNX over Keras exceeds 10%, indicating a practical and potentially statistically significant advantage. Future work may include paired statistical tests to confirm that the observed differences are not due to random variation, with confidence levels (e.g., $\alpha = 0.05$) supporting claims of significance.

6.3. Converted Model Accuracy

To evaluate the efficiency of the ONNX and the Tensorflow Keras-based models, we compared the probability distribution of the results obtained using the same input data.

Tables 2–6 depict the distribution probabilities performed by both kinds of models. The distribution probabilities are similar for the two types of machine learning frameworks. In conclusion, the converted ONNX model has maintained the same features as the original Tensorflow Keras model.

While this evaluation is based on point estimates from individual examples, the agreement between ONNX and Tensorflow across all test cases suggests that the conversion introduces no systematic bias in prediction. Future work will include statistical correlation and KL-divergence tests to quantify the similarity of output distributions with greater rigour.

Table 2. Probability distribution for the "yes" audio example using ONNX-based and Tensorflow-based models.

"yes" Audio Example	**Probability Distribution**				
	Yes	**No**	**Down**	**Up**	**One**
ONNX	0.964	0.021	0.015	0	0
Tensorflow	0.973	0.011	0.016	0	0

Table 3. Probability distribution for the "one" audio example using ONNX-based and Tensorflow-based models.

"one" Audio Example	Probability Distribution				
	Yes	No	Down	Up	One
ONNX	0.11	0.030	0.017	0.008	0.934
Tensorflow	0.014	0.021	0.012	0	0.953

Table 4. Probability distribution for the "no" audio example using ONNX-based and Tensorflow-based models.

"no" Audio Example	Probability Distribution				
	Yes	No	Down	Up	One
ONNX	0.012	0.913	0.019	0.024	0.032
Tensorflow	0	0.948	0.014	0.011	0.027

Table 5. Probability distribution for the "down" audio example using ONNX-based and Keras-based models.

"down" Audio Example	Probability Distribution				
	Yes	No	Down	Up	One
ONNX	0.018	0.020	0.872	0.015	0.075
Keras	0.028	0.013	0.894	0.011	0.054

Table 6. Probability distribution for the "up" audio example using ONNX-based and Keras-based models.

"up" Audio Example	Probability Distribution				
	Yes	No	Down	Up	One
ONNX	0.091	0.044	0	0.747	0.118
Keras	0.131	0.031	0	0.680	0.158

6.4. Oculus Quest VR—Performance Evaluation

Runtime tests on the Oculus viewer were carried out to validate the implemented system. In particular, we perform evaluation on different phases of the implemented flow, from capturing audio signals to the inference on the CNN model to performing SR. For testing the average system execution times, we analysed the following steps on a sample of thirty tests.

Figure 10 depicts the Confusion matrix (gradient plot) of the ONNX converted model.

Figure 11 Case (a) depicts the average execution times of the algorithm implemented for the capture of the audio data stream from the built-in microphone.

The average execution time of the algorithm for audio signal capture is 2.27 s with a standard deviation of 0.0504.

Figure 11 Case (b) shows the tests for the technique of extracting the MFCC coefficients necessary for processing the audio signal acquired by the headset.

The average execution time of the algorithm takes 2.27 s with a standard deviation of 0.0504. The low standard deviation across repeated tests (n = 30) reflects the high temporal consistency of the audio processing pipeline, making further optimisation easier and more predictable. Figure 12 Case (a) reports the execution times for converting input data into a tensor. The tests performed show that the average time for tensor conversion is equal to 0.83 ms, with a standard deviation of 0.8742.

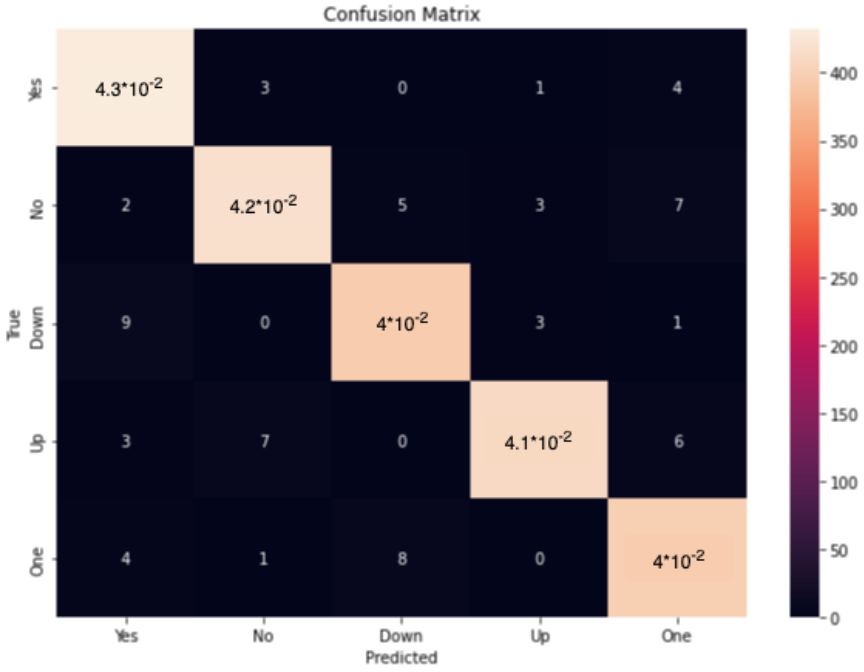


Figure 10. Confusion matrix (gradient plot) of the ONNX converted model.

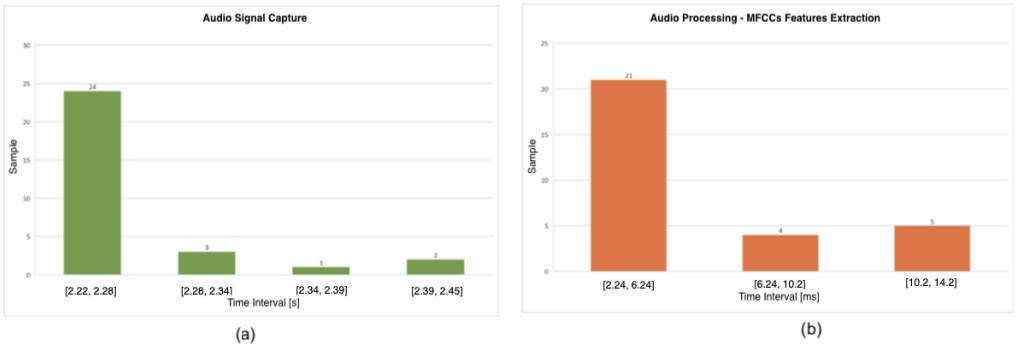


Figure 11. (**a**) Execution time for voice capture. (**b**) Execution time for MFCC feature extraction.

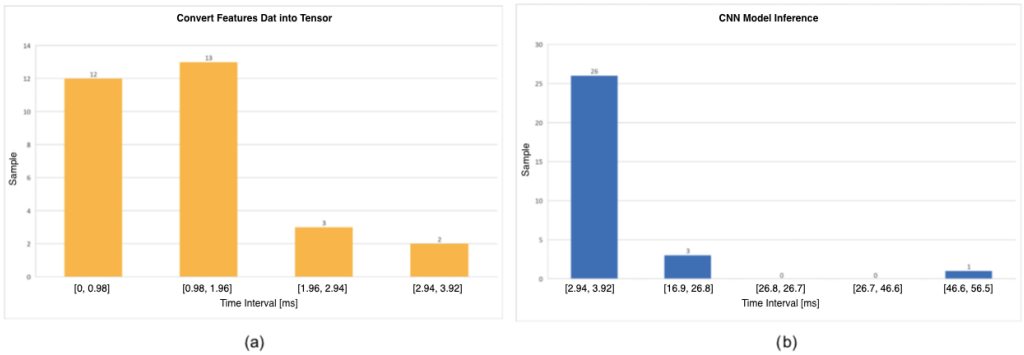


Figure 12. (**a**) Execution time for data-to-tensor conversion. (**b**) Execution time for CNN model inference.

Figure 12 Case (b) depicts the execution time for making inferences on the CNN model. The different tests performed show that this step takes on average 12.7 ms, with a standard deviation of 8.7688. Although variability is higher in the CNN inference phase, the observed standard deviation remains within acceptable limits for real-time applications. A paired *t*-test or non-parametric alternative (e.g., Wilcoxon signed-rank test)

could be adopted in future analysis to confirm the performance consistency across varying input samples.

6.5. Total Execution Time

To validate the implemented virtual reality application and its integration with the AI pipeline, an analysis of the total execution times of all the steps was carried out. Figure 13 shows the total execution times of the integrated system. An average time of 2.43 s is required for the system to work properly. From the experimental results, a standard deviation of 0.2258 is obtained.

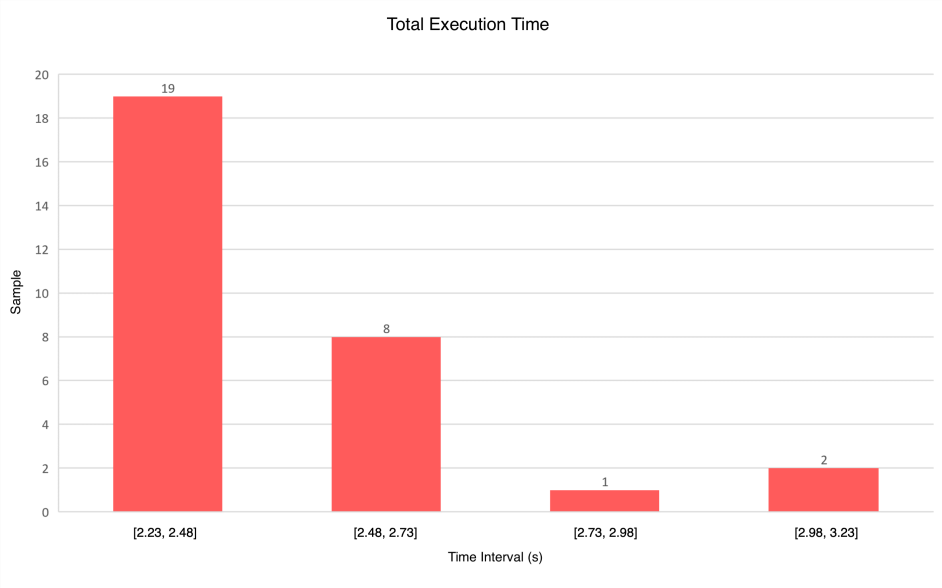


Figure 13. Execution time for the integrated VR application.

With a consistent response time under 2.5 s and relatively low variance, the prototype exhibits sufficient stability for real-world usage. A confidence interval estimation based on these values would support the system's reliability claims, and further testing could explore distribution normality to justify parametric tests.

7. Conclusions and Future Work

The rapid advent of VR technologies has led researchers to identify innovative solutions that make it possible to define the interconnection between the real world and the virtual one. DTs are three-dimensional models which, in addition to their physical characteristics, also reflect their behavioural characteristics within the reconstructed VR environment. To allow users to interconnect with DTs in a VR environment, it is necessary to build ad hoc applications based on ASR systems. In fact, the interaction of the user with the VR application through their voice allows them to enjoy a more realistic and engaging immersive experience.

In this paper, we presented a general methodology to integrate a trained CNN model on which the ASR system is based in a VR application in order to allow users to interact with the VR environment through their simple voice commands. In particular, we discussed how a trained ASR model can be converted to the ONNX format and imported into the Unity real-time development platform so as to create VR applications. Moreover, we discussed how to develop a prototype using the Oculus Quest headset as the Edge device from experiments. The ASR system has shown promising performance when used in the VR application. In fact, the interaction between the ARS system and the VR application

requires a very short time, on the order of a second, making the VR application usable and responsive for the user.

With this work, we contributed to improving the state of the art regarding the development of VR applications controlled through the user's voice commands, especially considering hardware/software-constrained headset devices. In fact, through our methodology, we overcome the limits of some headset devices in deploying a VR application integrated with an ASR system.

In future works, we plan to improve our solution by enabling a mechanism that optimises the ASR system in real-time. The main idea is to store the user's voice commands in the Oculus Quest headset's local storage in order to perform a CNN model optimisation. Accordingly, the CNN model pre-trained with a generic audio dataset will be further trained with the user's voice, adapting the system to the user's specific features and improving recognition accuracy over time. This makes the application progressively more responsive and fine-tuned to individual users. To do this, further evaluation of device-level training strategies and communication protocols will be necessary. This would enable the refinement of the model with user-specific data, though further study is needed to evaluate long-term performance, generalisation, and scalability. Our case study was based on the Unity Real-Time Development Platform. In future developments, we also plan to test our methodology considering the Meta XR Interaction SDK.

Author Contributions: Conceptualization, A.C. (Antonio Celesti), A.F.M.S.S., Y.-S.L., E.F.S. and M.V.; methodology, M.F.; software, A.C. (Alessio Catalfamo); validation, A.C. (Alessio Catalfamo) and A.C. (Antonio Celesti); writing—original draft preparation, A.C. (Alessio Catalfamo) and A.C. (Antonio Celesti); writing—review and editing, A.C. (Alessio Catalfamo) and A.C. (Antonio Celesti); supervision, M.V.; supervision, A.C. (Antonio Celesti); funding acquisition, A.C. (Antonio Celesti) and M.V. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by the Italian Ministry of Health Piano Operativo Salute (POS) trajectory 2 "eHealth, diagnostica avanzata, medical device e mini invasività" through the project "Rete eHealth: AI e strumenti ICT Innovativi orientati alla Diagnostica Digitale (RAIDD)" (CUP J43C22000380001), and the Italian PRIN 2022 project Tele-Rehabilitaiton as a Service (TRaaS) (code PRIN_202294473C_001-CUP J53D23007060006).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author(s).

Acknowledgments: The authors want to express our gratitude to Andrea Siragusa, a student at the University of Messina, for his valuable support in the implementation and field experiments.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Oculus Quest VR Tool. Available online: https://developer.oculus.com/quest/ (accessed on 17 April 2024).
2. Latif, S.; Zaidi, A.; Cuayahuitl, H.; Shamshad, F.; Shoukat, M.; Qadir, J. Transformers in Speech Processing: A Survey. *arXiv* **2023**, arXiv:2303.11607. Available online: http://arxiv.org/abs/2303.11607 (accessed on 22 February 2025). [CrossRef]
3. Lukaj, V.; Catalfamo, A.; Fazio, M.; Celesti, A.; Villari, M. Optimized NLP Models for Digital Twins in Metaverse. In Proceedings of the 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 27–29 June 2023; pp. 1453–1458. [CrossRef]
4. Padmanabhan, J.; Premkumar, M.J.J. Machine learning in automatic speech recognition: A survey. *IETE Tech. Rev.* **2015**, *32*, 240–251. [CrossRef]
5. Evrard, M. Transformers in Automatic Speech Recognition. In *Human-Centered Artificial Intelligence: Advanced Lectures*; Chetouani, M., Dignum, V., Lukowicz, P., Sierra, C., Eds.; Springer International Publishing: Cham, Switzerland, 2023; pp. 123–139. [CrossRef]

6. Yang, J.; Chan, M.; Uribe-Quevedo, A.; Kapralos, B.; Jaimes, N.; Dubrowski, A. Prototyping Virtual Reality Interactions in Medical Simulation Employing Speech Recognition. In Proceedings of the 2020 22nd Symposium on Virtual and Augmented Reality, SVR 2020, Porto de Galinhas, Brazil, 7–10 November 2020; pp. 351–355. [CrossRef]
7. Alimanova, M.; Soltiyeva, A.; Urmanov, M.; Adilkhan, S. Developing an Immersive Virtual Reality Training System to Enrich Social Interaction and Communication Skills for Children with Autism Spectrum Disorder. In Proceedings of the SIST 2022—2022 International Conference on Smart Information Systems and Technologies, Nur-Sultan, Kazakhstan, 28–30 April 2022. [CrossRef]
8. Neundlinger, K.; Mühlegger, M.; Kriglstein, S.; Layer-Wagner, T.; Regal, G. Training Social Skills in Virtual Reality Machine Learning as a Process of Co-Creation. In *Disruptive Technologies in Media, Arts and Design. ICISN 2021. Lecture Notes in Networks and Systems*; Springer: Cham, Switzerland, 2022; Volume 382, pp. 139–156. [CrossRef]
9. Aljabri, A.; Rashwan, D.; Qasem, R.; Fakeeh, R.; Albeladi, R.; Sassi, N. Overcoming Speech Anxiety Using Virtual Reality with Voice and Heart Rate Analysis. In Proceedings of the International Conference on Developments in eSystems Engineering, DeSE, Virtual, 14–17 December 2020; pp. 311–316. [CrossRef]
10. MacUlada, R.E.P.; Caballero, A.R.; Villarin, C.G.; Albina, E.M. FUNologo: An Android-based Mobile Virtual Reality Assisted Learning with Speech Recognition Using Diamond-Square Algorithm for Children with Phonological Dyslexia. In Proceedings of the 2023 8th International Conference on Business and Industrial Research, ICBIR 2023, Bangkok, Thailand, 18–19 May 2023; pp. 403–408. [CrossRef]
11. Doan, L.; Ray, R.; Powelson, C.; Fuentes, G.; Shankman, R.; Genter, S.; Bailey, J. Evaluation of a Virtual Reality Simulation Tool for Studying Bias in Police-Civilian Interactions. In Proceedings of the International Conference on Human-Computer Interaction 2021, Bari, Italy, 30 August–3 September 2023; pp. 388–399. [CrossRef]
12. Krishnamurthy, V.; Rosary, B.J.; Joel, G.O.; Balasubramanian, S.; Kumari, S. Voice command-integrated AR-based E-commerce Application for Automobiles. In Proceedings of the 2023 International Conference on Signal Processing, Computation, Electronics, Power and Telecommunication, IConSCEPT 2023, Karaikal, India, 25–26 May 2023. [CrossRef]
13. Siyaev, A.; Jo, G.S. Neuro-Symbolic Speech Understanding in Aircraft Maintenance Metaverse. *IEEE Access* **2021**, *9*, 154484–154499. [CrossRef]
14. Daneshfar, F.; Jamshidi, M.B. A Pattern Recognition Framework for Signal Processing in Metaverse. In Proceedings of the 2022 8th International Iranian Conference on Signal Processing and Intelligent Systems, ICSPIS 2022, Mazandaran, Iran, 28–29 December 2022. [CrossRef]
15. Li, J.; Feng, Z.; Yang, X. Multi-channel human-computer cooperative interaction algorithm in virtual scene. In Proceedings of the 2020 6th International Conference on Computing and Data Engineering, Sanya, China, 4–6 January 2020; pp. 217–221. [CrossRef]
16. Alasadi, A.; Aldhyani, T.; Deshmukh, R.; Alahmadi, A.; Alshebami, A. Efficient Feature Extraction Algorithms to Develop an Arabic Speech Recognition System. *Eng. Technol. Appl. Sci. Res.* **2020**, *10*, 5547–5553. [CrossRef]
17. Warden, P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv* **2018**, arXiv:1804.03209. [CrossRef]
18. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980. Available online: http://arxiv.org/abs/1412.6980 (accessed on 25 February 2025).
19. Barracuda. Available online: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html (accessed on 12 April 2023).
20. Accord.NET. Available online: http://accord-framework.net/ (accessed on 12 April 2023).
21. SharpDX. Available online: http://sharpdx.org/ (accessed on 12 April 2023).
22. Abdul, Z.K.; Al-Talabani, A.K. Mel Frequency Cepstral Coefficient and its Applications: A Review. *IEEE Access* **2022**, *10*, 122136–122158. [CrossRef]
23. Oculus Developer Hub 2.0. Available online: https://developer.oculus.com/blog/oculus-developer-hub-20/?locale=it_IT (accessed on 12 April 2023).

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