VOXReality

VOICE DRIVEN INTERACTION IN XR SPACESVOICE DRIVEN INTERACTION IN XR SPACES

Model Deployment Analysis V1

WP 4

29/02/2023

Funded by the European Union

Version	1.0
WP	4
Dissemination level	Public
Deliverable lead	SYN
Authors	Stavroula Bourou, Dimitris Kontopoulos, Apostolos Maniatis (SYN), Olga Chatzifoti (MAG), Leesa Joyce (HOLO), Athanasios Ntovas, Georgios Papadopoulos, Stefanos Biliousis, Petros Drakoulis (CERTH), Yusuf Can Semerci (UM)
Reviewers	Manuel Toledo (VRDays); Spiros Borotis (MAG)
Abstract	This document describes the work done in the first 17 months of project regarding the AI models deployment and sharing, once-for-all training, inference optimization and implementation details of XR applications.
Keywords	Model Deployment, Model Sharing, Deployment Guidelines, once-for-all Training, Inference Optimization, Interactive XR Application Development
License	

Dissemination Level	
PU	**Public**
PP	Restricted to other programme participants (Including the Commission Services)
RE	Restricted to a group specified by the consortium (Including the Commission Services)
CO	Confidential, only for members of the consortium (Including the Commission Services)

Nature	
PR	Prototype
RE	**Report**
SP	Specification
TO	Tool
OT	Other

Version History

Version	Date	Owner	Author(s)	Changes to previous version
0.1	2024-01-08	SYN	Stavroula Bourou	ToC released
0.2	2024-02-16	MAG	Olga Chatzifoti	Added in Sec. 4.2
0.3	2024-02-16	SYN	Dimitris Kontopoulos	Added in Sec. 4.1
0.4	2024-02-17	SYN	Stavroula Bourou	Added Sec. 3
0.41	2024-02-19	HOLO	Leesa Joyce	Added Sec. 4.3
0.5	2024-02-20	SYN	Stavroula Bourou, Apostolos Maniatis	Updated Sec. 3
0.6	2024-02-21	SYN	Stavroula Bourou	Executive Summary, Introduction, Conclusion
0.7	2024-02-23	CERTH	Athanasios Ntovas, Georgios Papadopoulos, Stefanos Biliousis, Petros Drakoulis	Added Sec. 2
0.8	2024-02-26	UM	Yusuf Can Semerci	Contribution to Sec 2, 3, 4
0.81	2024-02-26	UM	Yusuf Can Semerci	Controls and formatting
0.9	2024-02-26	SYN	Stavroula Bourou	Final formatting and controls
0.91	2024-02-28	VRD	Manuel Toledo	Internal review complete
0.92	2024-02-28	MAG	Spiros Borotis	Internal review complete
1.0	2024-02-29	SYN	Stavroula Bourou	Revisions complete

Table of Contents

List of Abbreviations & Acronyms

ADB	:	Android Debug Bridge
AI	:	Artificial Intelligence
AMD	:	Advanced Micro Devices
API	:	Application Programming Interface
ASR	:	Automatic Speech Recognition
BERT	:	Bidirectional Encoder Representations
BLEU	:	Bilingual Evaluation Understudy
CAD	:	Computer-Aided-Design
CI/CD	:	Continuous Integration/ Continuous Deployment
CLI	:	Command Line Tool
CNN	:	Convolutional Neural Network
Co2	:	Carbon dioxide
CPU	:	Central Processing Unit
CUDA	:	Compute Unified Device Architecture
CV	:	Computer Vision
DA	:	Dialogue Agent
DE	:	German (language)
DNN	:	Deep Neural Network
NL	:	Dutch (language)
ECS	:	Entity-Component-System
EN	:	English (language)
ES	:	Spanish (language)
FLOPS	:	Floating Point Operations Per Second
FOV	:	Field of View
FP	:	Floating Point
GPT2	:	Generative Pretrained Transformer 2
GPU	:	Graphics processing unit
EL	:	Greek (language)
IT	:	Italic (language)
ML	:	Machine Learning
MLP	:	Multilayer Perceptron
NLG	:	Natural Language Generation
NLP	:	Natural Language Processing
NLU	:	Natural Language Understanding
NMT	:	Neural Machine Translation
NN	:	Neural Network
OBS	:	Open Broadcaster Software
OFA	:	Once-For-All
ONNX	:	Open Neural Network Exchange
OS	:	Operating System
QAT	:	Quantization-Aware Training
RAM	:	Random-Access Memory
REST	:	Representational State Transfer
RNN	:	Recurrent Neural Networks

ROUGE	:	Recall-Oriented Understudy for Gisting Evaluation
SAST	:	Static application security testing
SDK	:	Software Development Kit
SIMD	:	Single Instruction, Multiple Data
SOTA	:	State-of-Art
UI	:	User Interface
UVC	:	Unified Visual Transformers Compression
VFX	:	visual effects
ViT	:	Vision Transformers
VL	:	Vision-Language
VR	:	Virtual Reality
WCAG	:	Web Content Accessibility Guidelines
WebGL	:	Web Graphics Library
webRTC	:	Web Real-Time Communications
WP	:	Work Package
XR	:	eXtended Reality

List of Figures

List of Tables

Executive Summary

The purpose of this document is to present an in-depth analysis and methodology for the training, optimization, deployment and sharing of VOXReality AI models, as well as to describe the implementation details of the VOXReality eXtended Reality (XR) applications. Specifically, the document provides an overview of the state-of-the-art “once-for-all” (OFA) concept, as well as various AI model optimization techniques. Additionally, the VOXReality model optimization approach is introduced to decrease the model size and computational requirements while maintaining performance, ensuring efficient and effective use during the deployment. This tool also facilitates the export of AI models into common formats like ONNX. Experimental results demonstrating the effectiveness of the VOXReality optimization tool are also presented. Furthermore, the document defines the deployment and sharing options for the pretrained VOXReality AI models, providing clear guidelines on effective deployment and access, including source code-based deployment and containerization strategies. It also includes the architecture, design principles, and implementation details of VOXReality's XR applications, specifically the VR Conferences, Augmented Theatre, and Training Assistant.

1 Introduction

The rapid growth of Deep Neural Networks (DNN) has led to architectures with hundreds of millions of parameters, demonstrating significant challenges in training and inference processes. Those challenges are intensifying when the models are deployed in resource-constrained devices, like edge devices or mobile phones. Specifically, the training phase of these AI models demands high computational resources, being particularly time-consuming and power intensive. In addition, the inference of these models requires significant computing power and can be time-consuming, particularly on devices with limited processing capabilities, making inference optimization essential for fast and efficient performance. Transformer models fall into this category of computationally intensive architectures, thus requiring a focus on training and inference optimization to ensure their efficient and effective use during the deployment.

VOXReality implements advanced Natural Language Processing (NLP) models based on transformer architecture. Therefore, it is crucial to overcome these computational challenges through innovative strategies, ensuring that these AI models are not only powerful but also practical for deployment in various hardware environments, including those with limited resources. In VOXReality, we address those challenges by exploring the “once-for-all” (OFA) training concept, allowing for the efficient creation of sub-networks tailored to specific hardware and use cases. Additionally, we implement an optimization tool that employs different techniques like pruning and quantization. This tool also facilitates the export of AI models into common formats like ONNX, enhancing their adaptability.

Ensuring that AI models are effectively utilized in real-world scenarios is a critical aspect. VOXReality addresses this by offering a variety of deployment options to facilitate the easy and efficient use of AI models across different platforms, from edge devices to cloud servers. The adaptability of these models is further enhanced by VOXReality's comprehensive deployment guidelines, which assist integration of AI technologies across various applications. Furthermore, the VOXReality AI models are shared on the Hugging Face platform, to promote wider adoption and to invite external developers to expand and refine these models, thus fostering innovation and broadening the scope of their AI solutions.

The VOXReality AI models have been deployed in three distinct use cases: VR Conference, Augmented Theatre, and Training Assistant, showcasing the versatility of these models in XR environments and their ability to create immersive experiences. The feasibility of implementing these applications will be verified through two rounds of pilot testing under real-world conditions, demonstrating their practical application and robustness. The detailed design and implementation of these AI models have been meticulously planned to align with both the user requirements and technical specifications, guaranteeing that the end solutions are not only innovative but also practical and user centric.

The technical work described in this document is performed in all three tasks of (T4.1, T4.2, T4.3) of WP4 until the end of the 17th month of VOXReality project. Specifically, Task 4.1 “Model deployment and serving” focuses on the deployment and sharing options of pretrained VOXReality AI models as well as on the deployment of those models in every use case scenario. Task 4.2 “Model training & inference optimization” investigates the SOTA methods for economic model training following the “once-for-all” training approach as well as the

different optimization techniques. Moreover, in this task, tools for “once-for-all” training and optimization are implemented by VOXReality consortium. Task 4.3 “Novel Interactive XR Applications” is responsible for developing the XR applications utilizing the VOXReality AI models in the use cases.

1.1 Intended Audience

The intended audience for this deliverable includes the VOXReality project consortium and third-party users, consisting of participants in the project's open calls as well as researchers and AI & XR professionals interested in exploring VOXReality research outputs. The document provides the VOXReality optimization approach that could be utilized by the AI engineers to create cost-effective and energy-efficient AI models, making them suitable for a wide range of deployment scenarios. Moreover, the provided deployment guidelines can also be employed by AI engineers and developers to correctly deploy the AI models across various applications. In addition, the implementation details of VOXReality XR applications can provide useful information on how technologies can be integrated into various sectors, offering practical insights for developers looking to adapt these applications to specialized use cases or environments.

1.2 Relations to the other activities

The VOXReality optimization tool, the deployment guidelines and the model sharing are strongly dependent on the VOXReality AI models. Therefore, this document is intrinsically connected with all the tasks of WP3 “Advanced AI muti-model for XR”. Additionally, the model deployment is closely correlated with the Task 2.3 “Development Infrastructure”. It should be mentioned that many decisions regarding the implementation of XR applications are made based on User Requirements and Technical Requirements, extracted from Task 2.1 "User Requirements" and Task 2.2 "Technical Requirements" respectively as well as by the pilot planning outlined in Task 5.1 “Planning and Validation”. Finally, this document is linked with the WP7 “Integration paths” since it offers useful insights into how third-party users from Open Calls can utilize the research outputs.

1.3 Document Structure

Section 1 provides an introduction of the deliverable’s intended audience as well as an overview of its content. *Section 2* introduces the SOTA algorithms for the “once-for-all” training approach as well the AI model optimization techniques, providing a detailed background overview of those topics. Moreover, this section describes the proposed VOXReality two-stages optimization pipeline that can be applied on the VOXReality AI models. In addition, it discusses the experimental results obtained from the application of this method. *Section 3* details the various deployment methods available for utilizing the AI models, along with comprehensive deployment guidelines. It also describes the process of model sharing through the Hugging Face platform. *Section 4* includes all the detailed information about VOXReality XR applications, covering the design and the implementation aspects, such as the development environment, creation of 3D models and scenes, development and integration of core algorithms, the various User Interface (UI) elements, etc. *Section 5* outlines the conclusions and discusses future work.

2 Model Training and Inference Optimization

In recent years, Deep Neural Networks (DNN) architectures have become extensively large with hundreds of millions of parameters. As a consequence, Neural Network (NN) training and inference phases have become increasingly challenging procedures. Many optimization techniques have been developed over the years to overcome this, while the growing demand for deploying neural networks on resource-constrained devices, such as mobile phones and edge devices, has further underscored the importance of developing efficient optimization techniques.

Neural Networks, in general, require high computational resources due to their complexity which typically involves a large number of linear algebra operations with high precision floating-point variables. The training phase in particular is notoriously time-consuming, often demanding several days to converge to an acceptable solution. Even the inference of these models can consume significant power, presenting challenges for deployment in energy-constrained environments, such as edge devices, which have significantly less computational power compared to personal computers and servers. Adding to that, mobile devices operate mostly on batteries, urging developers to optimise the applications to minimize power consumption and extend the battery life. Furthermore, for applications demanding real-time responses, such as gaming or communication apps, minimising the execution time becomes essential to ensure optimal user experience. Based on all the aforementioned, we are led to the conclusion that neural networks need to be run on a variety of heterogeneous hardware, each one with different, sometimes unpredictable, constraints.

In VOXReality, we are planning to provide the VOXReality platform with the means to conduct economical model training and deployment, by exploring a possible adaptation of the *"once-for-all" (OFA)* [1] training concept to the project's models, enabling the extraction of sub-networks from a single large one, that are conditioned on the targeted hardware and fine-tuned on the specific use-case. The VOXReality source models are developed under WP3 and focus on automatic speech recognition, multilingual translation, vision-language and context-aware conversational agents. In addition to the introduction of the OFA adaptation, we will also develop and release a Command Line Tool (CLI) that will allow users to further optimise networks produced by the project in a more "traditional" manner, by employing a series of pruning, quantization, graph optimization and compression operations, additionally exporting these models into various commonly used formats like ONNX and TorchServe.

All the models developed for the three use cases (Augmented Theatre, Virtual Conference and Training assistant) are Transformer-based [2]. Transformers are computationally, storage and memory expensive compared to Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for a few reasons. Namely, transformers use a mechanism called "Attention", which involves the investigation of the relationships between all input tokens. This can be quite demanding computationally, especially as the input sequence gets longer. Transformers typically hold a large number of parameters that require extensive memory for storage and processing. While transformers are great at understanding long-distance connections in data, this comes at the cost of having to access information from all parts of the input sequence, which further adds to their computational and memory requirements. Overall, while transformers are very powerful for specific tasks, they come with their own set of constraints

and disadvantages compared to CNNs and RNNs, deriving mainly from their higher computational and memory needs.

VOXReality's AI models are intended to be deployable in various environments: a) on edge devices (e.g. HDM or mobile phone - may require heavy quantization and compression), b) on small computers (e.g. PC or laptop - may require optimizations for SIMD CPU), c) on cloud servers (e.g. Microsoft Azure virtual machine - may require CUDA GPU optimizations) and d) on controlled specialised infrastructure may require the use of framework-specific packages (e.g. a specific game-engine).

In Figure 1, we can see the intended life cycle of a VOXReality model. It begins as a large-scale generic source model trained with the OFA principal that is afterwards fine-tuned on additional data, hardware-optimised and exported into a different format, targeted for a specific use-case. Core of this scheme is the possibility to adapt and utilise the same generic source model in a multitude of downstream tasks.

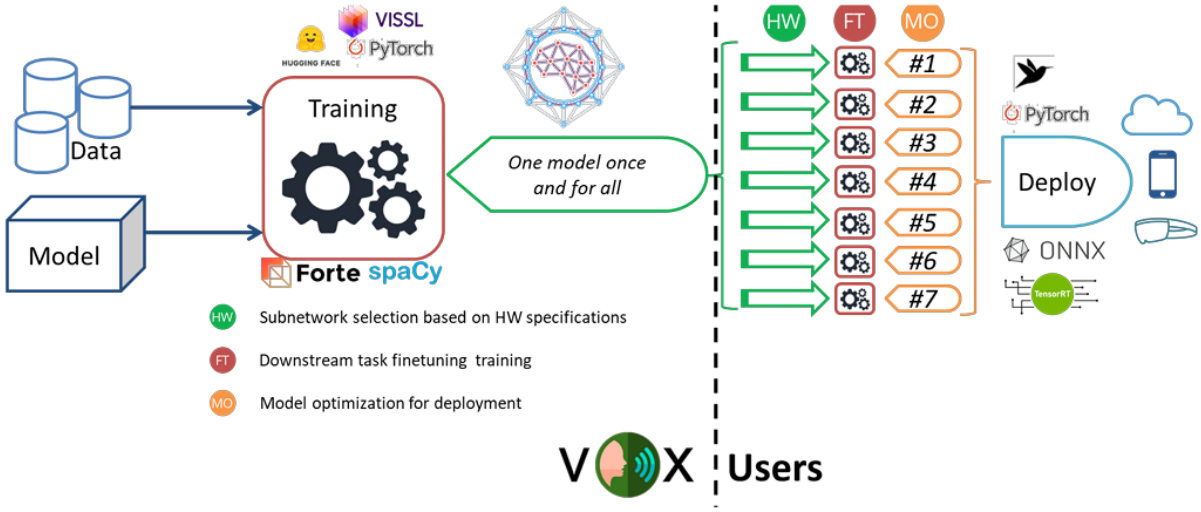

Figure 1:The intended life cycle of a VOXReality model.

2.1 Overview of State-of-the-Art Methods

There is a variety of transformers' optimization techniques and frameworks currently under development, signalling further the perceived importance of the field. One such framework is developed under *A fast Post-Training Pruning Framework for Transformers* [3] which eliminates the need for retraining, aiming to reduce model size and inference latency. The framework operates by taking a pre-trained Transformer model, a sample dataset and a FLOPs/latency constraint as input (Figure 2) and outputs a pruned model that meets the specified resource constraints. By employing a three-stage decomposition process involving a Fisher-based mask search algorithm, mask rearrangement and mask tuning, the framework identifies and prunes redundant components while preserving model accuracy. This retraining-free approach enables quick and efficient model compression, leading to significant reductions in FLOPs and inference latency, without compromising performance, making it a practical solution for optimising Transformer models.

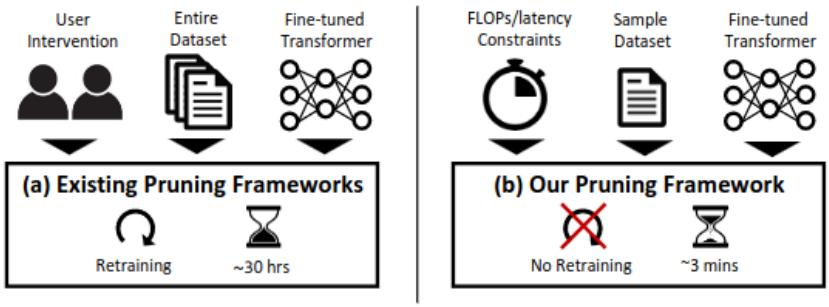

Figure 2: Conventional pruning framework VS Proposed pruning framework [3].

Another approach is the *Unified Visual Transformers Compression* [4] which presents a unified framework for compressing Vision Transformers, combining pruning, layer skipping, and knowledge distillation (Figure 3) techniques to optimise model performance under computational constraints. Pruning selectively removes redundant weights, layer skipping adjusts computation patterns across blocks and knowledge distillation transfers essential information from a larger model to a compressed one. By jointly optimising model weights, pruning ratios, and skip configurations under specific constraints, Unified Visual Transformers Compression (UVC) achieves efficient model compression while maintaining performance on vision tasks.

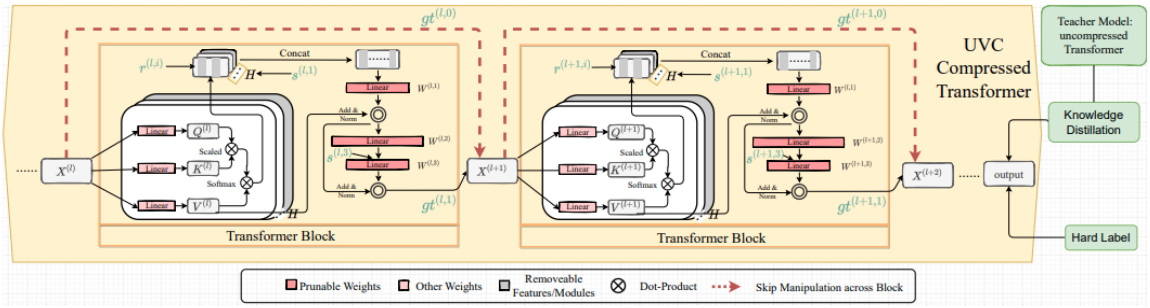

Figure 3: The compression scheme of Unified Visual Transformers Compression.

An example of extreme compression is *XTC: Extreme Compression for Pre-trained Transformers Made Simple and Efficient* [5] which focuses on ultra-low bit precision quantization to compress large pre-trained transformer models for efficient deployment on resource-constrained devices. By combining lightweight layer reduction methods, 1-bit quantization with deep knowledge distillation and data augmentation, longer training budgets, and careful hyperparameter tuning, XTC achieves state-of-the-art results in extreme compression, surpassing previous methods in both compression ratio and model performance. The study systematically evaluates the impact of key hyperparameters and training strategies on extreme compression, highlighting the importance of simplicity and efficiency in the compression pipeline. Overall, the research presents a simple yet effective method for extreme compression of pre-trained transformers, demonstrating superior performance and compression rates compared to existing approaches.

Another work is *MiniVIT: Compressing Vision Transformers with Weight Multiplexing* [6] that introduces a novel compression framework, MiniViT, for Vision Transformers. MiniViT combines weight sharing, transformation, and distillation techniques (Figure 4) to reduce the

number of parameters in Vision Transformer models while maintaining or even improving performance compared to the original models. By multiplexing weights of consecutive transformer blocks and applying weight distillation over self-attention, MiniViT effectively reduces model size without significant loss in accuracy. The framework demonstrates its efficacy through experiments showing substantial parameter reduction in pre-trained models like Swin-B and DeiT-B, with performance improvements on tasks such as ImageNet classification.

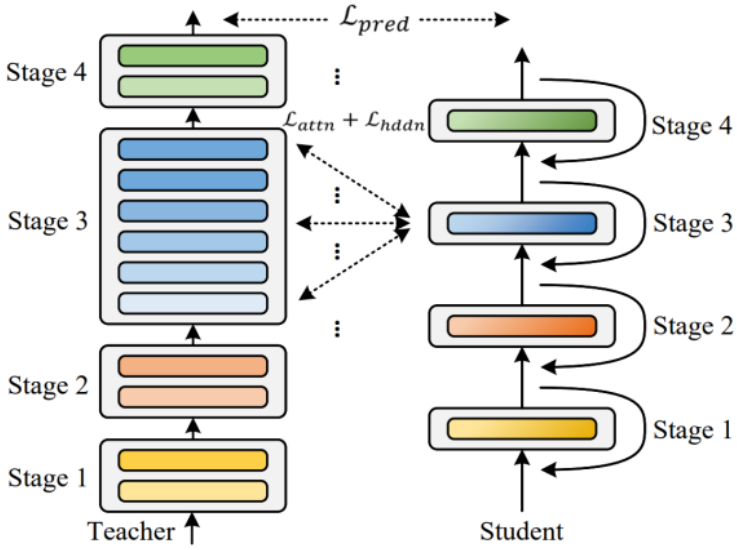

Figure 4: The compression scheme of MiniViT.

Lastly, *Compressing Large-Scale Transformer-Based Models: A Case Study on BERT* [7] *focuses* on compressing large-scale Transformer-based models, specifically BERT, to make them more suitable for low-capability devices and applications with strict latency requirements. By exploring various compression techniques such as quantization, pruning, and knowledge distillation, the document aims to help researchers and practitioners create lightweight yet accurate models for Natural Language Processing tasks. The insights provided clarify how these compression methods impact model size, performance, and efficiency, offering valuable guidance for optimising Transformer models for real-world applications.

2.1.1 Once-for-All Concept

The Once-for-All (OFA) network training technique introduces a novel approach to deploy neural networks across various devices with different resource constraints efficiently. Unlike traditional methods that require training specialised networks for each scenario, OFA decouples the training and sub-network search processes, enabling the quick selection of specialised sub-networks without additional training (Figure 5). This innovative methodology not only improves accuracy and efficiency on a wide variety of devices but also reduces computational costs and CO2 emissions significantly. By leveraging the OFA training scheme, users can achieve high performance while minimising the time and efficiently use the resources needed for model optimization and deployment.

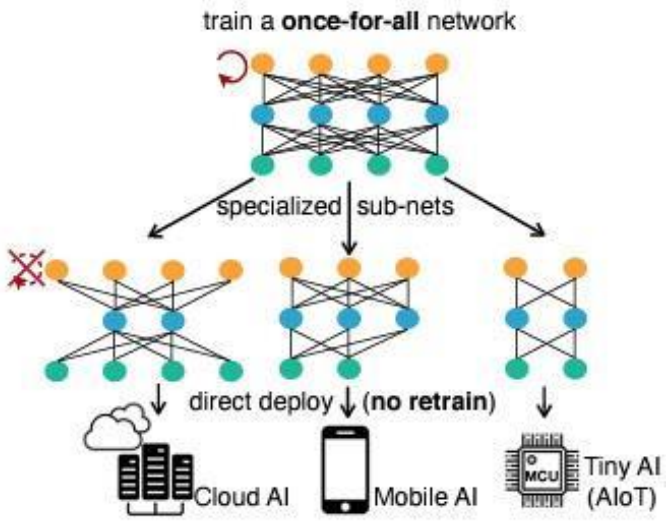

Figure 5: Train the network once and quickly extracts the appropriate sub-network for each different hardware setup.

Initially, the idea of OFA was conceived for Convolutional Neural Networks [8]. The process, in its core, is highly non-trivial since it requires the joint optimization of weights belonging to potentially different sub-networks, in such a structured way to maintain the accuracy of all of them simultaneously. It is computationally prohibitive to enumerate all sub-networks to get the exact gradient in each update step, while randomly sampling a few sub-networks in each step can lead to significant error drops. The challenge is that the different sub-networks are implicitly interfering with each other, making the training process and convergence of the whole once-for-all network complicated, at the very least.

The main idea of OFA in CNNs is the progressive shrinking algorithm (Figure 6). Progressive shrinking works by enforcing training order from large sub-networks to small sub-networks in a progressive manner within the Once-for-All (OFA) network. This training scheme aims to prevent interference between sub-networks by starting with training the largest neural network with maximum dimensions (e.g., kernel size, depth, width) and then progressively fine-tuning the network to support smaller sub-networks that share weights with the larger ones. By following this approach, progressive shrinking provides better weight initialization by selecting crucial weights from larger sub-networks and allows for the distillation of smaller sub-networks, thereby enhancing the training efficiency of the OFA network.

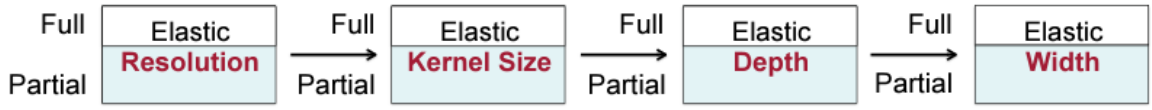

Figure 6: Illustration of the progressive shrinking process for CCNs to support different depth, width, kernel size, and resolution.

After training the Once-for-All (OFA) network, the process of selecting a specific sub-network for a particular hardware device involves utilising a predictor-guided architecture search. This search method leverages accuracy and latency predictors trained on a subset of sub-networks to guide the selection of an architecture that meets the requirements of the target hardware device. By using the predictors to estimate the performance of different sub-networks in terms

of accuracy and latency, the architecture search can efficiently identify the most suitable sub-network that balances accuracy and computational efficiency for the given hardware constraints. This approach enables the OFA network to be specialised for diverse hardware devices by selecting the optimal sub-network that best aligns with the device's capabilities and operational needs. Bringing this concept closer to VOXReality needs, essentially equates to decomposing and adapting the aforementioned processes to transformers.

A work that explores similar mechanisms for transformers is *DynaBERT* [9] which adapts to diverse architectural configurations by dynamically adjusting depth (network layers) and width (the dimensionality of the hidden layers) dimensions. Initially, it trains a width-adaptive BERT model able to flexibly adjust its width to suit specific tasks and hardware limitations. Subsequently, DynaBERT extends this adaptability to encompass both width and depth dimensions, allowing for fine-grained optimization of model size and latency. Through sophisticated techniques such as knowledge distillation and network rewiring, DynaBERT distills crucial insights from the full-sized model into smaller sub-networks while preserving essential features. The aforementioned scheme empowers superior performance across various efficiency constraints, positioning it as a versatile and potent solution for deploying efficient language models in real-world applications. Figure 7 shows the two-stage procedure to train with DynaBERT. First, it uses knowledge distillation to transfer knowledge from a frozen teacher network to a student sub-network with adaptive width (DynaBERTw). Then, using knowledge distillation it transfers knowledge from DynaBERTw to multiple student sub-networks with adaptive both width and depth (DynaBERT).

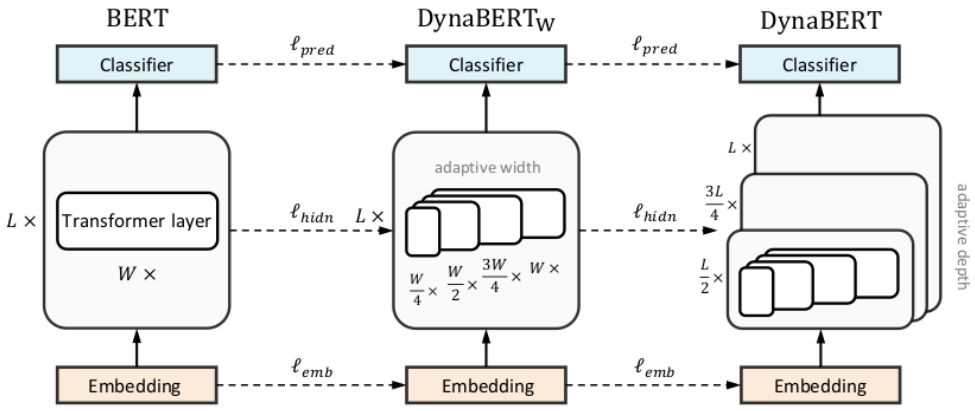

Figure 7: The two-stage procedure to train with DynaBert.

A different work is *Prune Once For All* [[10] (Figure 8). By leveraging weight pruning and model distillation techniques, the Prune OFA method inherently creates sparse pre-trained models with specific sparsity patterns. These sparse models can be considered as sub-networks of the original dense model, where certain connections or parameters have been pruned based on the defined sparsity ratio. These sub-networks retain the essential information required for efficient processing and can be utilised for inference tasks, reducing computational costs and memory requirements while maintaining high performance. The ability to extract sub-networks from the Prune OFA method adds flexibility and scalability to the deployment of sparse pre-trained language models in various applications.

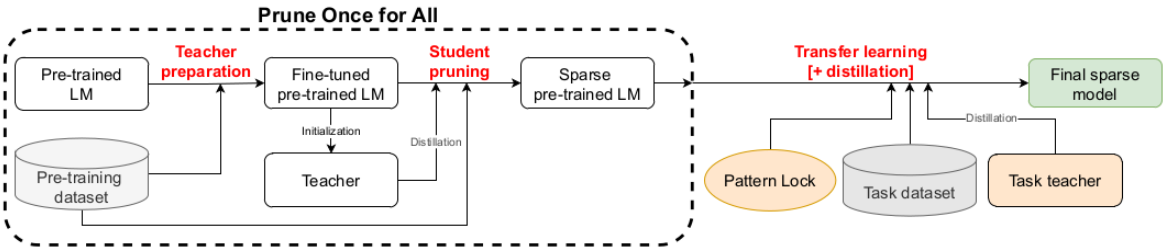

Figure 8: The Prune OFA training scheme.

Finally, the work we deem having the greatest potential for use in the VOXReality project is Autoformer [11], since it showcases the largest number of changeable/tunable parameters and is shown to work for vision transformers (ViT). The key changeable parameters in AutoFormer include depth, K-Q-V dimensions, embedding dimension, attention's number of heads and Multilayer Perceptron (MLP) ratio, which significantly impact model performance (Figure 9). By allowing all these parameters to be adjusted during training, AutoFormer enables the exploration of vastly diverse transformer structures, adapted to specific requirements.

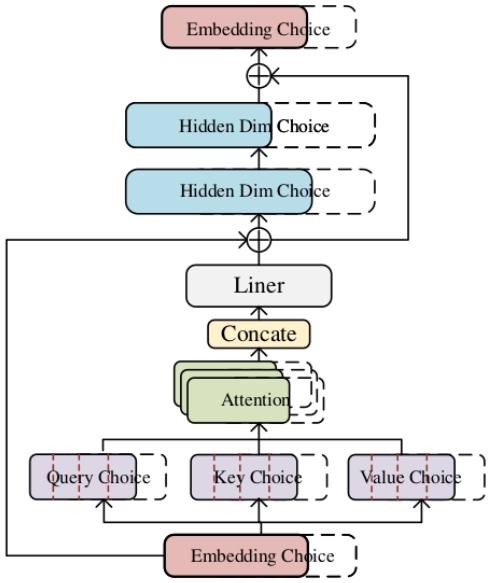

Figure 9: Detailed transformer block in an AutoFormer structure with all changeable parameters.

The training procedure of AutoFormer involves creating a super-network with adjustable changeable dimensions (Table 1) and for each epoch selecting a subset of this network for training. During training, only the weights specific to the chosen subset are updated while the remaining weights stay constant, a practice defined as "weight entanglement" (Figure 10). This process is repeated for each epoch until training is complete. By following this iterative approach, the sub-networks within the super-network are effectively trained, enabling efficient weight inheritance and the development of high-performance transformer models.

Table 1: The values of the changeable dimensions, where tuples in parentheses represent the lowest value, the highest value, and step of each tuneable parameter, respectively.

	Supernet-tiny	Supernet-small	Supernet-base
Embed Dim	(192, 240, 24)	(320, 448, 64)	(528, 624, 48)
Q-K-V Dim	(192, 256, 64)	(320, 448, 64)	(512, 640, 64)
MLP Ratio	(3.5, 4, 0.5)	(3, 4, 0.5)	(3, 4, 0.5)
Head Num	(3, 4, 1)	(5, 7, 1)	(8, 10, 1)
Depth Num	(12, 14, 1)	(12, 14, 1)	(14, 16, 1)
Params Range	4 – 9M	14 – 34M	42 – 75M

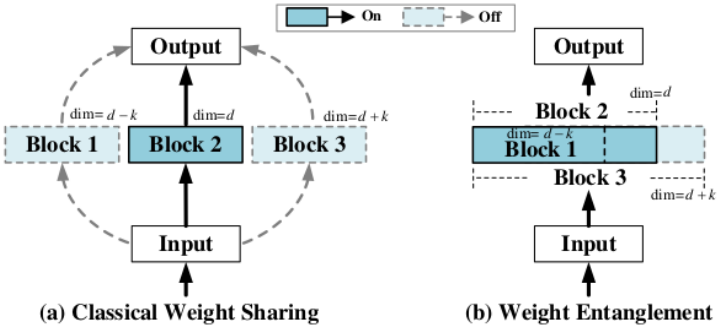

Figure 10: Classical weight sharing (left) vs. Weight entanglement of AutoFormer (right).

After training the super-network, an evolutionary search process is employed to select a specific sub-network with the desired number of parameters. This involves evaluating the performance of different sub-network architectures within the trained super-network and choosing the model that achieves the highest accuracy while meeting the specified parameters constraints. By conducting evolutionary optimization utilising typical selection, crossover and mutation operations, the search algorithm iteratively refines the selection of sub-networks based on their performance and parameter sizes. Through this iterative process, an optimal enough sub-network with the desired number of parameters and accuracy arises, ensuring a balance between model complexity and performance.

2.1.2 Standard AI Model Optimization Techniques

Traditional optimization techniques, such as quantization, pruning, graph optimization and entropy compression are long employed to enhance the efficiency and performance of deep learning models. At a glance, quantization refers to the reduction of the variables' and operations' arithmetic precision, reducing the memory requirements and computational complexity of the network. Pruning and graph optimization remove unnecessary connections and parameters from the model, and entropy compression compresses the remaining elements of the network achieving the smallest possible lossless representation of the model. Subsequently, we will delve into the various optimization components.

Quantization

Quantization can be applied at different stages of the model development process, such as:

Post-training quantization: This is the simplest and most widely used method, where quantization is applied after the model has been trained with floating-point data. This method does not require any re-training or fine-tuning, but it may introduce some accuracy loss due to the reduced precision. Post-training quantization can be further divided into static and dynamic quantization, depending on whether the activations are quantized during inference or not.

Quantization-aware training (QAT): This is a more advanced method where quantization is simulated during the training process, and the model parameters are pre-emptively adjusted in a way to minimize the post-training quantization error. This method can preserve the accuracy of the original model, but it requires more computational resources and time due to provisions. QAT can also be further divided into fake and real quantization, depending on whether the quantization is actually performed during training or not.

Hybrid quantization: This is a hybrid method, where quantization is applied only to some parts of the model during training, such as the weights or the gradients. This method can reduce the memory and computational costs of training while maintaining good accuracy.

Pruning

Pruning is a technique that reduces the size and complexity of neural networks by removing some of their components, such as weights, neurons or layers. Pruning can help improve the efficiency and speed of Transformer models, which are widely used for natural language processing and other tasks. There are different types of pruning techniques for Transformers, such as:

Unstructured pruning: This technique removes individual weights from the model based on some criteria, such as magnitude or importance. Unstructured pruning can achieve high compression rates, but it requires sparse matrix operations which are not well supported by most hardware.

Structured pruning: This technique removes groups of weights that have a regular structure, such as attention heads, filters or layers. Structured pruning can preserve the original matrix operations, which are more efficient and compatible with most hardware.

Graph Optimization

Graph optimization involves refining structurally the network's architecture to enhance efficiency without sacrificing performance. Think of it as reorganising a cluttered workspace to improve productivity. By identifying and streamlining redundant pathways and operations, the optimization process reduces computational overhead and memory usage. This results in faster processing times and more efficient resource utilisation. Ultimately, graph optimization ensures that the neural network operates more smoothly and effectively, akin to a well-organised workspace facilitating better workflow and in most cases is platform dependent (e.g. ONNX).

Entropy compression

Entropy compression refers to the widely and generically used entropy-coding based lossless compression techniques found in various zip/tar-like products. It can always be used as the last stage packing for storing and transferring data, and thus can be implemented to reduce the non-working (offline storage) memory of a model. It is implicitly already present in various common model representation formats like PyTorch's "pt" and "pth".

2.2 VOXReality Model Optimization Approach

Our plan is to create a two-stage optimization pipeline that can be applied on the VOXReality models either independently or in succession, resulting in ever greater efficiency gains. The second stage will be applicable to the majority of the VOXReality provided models (namely Optimization Methodology I) while the first stage of our pipeline (namely Optimization Methodology II) will be applicable only to some of the Vision-Language models to be provided, since it requires them to be built and trained in a special way.

2.2.1 Optimization Methodology I

For the second stage of our pipeline that will be applicable to the majority of the VOXReality models, we deem to arrange sequentially all the standard optimization elements described in section 2.1.2. As an additional step, we chose to utilise and export the widely used ONNX framework for cross-platform model deployment. Subsequently, we present the specific order of the operations (Figure 11) and analyse them in detail.

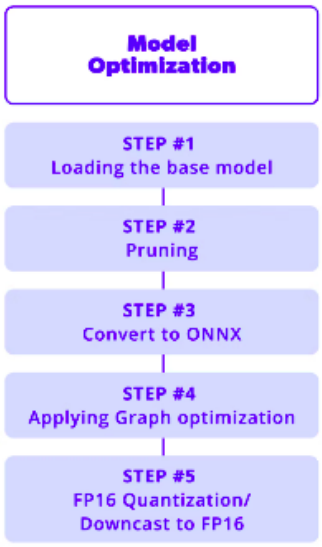

Figure 11: The proposed model optimization pipeline.

Steps #1 and #6, namely "model loading" and "exporting" respectively, are omitted from the following analysis for being trivial.

Step #2 Pruning

We are currently working on applying some form of unstructured post-training pruning in the pipeline (to be presented under the D4.2), similar to the one described in [3].

Step #3 ONNX Conversion

ONNX (Open Neural Network Exchange) serves as an open standard for representing machine learning models through a unified set of operators that define various model architectures. Major frameworks like PyTorch, TensorFlow, etc. support exporting and importing models in the ONNX format, facilitating seamless model migration across different platforms. We deem to use ONNX format as an intermediate representation inside our pipeline for its convenience and compatibility with various plug-and-play solutions for graph optimization, quantization and exporting. To convert our models to ONNX format, we utilise the Optimum library.

Step #4 Graph Optimization

The process of converting our model to ONNX introduces a range of new tools for further optimization. Optimising the ONNX graph involves various underlying techniques, with the most prominent being the Node Fusion. Node Fusion entails consolidating multiple nodes within the graph into a single node wherever feasible, thereby reducing the overall complexity of the graph.

Step#5Quantization

As we previously mentioned, downcasting involves transitioning the precision of its weights, typically from 32-bit (FP32) to 16-bit floating point (FP16) or alternatively to 8-bit, or 4-bit, integer representation formats. This adjustment yields several effects on the model itself. Firstly, by storing numbers in a format that requires fewer bits, the model naturally decreases in size. Since the model was initially trained with 32-bit precision, the shifting of operations to fewer bits can have detrimental effects on the model's accuracy. Nevertheless, in our experiments we showcased that if it is done carefully, the negative effects quantization bears can be contained to a minimal. The results of our experiments, regarding both inference time and inference memory consumption are presented in Chapter 4.

At the end of the pipeline, the optimised model is being converted, if necessary, and exported into the preferred representation format, either be ONNX, the standard PyTorch or the TorchServe format.

2.2.1.1 Experimental Results

In the following paragraph, we will present the results of a preliminary study regarding the application of Optimization Methodology I on a pre-trained version of the VL ViT-GPT2 model for image captioning. The testbed comprises a Windows PC powered by an Intel i7 CPU with 32GB of RAM and an Nvidia GeForce RTX 3060 GPU with 12GB of VRAM. The study was conducted on a 1000-image subset of the COCO 2014 evaluation dataset and regards two axes: inference time and prediction quality under five widely used evaluation metrics; BLEU, ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-Lsum. All those metrics are detailed described in D3.1 "Advanced AI multi-model for XR analysis" [12].

The initial reference model to be passed through the optimization pipeline is in native PyTorch format, with its parameters represented in 32-bit Floating Point (FP) precision. The other representations tested were PyTorch in 16-bit FP, 8-bit integer and 4-bit integer, and after conversion to ONNX format in 32-bit FP with and without graph optimization applied. Note that, not all these representations work for both CPU and GPU, so we split the study in two parts, evaluating separately only the applicable ones in each scenario. Table 2 presents the

inference time and evaluation scores of the model running on GPU while the Table 3 illustrates those results when the model running on CPU.

As it was expected, the model inference processing time and prediction quality seem correlated, as a general trend, with the details presented below. Nevertheless, at least for our tested data, post-training optimization with the exception of the extreme 4-bit quantization scheme does not seem to significantly hurt the quality of the model's predictions, in some cases even enhancing the model's generalisation ability. In the immediate future, we plan to conduct a similar study on an NLP multi-language translation model, extending our knowledge on the pipeline's total behaviour and the individual representations' nuances.

Table 2: Inference time and evaluation scores of ViT-GPT2 captioning model running on GPU.

Model (Arithmetic precision)	Inference Time	BLEU	Rouge-1	Rouge-2	Rouge-L	Rouge-LSUM
PyTorch (FP32)	168 ms	0.048	0.298	0.098	0.273	0.273
PyTorch (FP16)	127 ms	0.044	0.304	0.098	0.279	0.279
ONNX (FP32)	108 ms	0.048	0.298	0.098	0.273	0.273
ONNX Graph Optimization (FP16)	95 ms	0.045	0.293	0.084	0.266	0.266

Table 3: Inference time and evaluation scores of ViT-GPT2 captioning model running on CPU.

Model (Arithmetic precision)	Inference Time	BLEU	Rouge-1	Rouge-2	Rouge-L	Rouge-LSUM
PyTorch (FP32)	539 ms	0.045	0.308	0.095	0.280	0.280
PyTorch (INT8)	318 ms	0.044	0.303	0.092	0.278	0.280
PyTorch (INT4)	173 ms	0.022	0.294	0.083	0.264	0.264
ONNX (FP32)	384 ms	0.048	0.298	0.098	0.273	0.273
ONNX Graph Optimization (FP32)	311 ms	0.048	0.298	0.098	0.273	0.273

2.2.2 Optimization Methodology II

For the first stage of our envisioned optimization pipeline that will be applied only on some of the Vision-Language models to be provided, we plan to develop our own variant of AutoFormer and then extend it novelly to the decoder part of the architecture. The primary challenge of this approach thus, is the distinction between training a ViT-GPT2 (our intention) and a ViT-only (reference) architecture. Unlike ViT (Figure 12a) which comprises encoder-only blocks, the ViT-GPT2 combination (Figure 12b) features both encoder and decoder blocks with the encoder inherited from the ViT and the decoder from the GPT-2 architectures. The reference work of AutoFormer is dealing only with the vision transformer encoder case, and thus novel work is needed to go forward.

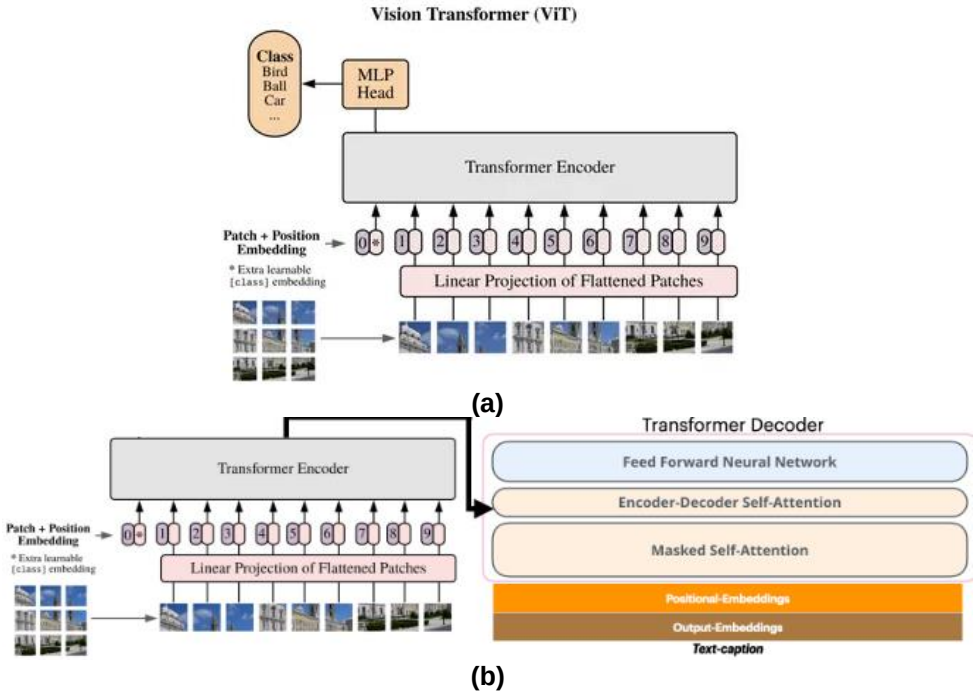

Figure 12: A ViT architecture encoder block (a) vs. a ViT-GPT2 encoder-decoder one (b).

After training and fine-tuning the source model in the AutoFormer manner, it will be possible to also feed it to the second stage of our pipeline to further increase its efficiency and explore the rest of the deployment options.

3 Model Deployment and Sharing

AI model deployment and sharing play a crucial role in ensuring that high-quality, robust ML and AI innovations are effectively translated into practical, real-world applications. First, deploying AI models on a development server is a critical phase to rigorously validate and test the various services. Moreover, the deployment processes involve the integration of these models into existing systems and workflows, enabling seamless operation in different environments under different conditions. On the other hand, model sharing refers to the processes of distributing the model itself, facilitating its wider use and development by others.

VOXReality utilizes NLP and CV advancements to develop robust AI models, aiming to address the challenges of human-to-human and human-to-machine interaction in XR environments. Detailed information on the development of VOXReality AI models can be found in D3.1 “Advanced AI multi-modal for XR analysis V1” [12]. The VOXReality trained AI models and the developed AI tools are automatically deployed in development server for further validation and testing from the consortium members. The deployment procedure is automated through GitLab’s CI/CD pipeline, which has been configured by VOXReality to enable the automation of integration and deployment procedures. The pipeline utilizes GitLab CI/CD platform's runners to execute these operations efficiently. A detailed description of VOXReality CI/CD pipelines is provided in D2.3 “Development Infrastructure and Integration Guidelines” [13].

Moreover, the trained models have been already deployed across three distinct use cases: VR Conference, Augmented Theatre and Training Assistant. A detailed description of the VOXReality application's implementation for each use case is provided in Section 4. Beyond these initial applications, the models can also be utilized by external application developers for a variety of tasks. To facilitate this adaptability, comprehensive deployment guidelines are provided outlining various deployment options. Additionally, all developed VOXReality AI models are made publicly available for sharing on the Hugging Face platform.

It is important to highlight that all research outputs of the project are publicly available, supporting the commitment of VOXReality to open science. This ensures that stakeholders can access and utilize these outputs. Specifically, the research outputs of the VOXReality project can be accessed through various repositories, which are listed here:

1. **VOXReality GitLab**, containing inference code and AI tools.
   https://gitlab.com/groups/horizon-europe-voxreality
2. **VOXReality DockerHub**, hosting docker images that encapsulate the operating environment, the AI models and the code required to utilize the model effectively.
   https://hub.docker.com/u/voxreality
3. **VOXReality HuggingFace**, listing various VOXReality AI models available for use.
   https://huggingface.co/voxreality

These research outputs can be used in various combinations by end users to leverage the VOXReality assets in their applications:

1. **Using GitLab Source Code to Create RESTful Services.** Users can employ the source code from VOXReality GitLab. It is recommended to create a Conda environment including the requirements of each service. The VOXReality pretrained models are obtained from Hugging Face. The source code from VOXReality is designed to employ RESTful architecture, enabling the creation of services that expose one or more endpoints. These endpoints facilitate efficient interaction with the service, following standard communication practices between computer systems. In VOXReality, FastAPI is used for building APIs with python.
2. **Direct Use of Docker Images.** Users can directly use the Docker image from VOXReality Docker Hub or Docker Compose from VOXReality GitLab. The images include the operating environment and the corresponding VOXReality pretrained model, which is automatically retrieved from VOXReality Hugging Face.
3. **Download Hugging Face models.** Users can download and fine-tune the VOXReality pretrained models from Hugging Face for specific tasks. Subsequently, they can use the source code from GitLab to build services that expose one or more endpoints, following the RESTful architecture. Alternatively, users can also build a Docker Image using this approach.

3.1 Deployment of VOXReality AI Models in Development Server

The deployment for VOXReality in the development server is a critical part of the development process. This stage enables the AI engineers to validate, test and refine the AI models within controlled, real-world scenarios, providing invaluable feedback that is essential for further enhancements. These activities can be characterized as laboratory tests and are carried out by the consortium members, with the aim of ensuring that all the components work together effectively. Detailed information and characteristics of VOXReality development environment are presented in D2.3 “Development Infrastructure and Integration Guidelines” [13].

Automatic deployment to the development server is essential of the development workflow followed by VOXReality, facilitated by the robust CI/CD pipeline. This automated system ensures that every code commit triggers a series of events, starting with the integration of new software features into the module’s functionality. Code changes are committed to a dedicated development branch in GitLab, triggering the CI/CD pipeline that executes any preconfigured unit tests. Successful tests lead to code merging into the main branch, while failures prompt necessary revisions. The CD phase then automates packaging and prepares the software for deployment, resulting in Docker images that are pushed to VOXReality DockerHub and then deployed to the development server for validation and testing. The CI/CD pipeline can be further enhanced towards security processes by applying SAST.

GitLab CI/CD can deploy jobs to build Docker images and publish them to a container registry. The basic steps to enable GitLab CI/CD on a VOXReality GitLab project and a sample pipeline template are described below.

Dockerfile

The first step includes the creation of the Docker file in the root of the repository. An example of Dockerfile is presented in Figure 13.

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.9

RUN apt-get update && \
      apt-get -y install sudo

RUN sudo apt install nano

COPY ./app/requirements.txt /app/requirements.txt

RUN pip install -r requirements.txt

COPY ./app/main.py /app/main.py
COPY ./app/openapi.json /app/openapi.json
COPY ./app/model /app/model

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]
```

Figure 13: Example of Dockerfile.

GitLab CI/CD Pipeline

The CI/CD pipeline is defined by the `.gitlab-ci.yam` configuration file, which specifies the stages and jobs that make up the CI/CD pipeline. The file is also created in the root of the repository. The GitLab Static Application Security Testing (SAST) can be enabled by navigating to ***Secure > Security Configuration***, to analyse our source code for known vulnerabilities.

In addition to the predefined Group Variables (*DOCKER_USER*, *DOCKER_PASSWORD, etc*), the *DOCKER_IMAGE* repository variable is specified by navigating to **Settings > CI/CD > Variables** in the repository. A new variable can be added as it is depicted in Figure 14. All the repository variables are visualized in Figure 15.

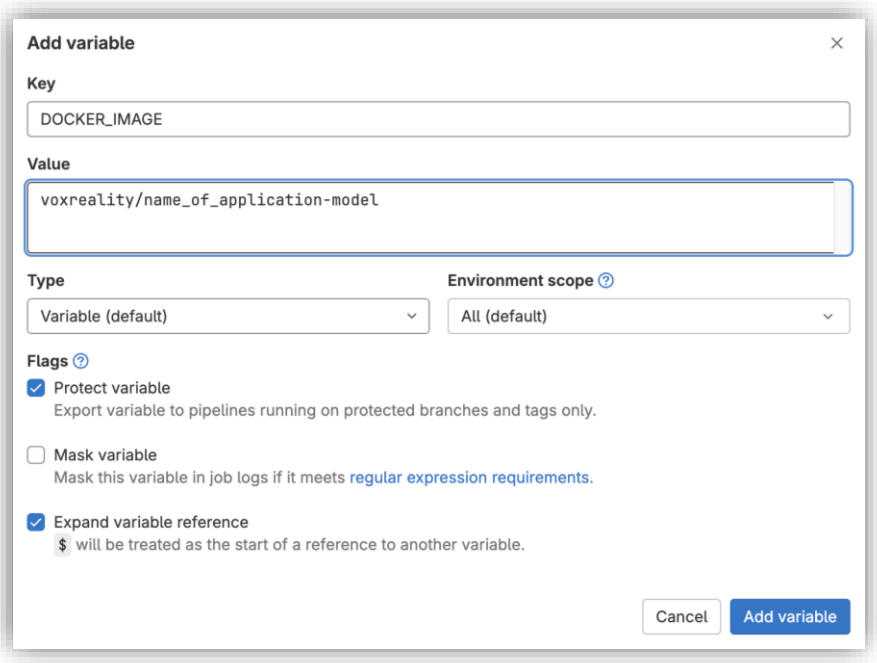

Figure 14: GitLab CI/CD Add variable.

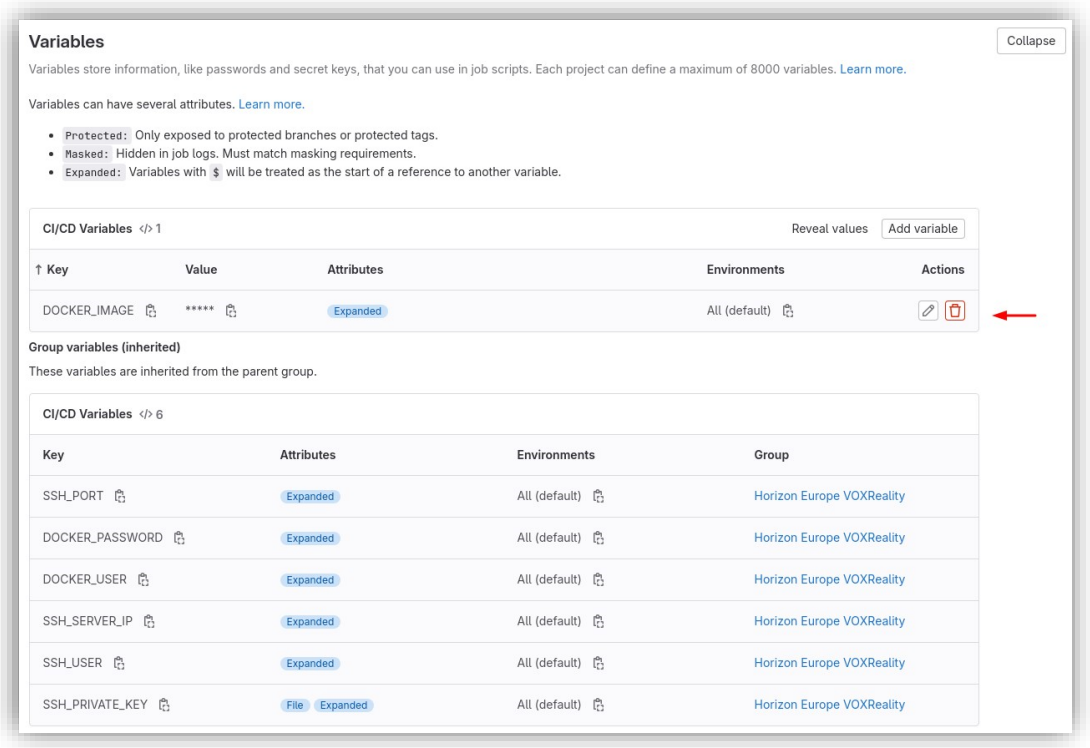

Figure 15: GitLab CI/CD Repository Variables.

Template .gitlab-ci.yaml file

In the provided *.gitlab-ci.yml* template file (Figure 16), the only necessary modification is to adjust the final command in the deploy stage to suit the specific application-model. This last command is responsible for running the application as a Docker container:

```
docker run -d -p 8080:80 -name $CI_PROJECT_NAME $DOCKER_IMAGE:$CI_COMMIT_TAG
```

The container name is the variable *$CI_PROJECT_NAME*, which is the project's name as shown in the URL.

```
stages:
- build
- test
- deploy

include:
- template: Security/SAST.gitlab-ci.yml

build:
  services:
  - docker:dind
  stage: build
  before_script:
  - echo "$DOCKER_PASSWORD" | docker login --username "$DOCKER_USER" --password-stdin
  script:
  - |
    if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
      tag=""
      echo "Running on default branch '$CI_DEFAULT_BRANCH': tag = 'latest'"
    else
      tag=":$CI_COMMIT_REF_SLUG"
      echo "Running on branch '$CI_COMMIT_BRANCH': tag = $tag"
    fi
  - docker build --pull -t "$DOCKER_IMAGE${tag}" .
  - docker push "$DOCKER_IMAGE${tag}"
  rules:
  - if: "$CI_COMMIT_BRANCH"
```

```
    exists:
    - Dockerfile

build-tags:
  stage: build
  before_script:
  - echo "$DOCKER_PASSWORD" | docker login --username "$DOCKER_USER" --password-stdin
  script:
  - docker build --pull -t "$DOCKER_IMAGE:$CI_COMMIT_TAG" -t "$DOCKER_IMAGE:latest"
    .
  - docker push "$DOCKER_IMAGE:$CI_COMMIT_TAG"
  - docker push "$DOCKER_IMAGE:latest"
  only:
  - tags

sast:
  stage: test

unit-test:
  image: alpine:3.18.0
  stage: test
  script:
  - echo "Running unit tests... This will take about 10 seconds."
  - sleep 10
  - echo "Tests passed succesfully"

lint-test:
  image: alpine:3.18.0
  stage: test
  script:
  - echo "Linting code... This will take about 5 seconds."
  - sleep 5
  - echo "No lint issues found."

deploy:
  image: alpine:3.18.0
  stage: deploy
  script:
  - chmod og= $SSH_PRIVATE_KEY
  - apk update && apk add openssh-client
  - ssh -p $SSH_PORT -i $SSH_PRIVATE_KEY -o StrictHostKeyChecking=no
$SSH_USER@$SSH_SERVER_IP "docker login -u $DOCKER_USER -p $DOCKER_PASSWORD"
  - ssh -p $SSH_PORT -i $SSH_PRIVATE_KEY -o StrictHostKeyChecking=no
$SSH_USER@$SSH_SERVER_IP "docker pull $DOCKER_IMAGE:$CI_COMMIT_TAG"
  - ssh -p $SSH_PORT -i $SSH_PRIVATE_KEY -o StrictHostKeyChecking=no
$SSH_USER@$SSH_SERVER_IP "docker container rm -f $CI_PROJECT_NAME || true"
  - ssh -p $SSH_PORT -i $SSH_PRIVATE_KEY -o StrictHostKeyChecking=no
$SSH_USER@$SSH_SERVER_IP "docker run -d -p 8080:80 –name $CI_PROJECT_NAME
$DOCKER_IMAGE:$CI_COMMIT_TAG"
  only:
  - tags
```

Figure 16: Template of .gitlab-ci.yaml file

Trigger GitLab CI/CD

The GitLab CI/CD is triggered in the following cases:

- When **pushes to main**, the first two stages are triggered as shown in Figure 17. The two phases are the build and test. The *build* job also pushes the $DOCKER_IMAGE to DockeHub with a tag **latest**.

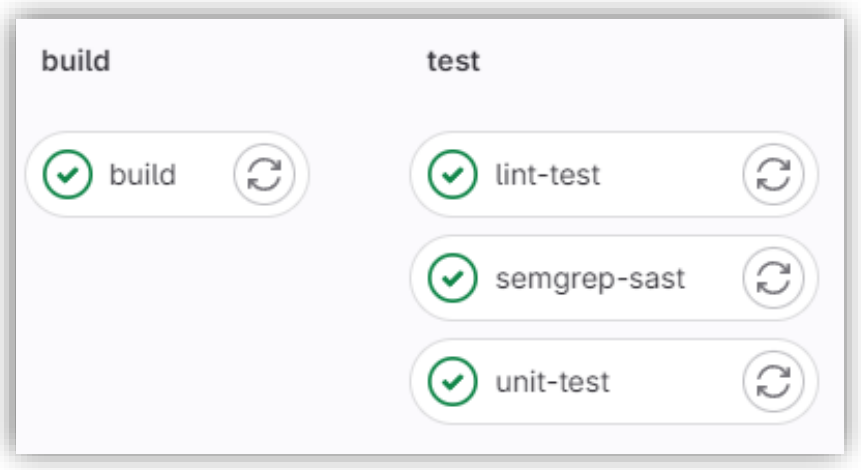

Figure 17: GitLab CI/CD Pipeline when push to main.

- When **creates or pushes a tag**, all stages are triggered, including build, test and deploy. A new tag is created by navigating to **code > tag > new tag,** as it is illustrated in Figure 18. The *build-tags* job also pushes the $DOCKER_IMAGE to DockerHub with tags **latest** and **$CI_COMMIT_TAG,** which is the commit tag name (e.g., v0.0.1). It is recommended using only Semantic Versioning. The *deploy* job connects to deployment server, pull the docker image with the specified tag and run the docker run command to start the container. It also removes the previous version of the container if exists. All the triggered jobs are displayed in Figure 19.

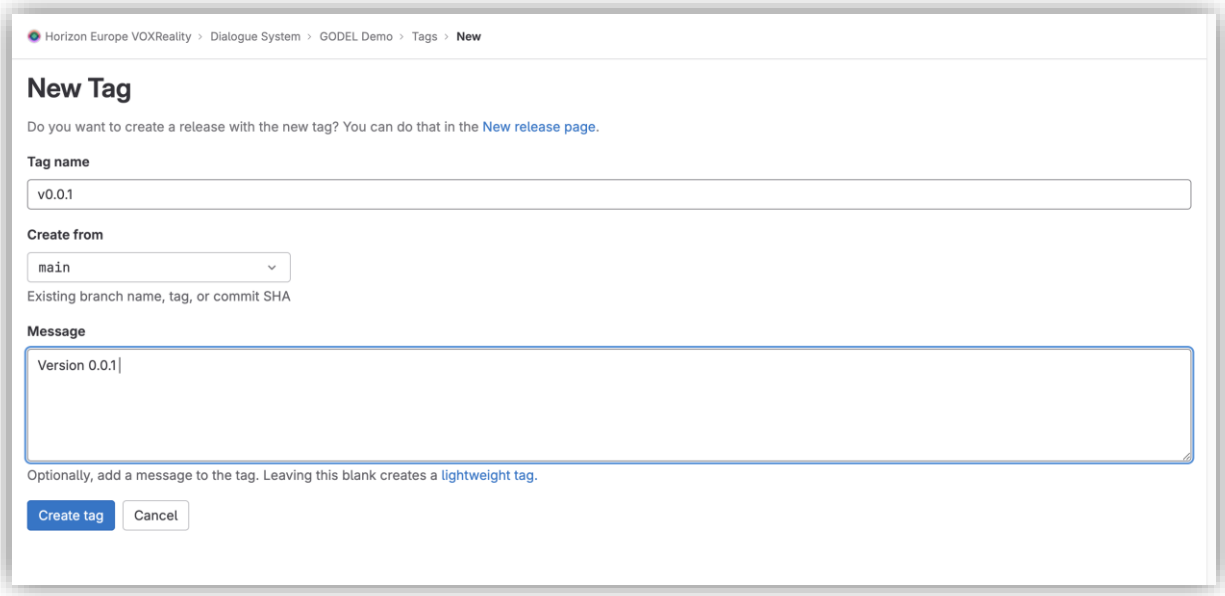

Figure 18: GitLab CI/CD Create a new tag.

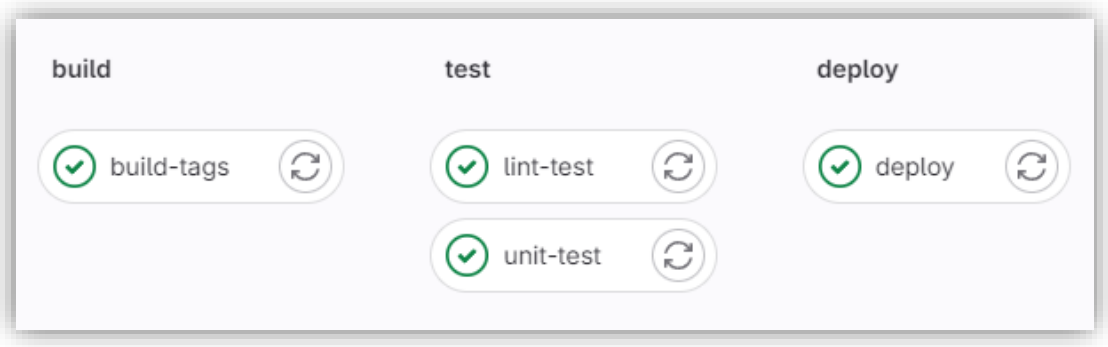

Figure 19: GitLab CI/CD Pipeline when create a new tag.

Summary of steps for automated deployment in development server

The above-described procedures for automatic deployment in VOXReality development server can be summarized here:

1. **Dockerfile Creation.** A Dockerfile is created in the root of the repository to define the environment in which the application will run.
2. **CI/CD Configuration.** A `.gitlab-ci.yaml` file is also placed in the root of the repository. This configuration file defines how the GitLab Runner executes the CI/CD jobs, orchestrating the build and the deployment process.
3. **Security Measures** (Optionally). Enable the Static Application Security Testing (SAST) to analyze the code for known vulnerabilities.
4. **Repository Variables.** A repository variable named $*DOCKER_IMAGE* is specified under *Settings* > *CI/CD* > *Variables*, which is used in the pipeline to refer to the Docker Image.
5. **CI/CD Pipeline Execution.** GitLab CI/CD pipeline is triggered by either pushing to the main branch or creating or pushing a new tag:
    a. **Push to main**. The *build* and *test* jobs are triggered. After a successful build, the Docker image, denoted as $*DOCKER_IMAGE,* is pushed to DockerHub with the 'latest' tag.
    b. **Create or push a tag**. When a new tag is created or pushed the *build, test* and *deploy* jobs are triggered. This is achieved by *code> tag > new tag.* The Docker image is then built and sent to DockerHub. Following this, the corresponding container starts running on the development server.

3.2 Deployment Guidelines

In the VOXReality project, a widely adopted and standardized software architectural style for communication between computer systems is following, which is the RESTful architecture as it is described in VOXReality Integration Guidelines of D2.3 “Development Infrastructure and Integration Guidelines” [13]. Specifically, FastAPI[1], a modern, fast web framework for building APIs in Python, is employed. In this approach, each service exposes one or more endpoints to which clients can send requests. These endpoints are essentially URLs through which the services are accessible. The endpoints serve as the interface for the service, allowing clients to interact with it using HTTP methods. Therefore, the deployment of VOXReality AI models

[1] https://fastapi.tiangolo.com/

is effectively managed through the creation of FastAPI applications. Details about the API calls of each service are provided in Appendices of D3.1 “Advanced AI multi-model for XR Analysis” [12]. Additionally, to enhance scalability, these applications can be containerized using Docker.

The VOXReality AI models can be deployed in various hardware environments following different deployment methods. This section provides general guidelines that are applicable to most of those models. However, considering the unique deployment requirements and potential modifications for each AI model, it is advisable to also refer to the individual GitLab pages of each model for more specific and targeted guidelines. On these individual GitLab pages, one can find deployment instructions for all deployment methods. It should be noted that all the following deployment methods create RESTful Services.

The deployment options that are described here include:

1. Deployment from Source Code
2. Containerization
    a. Using single images from Docker Hub
    b. Using Docker Compose

3.2.1 Source code-Based Deployment

All the VOXReality code, including the inference code and AI tools, is publicly available in the GitLab group: https://gitlab.com/groups/horizon-europe-voxreality. The main thematic entities developed in the VOXReality project are organized as subgroups within this main group. Additionally, each subgroup may contain several projects, with each project providing a specific service as well as detailed documentation in the form of README files. The steps to set up and run the VOXReality models by utilizing the source code from GitLab repository are described in this section. By following these steps, the VOXReality AI models are utilized, and the local API is up and running for further development and testing, however it is important to select the appropriate subgroup and project that meets someone specific requirements.

Moreover, those guidelines describe the general steps for utilizing the source code from VOXReality GitLab repository accompanied either with the corresponding VOXReality AI model in Hugging Face repository or with a locally stored model. The locally saved model can be one that has been directly downloaded from VOXReality Hugging Face repository or that has been finetuned.

1. **Clone the Repository.** Start by cloning the project repository from GitLab using the command:

```
git clone https://gitlab.com/horizon-europe-voxreality/subgroup/project.git
```

2. **Create a Conda Environment.** If you have not already, create a new Conda environment with Python 3.8 by running:

```
conda create --name env_name python=3.8
```

3. **Activate the Environment.** Activate the created Conda environment with:

```
conda activate env_name
```

4. **Install Dependencies.** Navigate to the project directory and install the required dependencies using pip:

```
cd project
pip install -r requirements.txt
```

5. **Navigate to Application Directory.** Change into the application's directory:

```
cd /app
```

6. **Configure AI Model Storage Path.** Set the path where the AI model is stored by editing the `config.yaml` file. This path should point out to the relevant AI model within the VOXReality Hugging Face repository, or to a local version of the AI model that has been either directly downloaded from Hugging Face or further fine-tuned locally.
    - To do this manually, open the `config.yaml` file in text editor and modify '*PRETRAINED_MODEL_NAME_OR_PATH'* with the correct path.
    - Alternative, update the `config.yaml` file via the terminal.

7. **Launch the API Locally.** Start the local server with the Uvicorn command that points to your application.

```
uvicorn main:app --host 0.0.0.0 --port 8000 --reload
```

The above command starts the Uvicorn server hosting the application defined as "*main:app*".

3.2.2 Container-Based Deployment

The VOXReality CI/CD pipeline is configured to automatically build and upload Docker images to the VOXReality DockerHub: https://hub.docker.com/u/voxreality. These images are readily available for end users to deploy within their applications, as well as for further development and testing, encapsulating both the operating environments and the AI models. The currently available VOXReality docker images are presented in Figure 20.

This section provides a description of two containerization strategies for deploying VOXReality AI models, which are:

1. **Using single images for VOXReality Docker Hub repo.** This approach regards the deployment of AI models using pre-built, standalone Docker images available on Docker Hub. Each image runs as a separate container. This approach is ideal for directl deployments where a single container can fulfil the requirements.
2. **Using Docker Compose.** This method is essential when the applications require a more complex environment, involving multiple interdependent services. It allows to define and manage the multi-container scheme with ease, offering a more integrated deployment process. Additionally, with this method the multiple containers are orchestrated to work together.

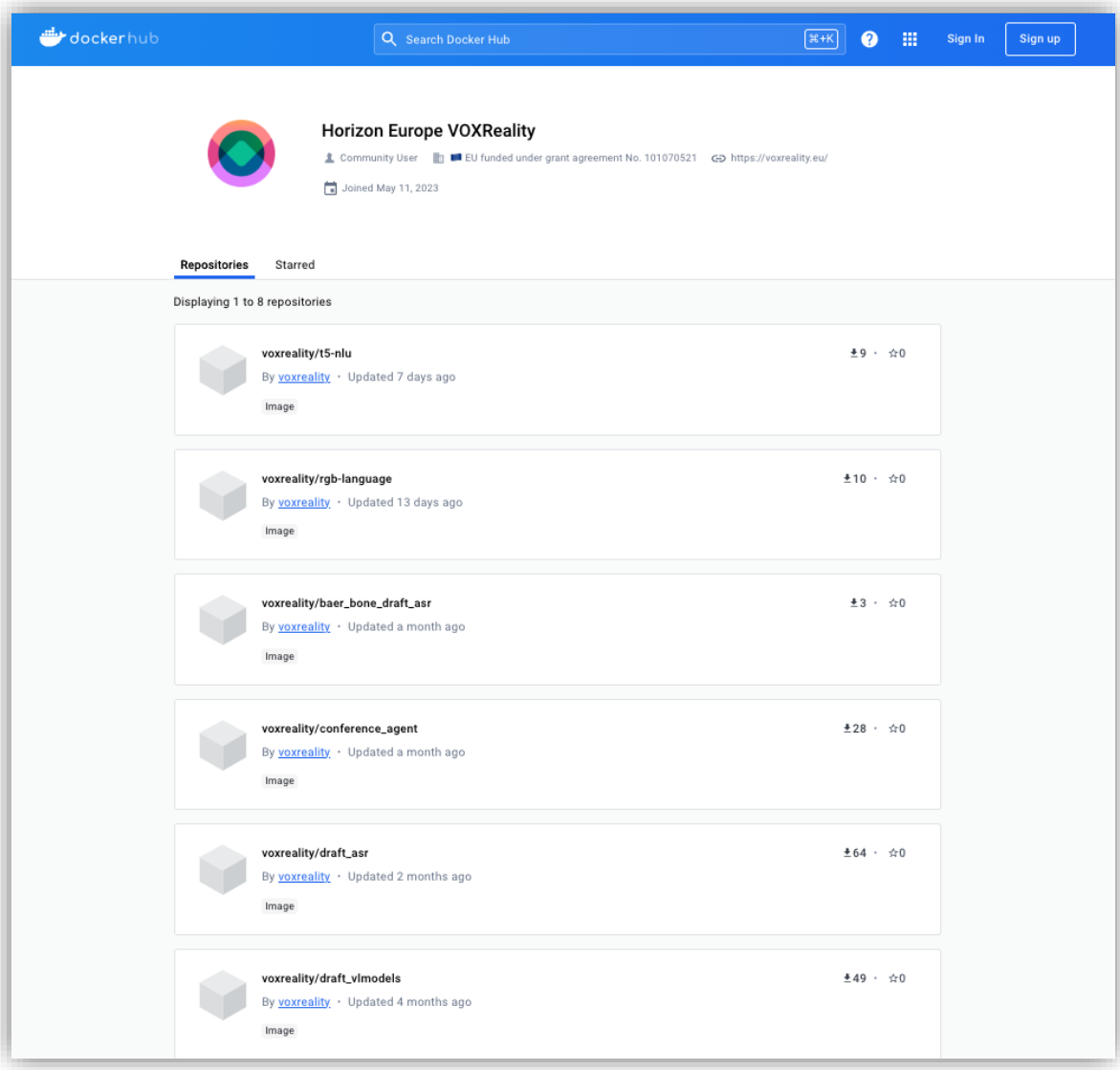

Figure 20: VOXReality DockeHub.

3.2.2.1 Deployment using Docker Hub Images

When deploying, the Docker images are pulled and run as containers in the target environment. This containerization guarantees uniform operation of VOXReality AI models in any environment, effectively abstracting away any discrepancies in underlying hardware or software.

Depending on the intended hardware, different Docker commands are used to deploy VOXReality models, tailored specifically for CPU or GPU environments. However, it is recommended to deploy the VOXReality AI models on GPU for optimal performance.

For deployments using a **CPU**, the Docker command is as follows:

```
docker run -p 8000:8000 <name_of_image>
```

For deployments intended to utilize **GPU**, the command is slightly modified to enable GPU access:

```
docker run --gpus all -p 8000:8000 <name_of_image>
```

The `--gpus all flag` assigns all available GPUs to the container, which is necessary for models that require or significantly benefit from GPU acceleration.

In both cases, `<name_of_image>` should be replaced with the actual name of the Docker image that contains the VOXReality model to be deployed. These commands ensure that the VOXReality models are deployed in a Docker container with the appropriate hardware access for optimal performance.

3.2.2.2 Deployment using Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Docker Compose, a YAML file, typically named `docker-compose.yml` is used to configure the application's services, networks and volumes. This file serves as a template for Docker to understand how to run and interconnect the various containers that make up the application. Figure 21 displays an example of the `docker-compose.yml` file that combines the NMT, the ASR, and the conference agent.

```
version: "3.9"

services:
  conference_agent:
    image: voxreality/conference_agent:v1
    ports:
      - "8000:8000"
    env_file: ".env"
    volumes:
      - pdfs/:/app/pdfs:ro
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
    restart: unless-stopped
  umlib:
    image: voxreality/draft_asr:v1
    ports:
      - "5033:5033"
    deploy:
      resources:
        reservations:
          devices:
              - driver: nvidia
                count: all
                capabilities: [ gpu ]
    restart: unless-stopped
```

Figure 21: Example of docker-compose.yml file

The `docker-compose.yml` files of VOXReality are configured to use Docker Images hosted in the VOXReality Docker Hub repositories. Meanwhile, the various `docker-compose.yml` files are maintained and can be founded in the VOXReality GitLab repo.

To run the Docker Compose in background, the following command can be used:

```
docker-compose up -d
```

To run a specific service defined in a `docker-compose.yml` file, the '`docker-compose up`' command is followed by the name of the service that it is desired to start:

```
docker-compose up -d [service_name]
```

In this case, it is important to remember that the service names used in the above command should match exactly as it is defined in the `docker-compose.yml` file.

3.3 Model Sharing

Hugging Face is a central platform in the AI community for sharing AI models, particularly those related to NLP. It provides a central hub where developers and researchers can upload their pre-trained models, making them accessible to the wider community. The platform supports a collaborative environment, allowing users to contribute to the development and improvement of models in various applications. Hugging Face allows for seamless integration of models into various projects through its comprehensive library of '*transformers*'. This library supports the download and use of these models for NLP applications as well as the fine-tuning. Sharing AI models through this platform offers numerous benefits, including increased visibility, community feedback and the potential for collaborative improvements. Specifically, sharing through the Hugging Face ensures that cutting-edge models are readily available for use and further development. This approach not only enhances the models but also contributes to the advancement of the field.

The trained VOXReality AI models, after undergoing extensive testing and validation, are uploaded to the VOXReality Hugging Face Community by AI Engineers. Figure 22 illustrates the VOXReality Hugging Face repository. This dedicated repository on Hugging Face allows researchers to easily discover and utilize VOXReality AI models, leveraging the comprehensive documentation provided for each model to enhance their research and applications.

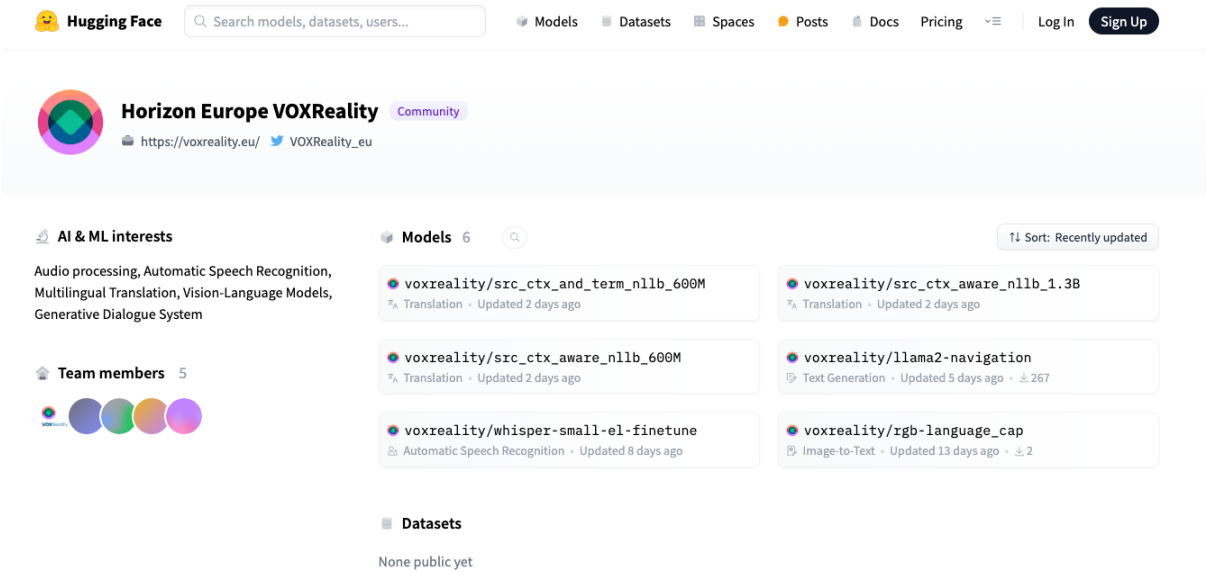

Figure 22: VOXReality Hugging Face repository.

Each uploaded VOXReality model is accompanied by detailed documentation that covers various aspects such as the model's architecture, training data, performance benchmarks, and intended use cases. Additionally, this documentation often includes examples and tutorials to help users better understand and utilize the models effectively. The documentation for each model is provided in a "Model Card," which are files that accompany the models, offering a comprehensive overview and guidance for users. Moreover, the "Model Card" includes metadata, providing essential information such as the model's name, version, language, and license. The metadata acts as an informative summary, supporting easy navigation, AI model discovery and easier use of each model. It greatly simplifies the process for users to search for and filter models based on specific criteria like language, model type, or application domain, ensuring a more efficient and user-friendly experience.

The following guidelines aims to provide clear instructions on how to access VOXReality AI models form Hugging Face repository, enabling efficient integration of those models into various applications.

1. **Create a Hugging Face Account (Optional).** While this is not necessary, creating an account on the Hugging Face can provide access to additional features.
2. **Select the Desired Model.** In the VOXReality Hugging Face repository, select the model that fits your need.
3. **Access the model.** There are 2 ways to access the pre-trained AI models in Hugging Face VOXReality repository.
    a. **Using Git to download the AI model.**
    Since all models on the Hugging Face are Git repositories, the desired model can be cloned locally by running:

```
git clone clone https://huggingface.co/voxreality/<model_name>
```

    b. **Using Transformers Library**
        i. **Set up the Environment.** If you have not done already, create a Conda environment and install the needed libraries. This can be done, following the next steps:
            1. **Create a Conda Environment.** If you have not already, create a new Conda environment with Python 3.8 by running:

```
conda create --name env_name python=3.8
```

2. **Activate the Environment.** Activate the created Conda environment with:

```
conda activate env_name
```

3. **Install the Hugging Face "*transformer*" library.** This can be done by running.

```
pip install transformers
```

4. **Load the model directly.** You can directly use the model in your python script using the following command. The specific commands for each model are generally provided by Hugging Face under the "*Use in Transformers*" section:

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer= AutoTokenizer.from_pretrained("voxreality/model_name")
model=AutoModelForCausalLM.from_pretrained("voxreality/model_name")
```

4 VOXReality XR Applications

The pre-trained VOXReality AI models have been deployed in three different use cases: VR Conference, Augmented Theater and Training Assistant. Table 4 presents the VOXReality components integrated into each use case.

Table 4: VOXReality components used in each use case.

Components / Use Case	Automatic Speech Recognition	Neural Machine Translation	Vision Language Models	Dialogue System
VR Conference	X	X	X	X
Augmented Theatres	X	X	X	
Training Assistant	X			X

This section details the design and implementation of the VOXReality applications, covering both conceptual and practical aspects implemented by M17. Any updates and changes will be documented in the deliverable D4.2 “Model deployment analysis V2”. Specifically, this section includes the creation of 3D models and scenes for each XR application, as well as the workflow of the applications. Additionally, this section discusses the necessary tools, software, and hardware required for building these applications, along with the key algorithms and programming techniques implemented in each solution. It also provides insights into the user interface design. Finally, the section concludes with a brief description of how the user and technical requirements have been met, while the detailed description will be provided in deliverables of WP2 and WP5.

4.1 VR Conference

The VR Conference application emulates most recognisable attributes of a real-life professional conference setting. The experience is enhanced by a real-time multilingual translation service between users and the introduction of a dedicated, voice driven Virtual Agent. The Agent intends to help users during their navigation to the conference by providing navigation instructions through the virtual space, answering to questions about the conference’s program, giving relevant info about the booths that exist in the conference area and by giving a description about the virtual scene. The presence of the Virtual Agent is meant to be non-intrusive to the users, who can choose to deactivate it and reactivate it at any time.

4.1.1 System Architecture and Design

4.1.1.1 3D Models and Scenes Design

From a spatial perspective, the application should simulate a real venue in a 3D world, incorporating all essential rooms required for hosting a conference. It has been decided that, among all available areas a conference venue may include, five of them will be designed:

1. The **Lobby Room**, working as the entrance to the main area of the venue.
2. The **Trade Shows Area**, being the main space of the conference, where exhibitor’s booths will be placed and access to all other rooms would be possible through it.
3. The **Business Room**, for one-to-one communication between participants.
4. The **Social Area**, for 1-to-1 communication, and
5. The **Conference Room**, where the conference session will take place, facilitating 1-to-many communication.

To increase performance and decrease loading time, each area will be a separate 3D scene and interconnection between scenes will be implemented through doors moving between them.

So far, 3 out of 5 spaces (3D scenes) have been designed and developed: the Lobby Room, the Trade Shows Area, and the Conference Room. The architectural design of the rooms aligns with the intended usage of every space and their functionalities. Rooms that serve a specific purpose include less spatial information to limit the level of distraction while multi-purpose spaces, such as the Trade Shows Area, have bigger dimensions and have increased complexity. Permitted communication type of each room will be promoted by the design in several ways: one-to-one communication can be highlighted with the existence of tables and chairs, and one-to-many can be introduced through an amphitheatrical allocation or the existence of a stage.

All 3D models integrated into the application have been created following the same principles. Their size is minimized as much as possible, by reducing the complexity of geometry and lowering the quality of textures, to control loading delays. The acceptable level of this optimization process is actively constrained by how realistic the final output would look like. For this to be feasible, a low-polygon aesthetic has been selected, adopting a simpler, more light-weighted overall style that promotes less complex geometrical shapes.

4.1.1.2 Application Workflow Diagram

The application core functionalities introduced to the users can be separated into two main categories. The first category incorporates the communication system between the user and their Virtual Agent, and the second one describes the pipeline needed for the real-time translation system.

Virtual Agent Functionalities

The Virtual Agent, being represented by a virtual avatar, is a non-intrusive entity, thus, a conversation with it is initialized only by the user. The starting point of the communication is enabling the Virtual Agent by pressing the corresponding toggle. The instantiated Virtual Agent entity will greet the participant with a welcoming message, in their language and will follow them during their browsing in the VR space. The workflow is illustrated in Figure 23.

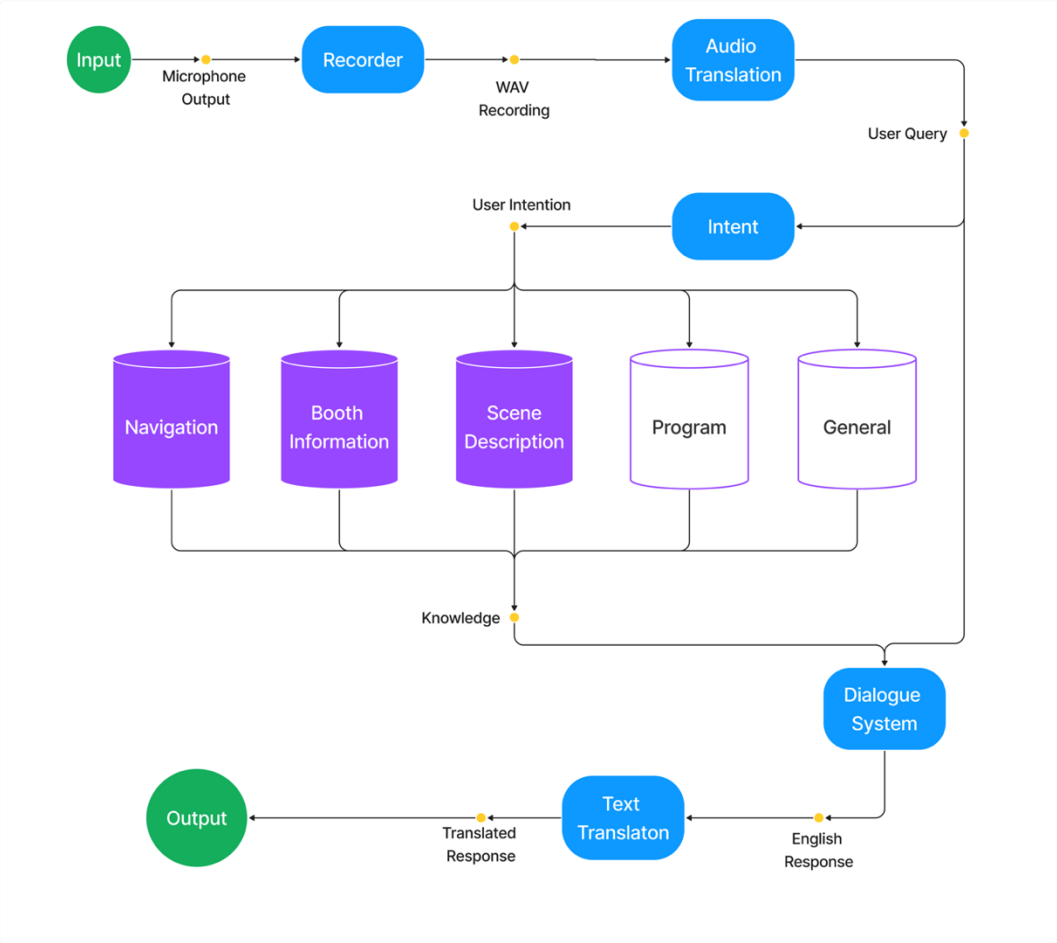

Figure 23: Workflow for communication with the Virtual Agent.

The user can ask a question via a push-to-talk mechanism, that records their voice while the Agent is active and prevents the microphone output from being networked to other participants. This way the conversation with the Virtual Agent remains private. Finishing a voice message triggers the first part of the workflow, that formats the microphone output to a WAV file and sends it to the Translate Audio endpoint, which combines the ASR (Automatic Speech Recognition) and the NML (Neural Machine Translation), along with the user's language and the translation's desired language, that is always English at this point of the workflow, as parameters. The response of this endpoint contains both the transcription and the English translation of the voice message. In case of an English-speaking user, both the transcription and the translation contain the same information.

The English text Is next propagated to the intent endpoint of the dialog agent that is responsible to analyze it, retrieve its context and return relative information. This component works as a router for the system, enabling the appropriate workflow depending on the requested task. The user can ask the Agent about five different topics:

1. Navigation
2. Conference Program
3. Booth Details
4. Scene Description
5. General

Each topic follows a different pipeline to generate the needed knowledge, whenever it is necessary, and serve it to the dialogue agent endpoint. In Figure 23, shapes with purple fill indicate the existence of a dedicated pipeline before the dialogue system while those with purple outline represent a simple propagation of the user query to the next component of the diagram. The response of the endpoint, being a human language English text, depending on the selected language of the user, may require to be inferenced to the text translation endpoint in order to be transformed into the correct language. The final text output is sent to the virtual agent and gets rendered on a text panel.

Navigation

When the application is launched, the navigation system (Figure 24) retrieves information about the 3D scene like its dimensions, the destination names and positions, possible anomalies of the 3D space, connecting points etc. Once this piece of information is gathered, the application creates a symmetrical node graph that maps the geometry of the space based on Graph Theory and is ready to receive request for navigation.

If the intent endpoint determines that the user is requesting navigation instructions, it extracts the desired destination name from the translated text and activates the navigation system to handle the request. Simultaneously, the system calculates the closest node to the user's position, and the destination node from the destination name. The Dijkstra algorithm receives those as arguments and calculates the shortest path from the starting to the destination node, resulting in a list containing all intermediate nodes in a sensitive order. To change this list to an accepted format for the dialogue agent, the system computes continuous movement parts of the navigation and the orientation of the turns and propagates them to a module that transforms received data to the appropriate format. The line and turn calculator output is also needed to create and export correct graphic cues (Figure 25) to the 3D scene such as arrows and lines. The color palette of the graphic cues contains vivid shades – green lines and blue arrows – and is selected in such a way that the cues can be easily distinguished from the rest of the environment.

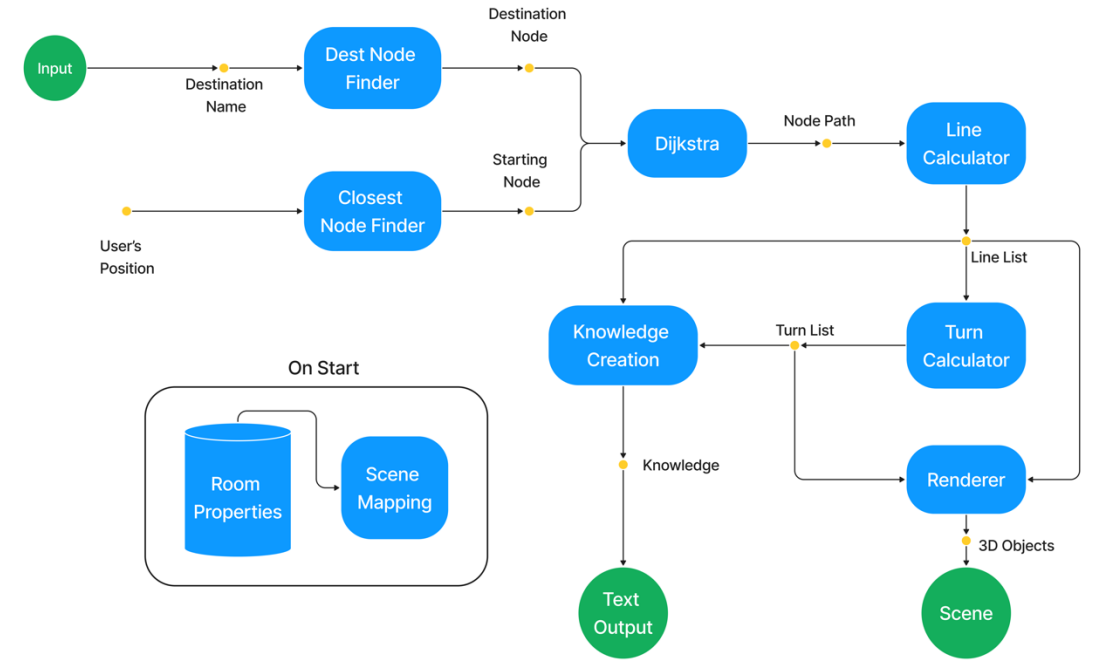

Figure 24: Workflow of the Navigation System.

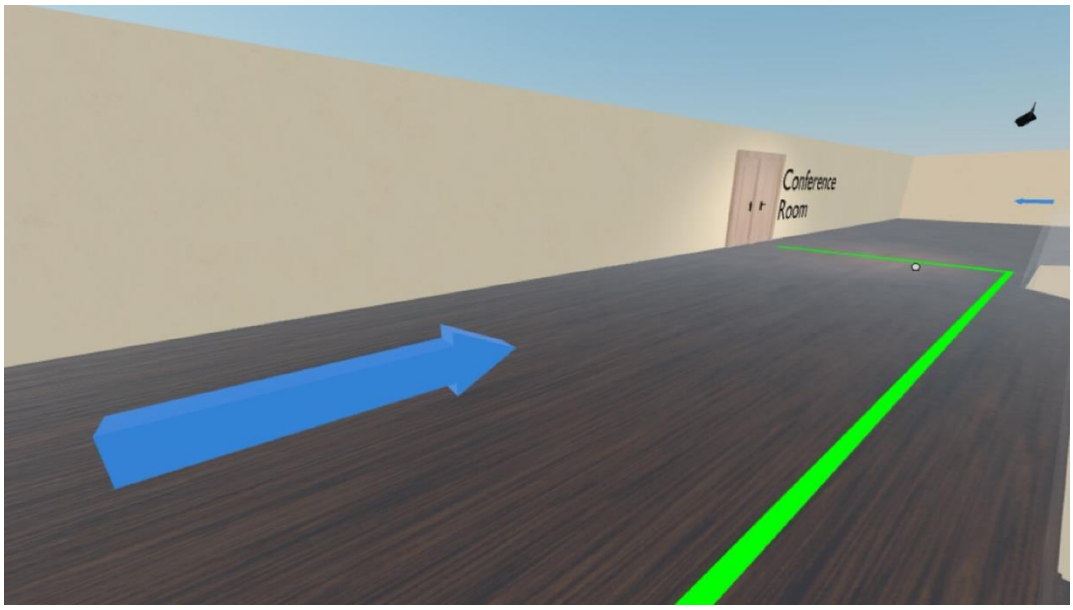

Figure 25: Graphic Cues of the Navigation System.

Booth Description

Information about the owner of each booth inside the trade shows area is retrieved as meta data when entering this room and gets associated with the respective 3D objects inside the VR scene (Figure 26). If the intent endpoint detects that the user requests additional information about a booth in the tradeshow area, the booth management system is activated. The request is sent to the system to determine which booth in the trade shows area is the request's target. Once this is specified, the metadata of the 3D objects get collected and reshaped in an accepted format by the dialog agent.

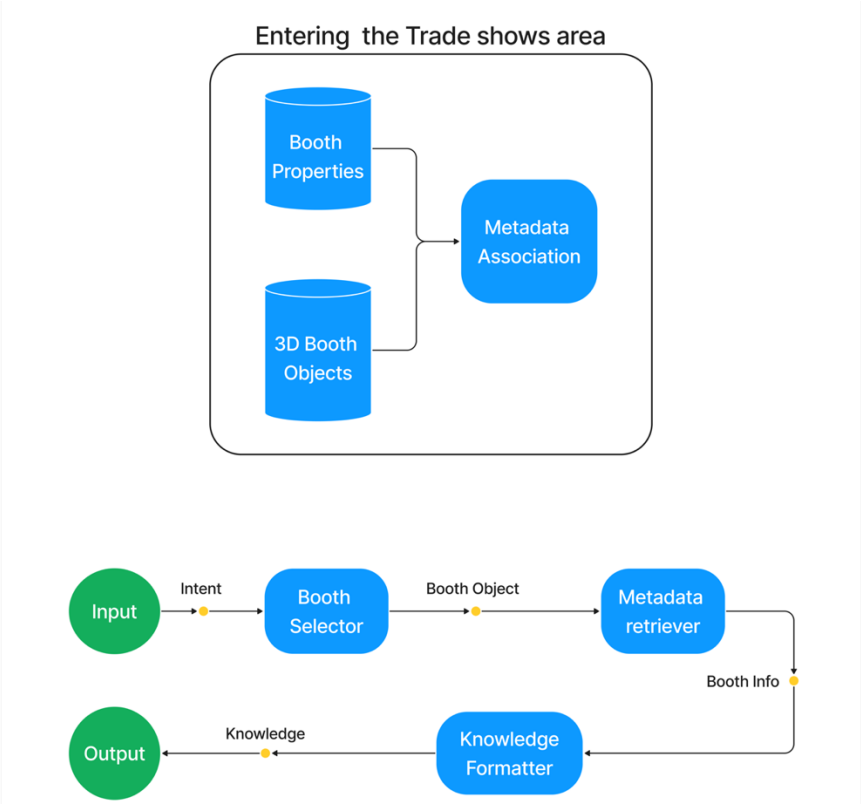

Figure 26: Workflow of the Booth Description System.

Scene Description & Remaining Functionality

User queries for scene description, call the screenshot module that takes a photo of the FOV (Field of View) of the participant and sends it to the VL Model for inference (Figure 27). The response of the VL Model is directly propagated to the Dialogue System.

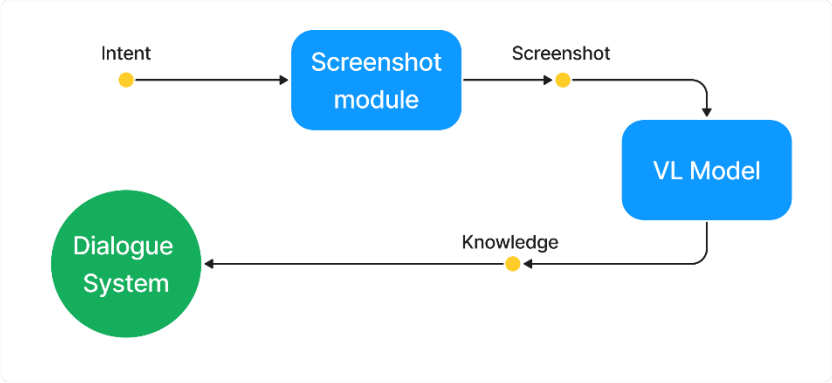

Figure 27: Workflow of the Scene Description System

The remaining functionalities, i.e. answering about the conference program and respond to general questions do not need a specific input from the application (Figure 28). In this case, the dialogue system is completely responsible for handling the request and the application simply redirects the user query and user intention directly to its endpoint for inference.

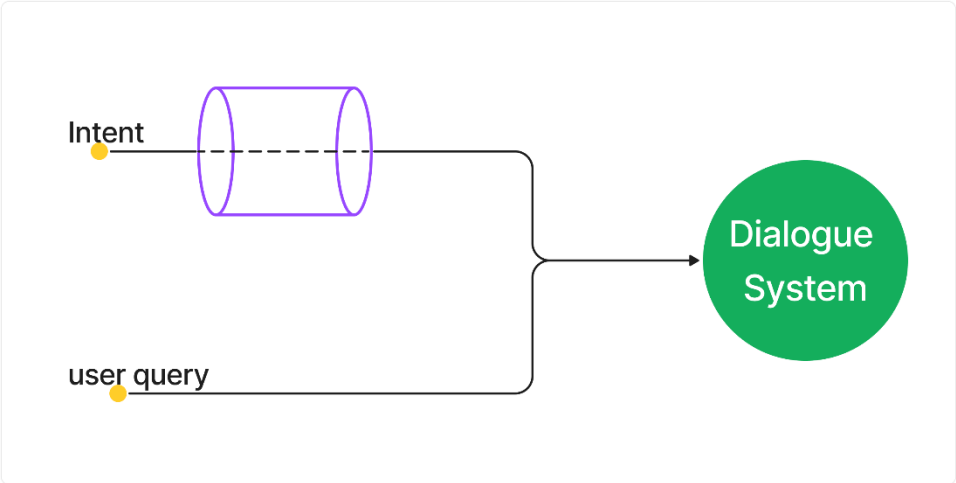

Figure 28: Workflow of the Conference Program and General Questioning System.

Real-time Translation

The real-time translation system is not managed by the virtual agent entity. In spaces that allow communication between participants, when the virtual agent is not enabled the translation system is activated in stand-by mode (Figure 29).

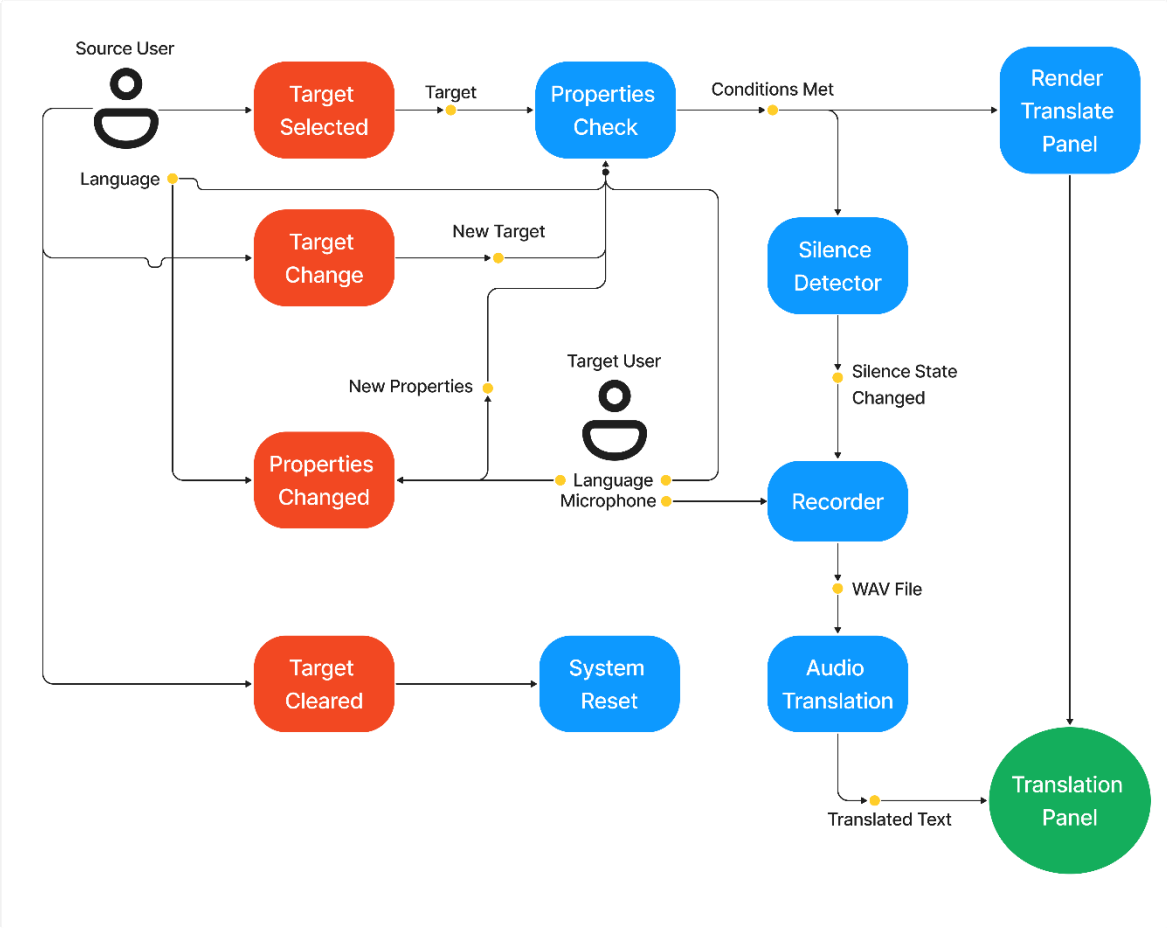

Figure 29: Workflow of the real-time Translation System.

If a user wishes to start receiving a translation from another participant, he/she must choose him/her as its desired target. When a user target is selected, the translation system checks whether both communicating parties have selected their speaking language. Only when all those conditions are met does the system start the actual translation, otherwise it remains in stand-by mode.

To achieve translation in real time, the networked microphone output of the target user gets recorded by the recording module and simultaneously analyzed by a silence detector. If the detector recognizes noise from the microphone, the recording starts, and when it detects a period of silence, the translation stops and the audio file gets translated using the Audio Translation endpoint, having as parameters the speaking language of the source and the target user. The response of the endpoint gets rendered directly in the 3D scene, on a panel at the front of the target user.

The system also listens for any events that may need to change some of its inference parameters such as, removing source and/or target language or changing their values. In case of absence of one or more parameters, the system remains in standby mode until they change to an acceptable value again.

The source user can always deactivate the translation or change target and the system will automatically reset.

4.1.2 Implementation Details

4.1.2.1 Development Environment Setup

The VR Conference application is being developed based on Hubs[2], an open-source web application for interacting in networked 3D spaces, powered by Mozilla. For the development process, the Community Edition of Mozilla Hubs is hosted, on a Kubernetes Cluster that automatically manages all the necessary projects of the application, using Google Kubernetes Engine[3] running in a 2x2vCPU machine with 4GB of RAM. In parallel, minor changes to the application are made directly using the Development Server of Mozilla, to avoid deployment latency.

The VR conference, being a web application, is accessible via every web browser that supports WebGL. The virtual reality mode of the application additionally requires a VR Headset and a web browser supporting WebXR. Almost all modern VR Headset of the industry are compatible with the application and in case of a standalone model, a computer is not required, as access can be granted directly from the headset's browser or by streaming the application from the computer to the headsets. For development purposes Meta Quest 2[4] and Meta Quest 3[5] are being used so due to compatibility reasons streaming the application to the headsets requires a MS Windows OS client computer.

[2] https://hubs.mozilla.com/
[3] https://cloud.google.com/kubernetes-engine
[4] https://www.meta.com/quest/products/quest-2/
[5] https://www.meta.com/quest/quest-3/

The server side of the application, named Reticulum, is written in the Phoenix framework of the Elixir programming language. Reticulum is responsible for networking all information except communication media such as the microphone and camera feedback. Those get managed by a separate Node.js webRTC server based on the open source MediaSoup project, called Dialog. Another worth mentioning project, that is part of the application, is Spoke, a lightweighted scene editor written also in JavaScript.

All additional features implemented in Mozilla Hubs are part of modifying and extending the Hubs client code. The client project is a mixture of JavaScript and TypeScript languages, that manages both 3D objects and UI elements. The User Interface is written in the *React.js* framework while the 3D scene is being developed by a combination of the *THREE.js* library and the Networked A-FRAME framework, that works as wrapper for *THREE.js* elements. All physics in the 3D world is being computed by the *ammo.js* library.

4.1.2.2 3D Models and Scene Creation

Designing the virtual environment of the application is conducted mostly using Blender, an open-source 3D creation suite, to sculpt the geometry and customize the shading of 3D objects. Creating scenes that represent different rooms of the virtual conference benefited from the Archimesh extension, an architectural Blender tool, which allows for simple, geometrically light-weighted room designs (Figure 30). Decorative elements of the space including objects required for the room, such as the booths of the trade show area are either free licensed 3D models, imported to the project from the internet or are created specifically for the VR conference application.

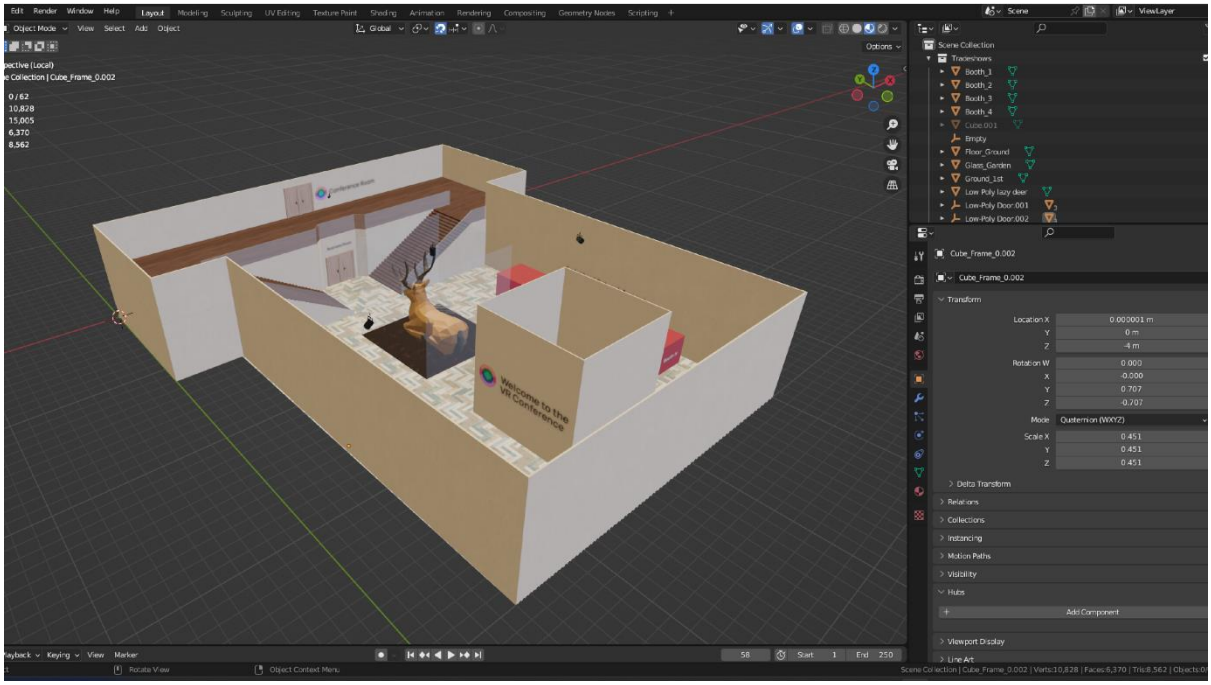

Figure 30: Blender Interface for room designing.

Shading elements that don't have materials is done using free licensed texture files from the internet to create new ones and assign them to the models (Figure 31). However, the textures must be of medium to low quality to avoid decreased performance on runtime.

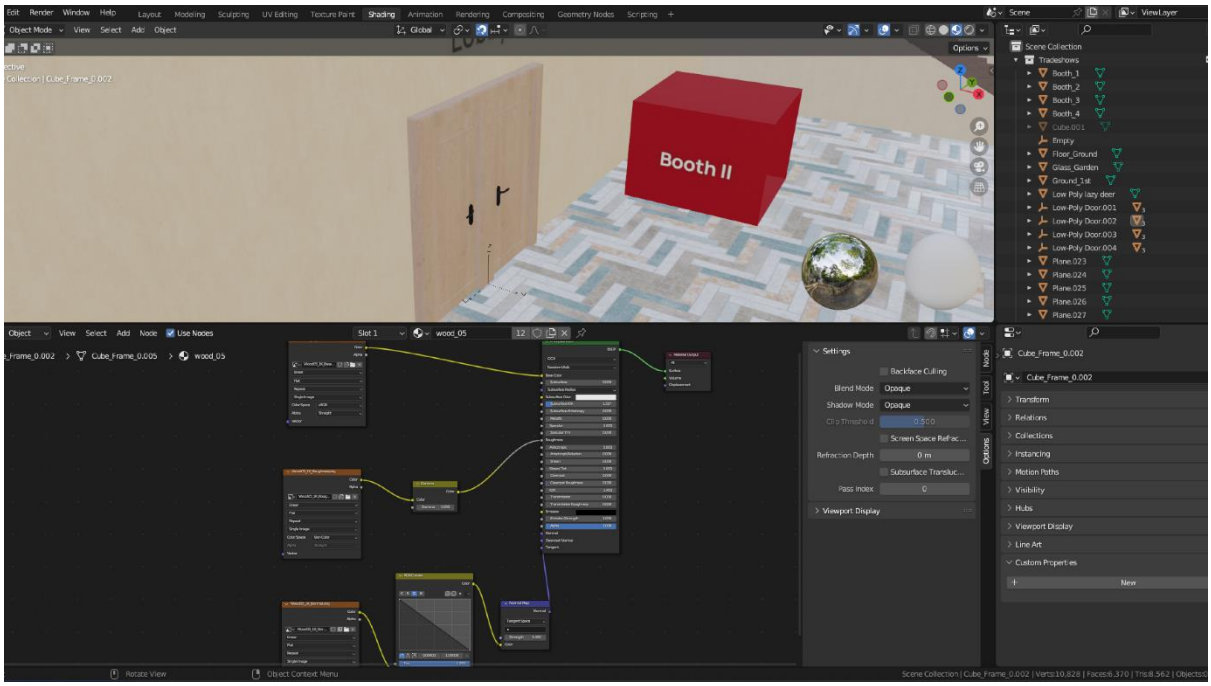

Figure 31: Blender Interface for shading.

Once a space is completed, in terms of design and shading, the models are exported as a single file using Blender's GLTF/ GLB exporter since GLB is the only acceptable format by Hubs for 3D model importing. Final editing stages happen in Spoke thought which the 3D model is imported inside a new scene and then lighting sources (Figure 32), Hubs custom components such as connection gates with other already finished rooms and spawn points are added to it and the scene gets published.

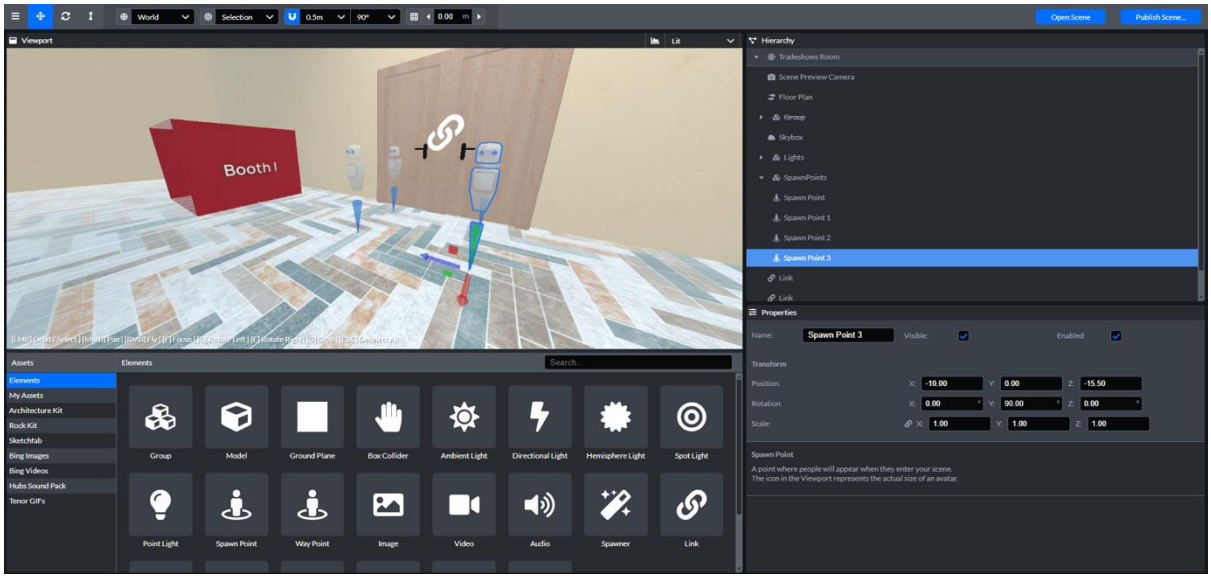

Figure 32: Mozilla's Spoke Interface.

When a participant enters inside a room of the application, a random published scene is loaded on their device. Since the scene of each room is configurable, publishing a finished scene enables the association of it with a specific room. With this process, three different VR

conference rooms have been created associated respectively with the Lobby Room, the Trade Show Area, and the Conference Room scene.

4.1.2.3 Core Algorithms and Techniques

Dijkstra Algorithm

The Virtual Agent assigned to each user of the application should provide efficient instructions for navigation when requested, both in textual and graphic format. The process of calculating the shortest path from a starting to a destination point is done by implementing the Dijkstra algorithm [14], which is a shortest path finding algorithm based on weighted graphs.

For the Dijkstra algorithm to function properly, when a user enters a scene and the application retrieves the properties of the room, a node grid is constructed on the fly to map the VR space. The grid's density is a configurable value that also gets retrieved with the fetched room properties. Extra nodes, such as available destination points of the navigation system and hardcoded areas are appended to the node grid.

To progress from a node grid to a weighted graph, specific rules to determine adjacency are applied to each node. The rules define the maximum distance between neighboring nodes, forbid diagonal edges and set the weight of each edge as the Manhattan distance between the connected nodes. The Manhattan distance of two points represents the sum of the absolute differences of their Cartesian coordinates.

Once the graph is calculated, the system is ready to accept navigation requests. To process the user query, Dijkstra algorithm assigns a tentative distance value to every node. The starting node is set to 0, and all other nodes are set to infinity. A priority queue is used to keep track of the nodes with their tentative distances, prioritizing the node with the smallest one. While there are unexplored nodes, the node with the smallest tentative distance is selected from the priority queue and the tentative distances of all its neighbors through the current node are calculated, get compared with the current assigned value, and updated if the new distance is smaller. Once a node has been explored and its neighbors are updated, it is marked as visited to avoid redundant calculations. This process is repeated until the destination node is reached. The algorithm's output contains the list of nodes from the source to the destination, constructed by following the predecessor’s link from the destination node back to the source.

Silence Detector

The silence detector component participates in real-time translation functionality of the application when one to one communication is conducted. The purpose of this component in this workflow is to function as an event emitter, to inform the system when recordings should start and stop. This way the application ensures that neither redundant information will be sent for translation nor meaningful audio will be lost during the process.

To function properly, the silence detector needs to be configured with an amplitude threshold value below which the analyzed sound would be considered as silent and a time interval to determine the loop frequency. Through testing and validation, the silence threshold was set to 0.6 seconds and the loop time interval to 20 milliseconds On initialization the given audio stream is connected to an audio analyzer. At regular intervals, determined by the configuration

parameters, the audio data obtains the amplitude of the audio signal and calculates the average value. If the average value falls below the given threshold, the system considers that this is a silent interval. Consecutive silent intervals are tracked and if the accumulated time exceeds a specified duration, the system marks the overall state as silent. If the amplitude is above the threshold, the silent interval counter is reset, and the system returns to a non-silent state.

Every time a state change occurs, this change is propagated to the Audio Recorder as a signal to either start recording or stop and inference the audio.

Entity Component System

The software architecture of the application is based on an entity component system. Entity-Component-System (ECS) is an architectural pattern commonly used in game development to organize and manage the complexity of entities, their behavior, and their interactions. It's designed to improve modularity, reusability, and performance of the application.
The application's ECS architecture consists of three main components.

1. **Entity**: A general-purpose object in the 3D world. It doesn't have any inherent behavior or data associated with it. Instead, it serves as a container for components. Entities are represented as a unique number in the application. A basic example of an entity in the VR Conference is the Virtual Agent of the user.
2. **Component**: A modular, reusable piece of functionality or data that can be attached to an entity. Components define specific aspects of an entity's behavior or appearance. For example, the component "Agent" is attached to the Virtual Agent entity to store some references of other entities necessary for the functioning correctly, such as it's text panel.
3. **System**: A system is responsible for processing entities that have specific sets of components. Systems encapsulate the logic that operates on entities with particular component configurations. Each system focuses on a specific aspect of the application and operates independently. The Agent System contains all functionality of a virtual agent and every entity that has the Agent component attached to it, is processed by this system on the main loop of the application.

4.1.2.4 User Interface Implementation

The application introduces to the user multiple UI elements both for toggling available functionalities and to display output once available. Additional functionality is developed in a way that tasks are activated as automatically as possible, to avoid overstimulating the user with redundant information. However, when user requirements define that a mechanism should not be intrusive, the presence of a UI toggle is cannot be avoided.

User Panel

The user panel (Figure 33) is the main UI element of the VR conference application. It is a system bar that appears at the top of the user's POV (Point-of-View) whenever they move their head upwards and it functions as a control center, containing multiple toggle buttons together with some extra options not related to the scope of the application. From the user panel, it is possible to enable/disable the presence of the Virtual Agent in the 3D environment, the Map component, and the Language Panel, the functionalities of which are discussed below. When one of the above components is active in the virtual environment, the respective

toggle is highlighted with a blue ring, to indicate its status. Additionally, when a functionality is not available for a specific room the saturation of its toggle is lowered to state that this option is not provided. Extra indicating methods about the application status are implemented for the language selection. When users select their desired language, the corresponding toggle changes its appearance to inform them about their active selection by using language codes (IT, ES, EL, NL, DE, EN).

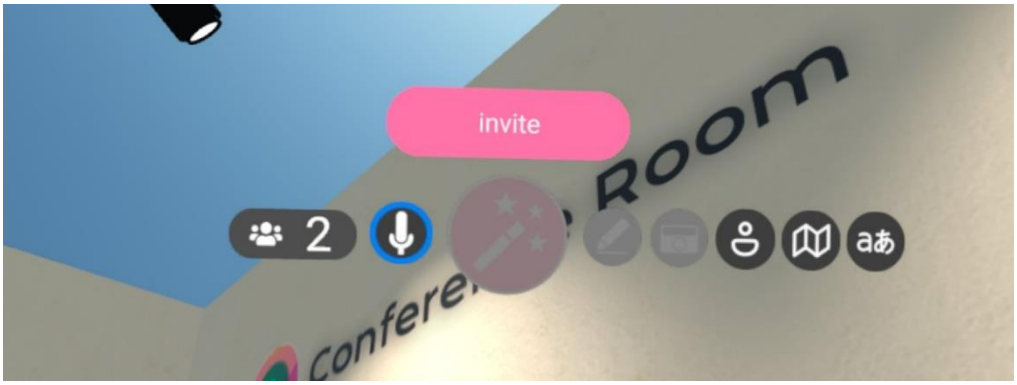

Figure 33: User Panel element.

Map Component

If enabled from the user panel, a 3D map component gets renders in the 3D scene, configured to change its position and orientation according to the user's. The Map panel (Figure 34) displays the top view of the scene's room on a smaller scale, along with informative text about the room's objects. To enhance the experience, a system that tracks the user's position relative to the space, maps it to an overlaying red dot on the map to help them orient. This tracking system is dynamic, and it gets updated in real time, so that the participants can see the impact of their movement on the map.

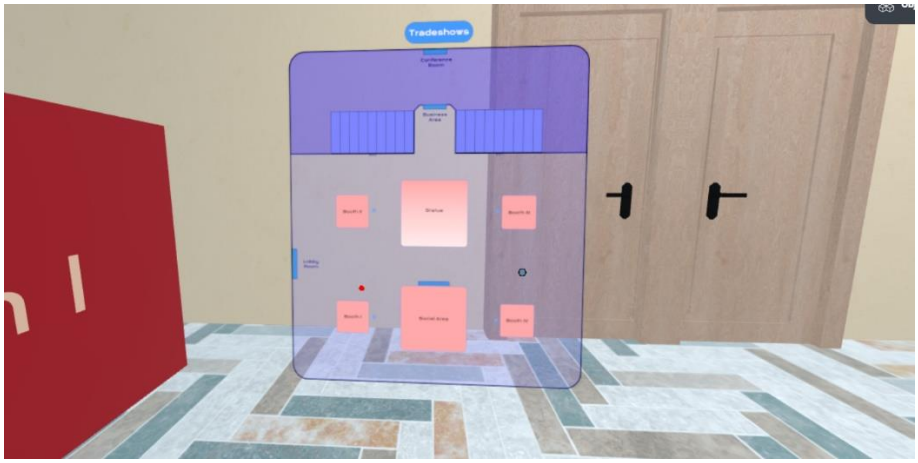

Figure 34: Map Component for the Trade Show Area.

Language Panel

Behaving in a similar way to the Map component, when selected from the User Panel, a Language Panel (Figure 35) that provides buttons for the users to select their language, is appended to the 3D scene. The selected language represents both their spoken language and the language of the subtitles. Each button of the panel contains an image of a flag to express its value and if one of them gets selected, it is highlighted with the same blue ring

effect. The panel closes automatically when a selection happens to make this process less complicated and quicker.

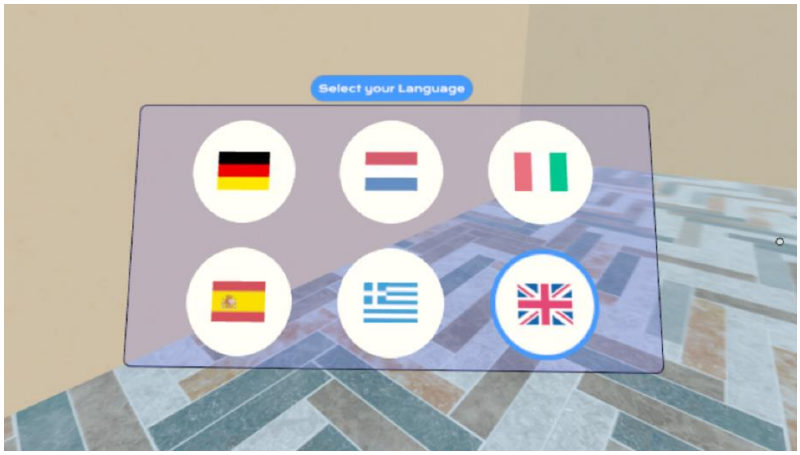

Figure 35: Language Panel with English language selected.

Behaving in a similar way to the Map component, when selected from the User Panel, a Language Panel that provides buttons for the users to select their language, is appended to the 3D scene. The selected language represents both their spoken language and the language of the subtitles. Each button of the panel contains an image of a flag to express its value and if one of them gets selected, it is highlighted with the same blue ring effect. The panel closes automatically when a selection happens to make this process less complicated and quicker.

Translate Button

In rooms where the application permits one-to-one communication and translation, every participant has a predefined spatial border around them. The system computes the relative distance from one participant to all the others in order to detect when someone has crossed this border. If this happens, a button in front of every avatar inside this border with the word “Translate” appears in the 3D scene (Figure 36). Pressing this button will make it disappear, will activate the real time translation system and if all conditions are met, the translation will start. While a user is selected, the application continues to calculate the relative position of the remaining avatars inside the scene, so the option to change target becomes trivial and not time-consuming.

Figure 36: Translate Button element.

Translation & Virtual Agent Text Panel

If a target is chosen and both communicating participants have selected their preferred languages, a text panel gets rendered in front of the target in the same position the Translate Button was rendered previously. The value of this panel (Figure 37) contains initially a phrase in the user's language, informing them that the translation output will appear on this element. If any of the two parties deactivates their languages the translation panel disappears automatically, until a new valid change occurs.

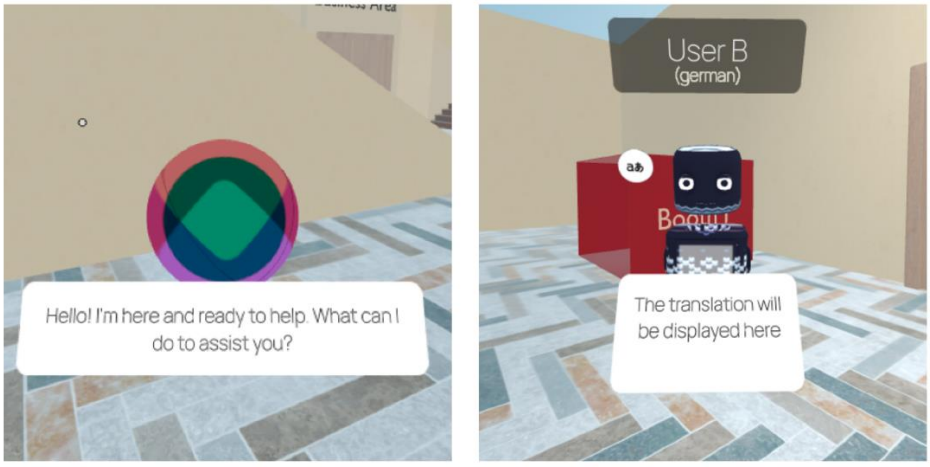

Figure 37: Virtual Agent Panel (Left), Translation Panel (Right).

The same configuration applies to the text panel of the Virtual Agent (Figure 37), but in that case the panel stays always visible, until the user disables the presence of the Virtual Agent. The information that gets displayed on this panel contains greeting messages, responses of the dialogue agent module and potential error messages.

Translate Target Indicator

When the user chooses a translation target, a small translation button (Figure 38), containing the translation symbol, will also appear on top of their head. This button helps the user detect their target inside the room and distinguish them from others. This could be particularly helpful in cases where many avatars are inside the same room or when the translation panel is not

visible due to missing source or target language. When hovering over this button, the symbol transforms into an “X” to declare that by pressing it, the selected target option will get cleared.

Figure 38: Indicator Button (Left), Hovered Indicator Button (Right).

Loading animation

The last UI element is a panel of the Virtual Agent to keep the participant updated about their request status (Figure 39). While they are phrasing their message to their assigned Agent, five dots with a breathing animation appear, indicating that their question is indeed being recorded. When the recording stops, while the voice message is being analyzed and until an output is sent back to the user, the same dots change to a loading animation, confirming that their request has been successfully sent and it is being processed.

Figure 39: Loading animation.

The last UI element is a panel of the Virtual Agent to keep the participant updated about their request status. While they are phrasing their message to their assigned Agent, five dots with a breathing animation appear, indicating that their question is indeed being recorded. When the recording stops, while the voice message is being analyzed and until an output is sent back to the user, the same dots change to a loading animation, confirming that their request has been successfully sent and it is being processed.

4.1.2.5 Summary of Achieved User Requirements

In summary, we have successfully met 25 of the 49 user requirements for the VR Conference application, including 19 of the 32 high-priority and 5 of the 10 medium-priority. Users are

represented as virtual avatars; each assigned a dedicated virtual agent with a cartoonish appearance. These agents not only provide welcome greetings but also interact with users on demand. For navigation within the XR environment, a virtual map is available, along with clear visual cues such as arrows, guiding users to their destinations. Furthermore, the system provides real-time translation in five languages (DE, NL, IT, ES, EL). Those translations are available in textual format, and subtitles can be toggled on or off. Each speaker has uniquely assigned subtitles, ensuring a seamless and immersive experience.

4.2 Augmented Theatre

The Augmented Theatre application is intended to be used in AR-enhanced theatrical play to provide the audience with personalized, augmented reality audiovisual and textual content to cover accessibility and entertainment needs. According to the gathered user requirements, the application should deliver a stream of timely, translated subtitles to the end-user in the language of their choice (high priority), a set of virtual effects (VFX) in specific moments of the play triggered by verbal cues as determined by the theatrical director (high priority), and lastly, additional background information about the play upon user request (low priority) (Figure 40). Given the capabilities of the VOXReality services, it has been also examined to incorporate an accessibility feature of stage action (scene) description, which is treated as optional ("could-have") in terms of development priorities.

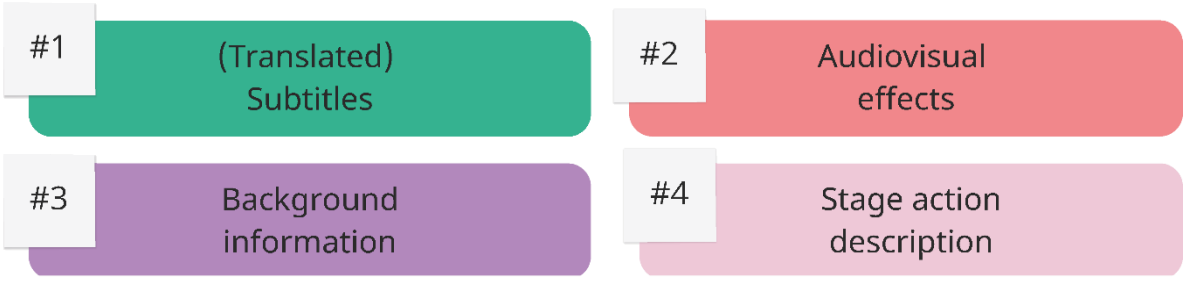

Figure 40: AR Theatre app features, indexed by priority, based on user requirements.

The application should target augmented reality glasses which allow the user to view the theatrical stage with an overlay of digital 3D content. Furthermore, the application should operate with minimal disturbance in terms of head and hand movement, without voice commands and in low light conditions, therefore any input to the application should be made using a dedicated controller. For those reasons, the Magic Leap 2[6] device was chosen.

The application should offer a personalized experience that is in sync with the theatrical play happening on stage. Therefore, a server-client system should be implemented to ensure that the experience can be centrally controlled, quality assured, and synced across all members of the audience. Consequently, to achieve the best possible audio quality, the server should be responsible for receiving the audio from the actor's speech through dedicated microphones, generating a transcript using the respective VOXReality model and streaming the generated transcription to all clients. Furthermore, the server should be responsible for receiving a visual feed of the stage from a dedicated camera, generating a verbal description of the scene using the respective VOXReality model and streaming the generated description to all clients. In turn, each client should be able to request a translation of the received transcription in the user's preferred language from the respective VOXReality model and

[6] https://www.magicleap.com/magic-leap-2

should use the scene description as contextual information for the translation request. Both the received transcription and the received scene description should be used in logic running on the client to manage the display of VFX and to respond to user requests for background information about the play.

Finally, an important factor for the design of the AR Theatre application is the fact that it targets mainly non-experienced in augmented reality theatre goers. As a result, it is important to design and implement an introduction step that can cover both a familiarization process with the overall AR technology and instructions on how to use the application itself. As part of this, the introduction and the user customization process are designed to take place before the start of the performance and with a staggered approach to accommodate a slow learning curve.

4.2.1 System Architecture and Design

For the AR Theatre use case, a server-client system with a local Wi-Fi network connection was chosen. The server runs on a GPU-enabled Windows laptop and the clients operate on Magic Leap 2 devices which run on an Android-based OS. The detailed reasons for the selection of the server were 1) to centralize the audiovisual input from the scene (microphone and video input) in order to safeguard input media quality, 2) to simultaneously distribute the output to the end-user devices in order to assure synchronicity in the audience and 3) to offset the demanding computational needs to a more powerful device in order to avoid battery and overheating issues on the XR devices. To accommodate these needs, the server receives audio and video input from the stage using appropriately placed camera and microphone devices with a wired connection for stability and speed. A USB-connected camera with a view to the physical stage is attached to the laptop, and currently the microphone on the camera is used to receive audio. In future stages, the theatre's microphone setup will be used instead. The server and the clients are connected to the same local Wi-Fi network and communicate using a WebSocket protocol for frequent and fast communication.

In addition to the server, the GPU-enabled Windows laptop also runs locally the two required models in containers using Docker Desktop (Speech Translation model code found here and Vision Language/GPT2 model code found here). Running the models locally is preferred to a remote cloud call, because the system can operate with high security standards, isolated -and thus stable- network conditions and minimal communication latency, all of which are especially important during the theatrical performance. The server code communicates with the models running on the same device by making RESTful API calls to the local host which ensures the highest possible response speed and contributes to the critical goal of overall low latency in the provision of the subtitles. The downside of the co-existence of both the neural network models and the WebSocket server on the same hardware device is that this solution has a higher risk distribution and computational strain to the device. To mitigate this, an improved resilience and recovery strategy is being designed iteratively and is informed by continuous testing. Currently, part of the mitigation plan is to separate the WebSocket traffic from the API call traffic, to prioritize direct API calls in the clients as much as possible, and to implement different fallback plans on the clients in case any of the two endpoints fail to respond (e.g., due to an application crash or OS crash). Design choices informed by this strategy influenced the communication architecture described below.

The clients are receiving two main categories of messages from the server using the WebSocket protocol: 1) information about the current ongoings of the theatrical play, which is divided into a) a text transcript of the current actor's speech produced by the server using the ASR module and the microphone feed, and b) a text description of the physical stage layout produced by the server using the VL-GPT2 module and the camera feed and 2) commands on which features of the application to enable in the clients (e.g. 2d subs/3d subs/etc..) which is in accordance with the performance delivery and evaluation plan.

The clients are in turn sending two main categories of messages back to the server device IP: 1) their status at every frame via the WebSocket, so that the facilitator can monitor the stability of the system (e.g. detect a client device crash or any other lag or non-compliance), and 2) a translation request in the user's preferred language directly to the server's IP and exposed port using the RESTful API. The client application also has a fallback cloud server address in case the local IP/port for the API calls is not replying as part of the system resilience improvement. It should be noted that although the neural network model can also provide a direct audio-to-text translation service, this option is not preferred for the AR Theatre use case since the combination of a single server-located transcription call and multiple client-located translation calls achieves the minimum possible latency when scaling up the solution for a larger audience.

Logic related to the display of VFX and other background information about the play runs independently in the clients. The reason for this is twofold: first, since these are low intensity computations that the XR device can perform without strain, having them run on the client follows the risk distribution principle. Secondly, this further allows for a diversification of the end-user experience, especially with regards to the VFX or the user request for background information. Although this diversification is not currently in the user requirements, this architecture can cover future requirements where e.g., the theatrical partner may wish to provide different content based on the declared user profile, like the age group, or other preference settings. This kind of relies logic should typically run on an independent client-level.

Concludingly and in high level summary (Figure 41), the AR Theatre use case incorporates the Neural Machine Translation service (NMT), the Automated Speech Recognition service (ASR) and the Vision Language – captioning (VL) service using the above-mentioned edge server deployment architecture.

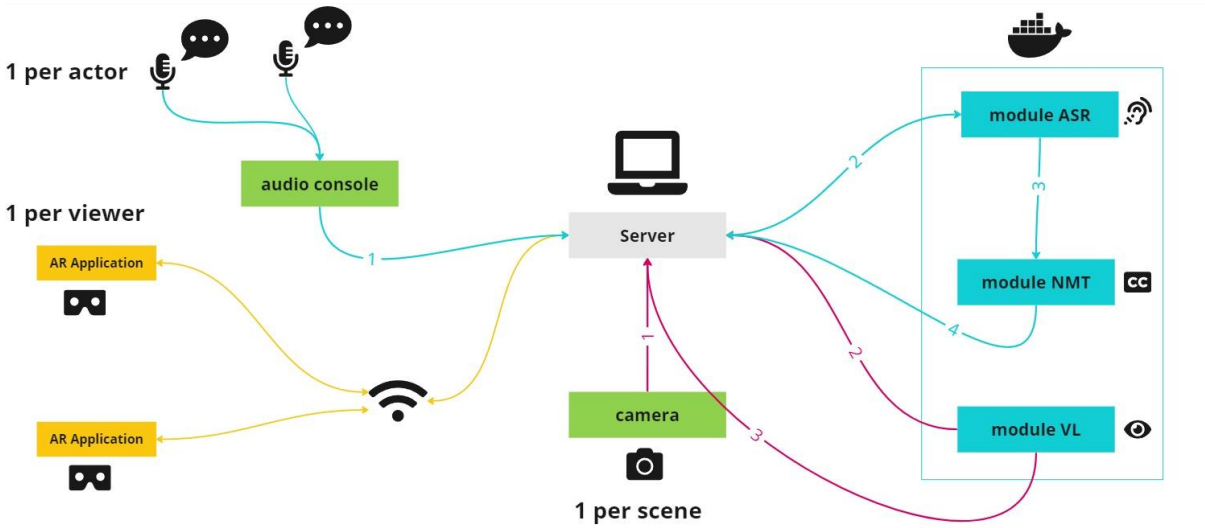

Figure 41: Communication flow between client, server, devices and docker containers.

4.2.1.1 3D Models and Scenes Design

The AR Theatre application contains two categories of visual assets: 1) assets that create the Augmented Reality visual effects to be used during the AR play, which are termed as AR VFX, and 2) assets that digitally simulate the physical elements of the theatre, which are typically not accessible or rarely accessible during development (i.e. theatrical stage and human actors), which are termed digital simulation.

The digital simulation assets will evidently not be used during the actual performance and will be substituted by the physical elements during the performance itself. Nevertheless, the digital simulation is being used in the introduction and tutorial material, so that the introduction step for the audience can take place with higher flexibility (e.g., even before entering the theatrical hall). The digital simulation is also being used as a communication medium to discuss and plan the physical characteristics of the play with the theatrical partners in advance.

As a sidenote, the simulated stage has smaller dimensions than the theatre stage that is currently assigned for the actual performance to better adapt to testing in (office) areas which are typically smaller to theatrical venues. Furthermore, the stage design and the placement of the actors is indicative and subject to change in ways that do not impact the application.

The AR VFX category should be determined and designed by the theatrical partner and the assigned director. For development purposes, a placeholder VFX has been created to illustrate the capabilities of the medium. The placeholder AR VFX helps to experientially transfer knowledge and know-how from the development team to the theatrical team in terms of technical guidelines and expressive capabilities. In technical regards, the developed VFX highlights the need for 3D models with an ideal mesh size, shaders with low computational needs, materials with no reflective or transparent properties, and low-quantity particle systems. In addition, the rendering framework of the application has been designed without real-time lights and shadow generation, as well as no post-processing, which also indirectly affects the VFX.

Expressively, the placeholder VFX that have been developed showcase different approaches to AR content, for example: 1) an AR visual extension to the physical stage using static models for objects with or without animation (e.g. rocks), 2) AR narrative agents or elements using dynamic models with movement animations (e.g. horses running, standing, grazing) and/or environmental effects using code-based animations (e.g. waters raising or boiling) and/or audio effects with spatialized audio sources (e.g. a thunder audio effect from far away, or a crash audio effect from nearby), and finally, 3) an AR actor (e.g. Goddess Artemis), which is an AR avatar that can appear on stage next to the physical actors and performs similar functions using animations and audio sources. In the implemented example, the AR actor performs a series of visual transformations (from deer form to human form) and is voiced by a spatialized audio source using pre-recorded theatrical lines.

Concludingly, it should be noted that all the placeholder VFX are indicative and subject to change both in terms of concept and appearance, according to the instructions of the theatrical partner. The simulation scenes are presented in Figure 42-46

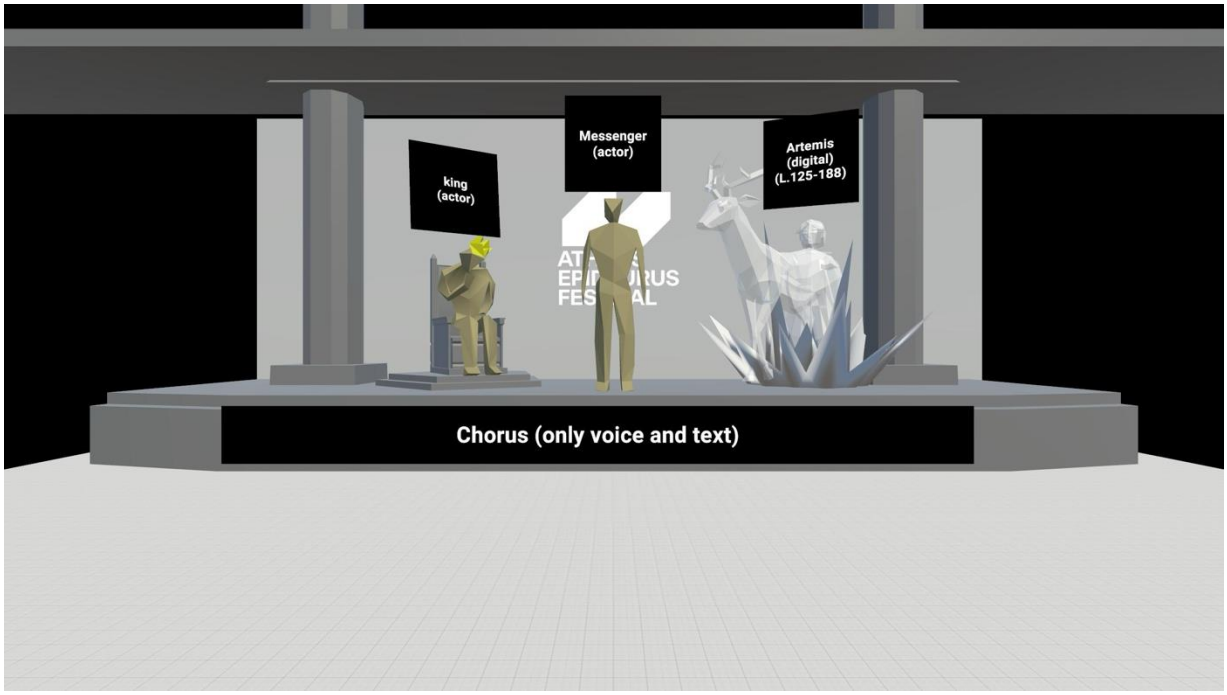

Figure 42: Digital stage simulation with avatars for 2 physical actors (King, Messenger) and a VFX actor (Artemis) - front view

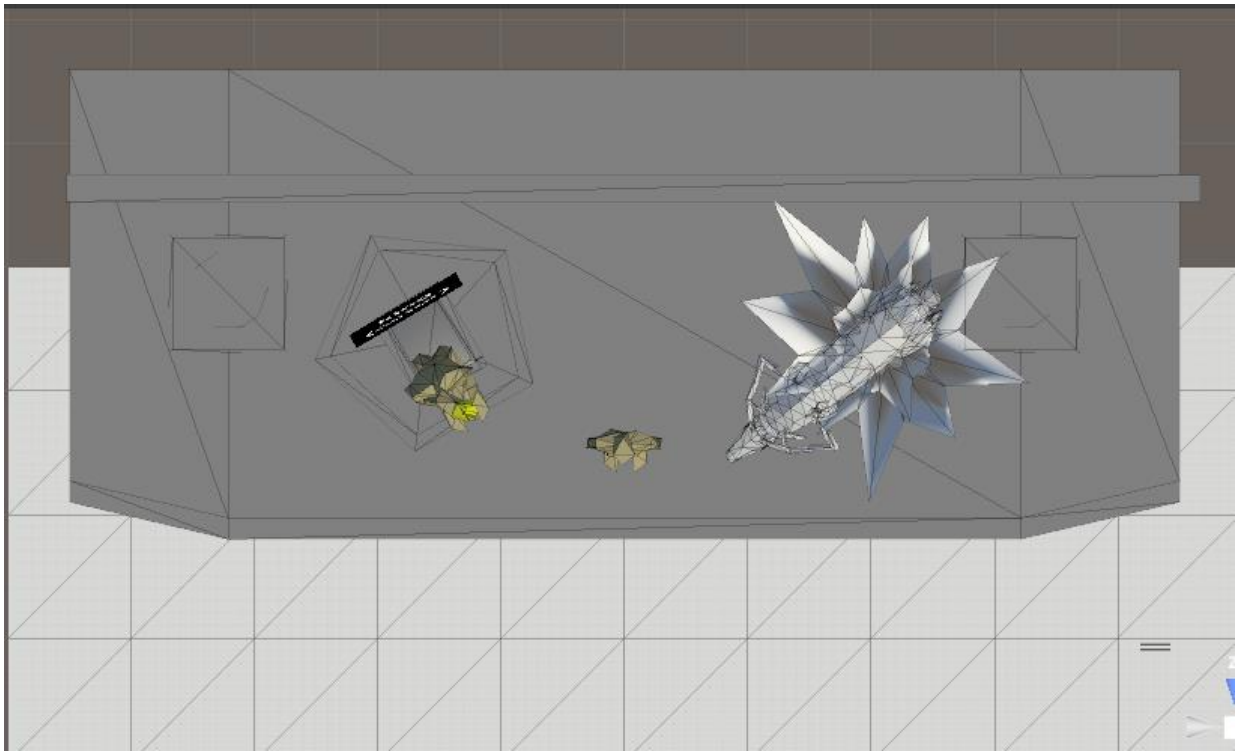

Figure 43: Digital stage simulation with avatars for 2 physical actors and a VFX actor – topdown view.

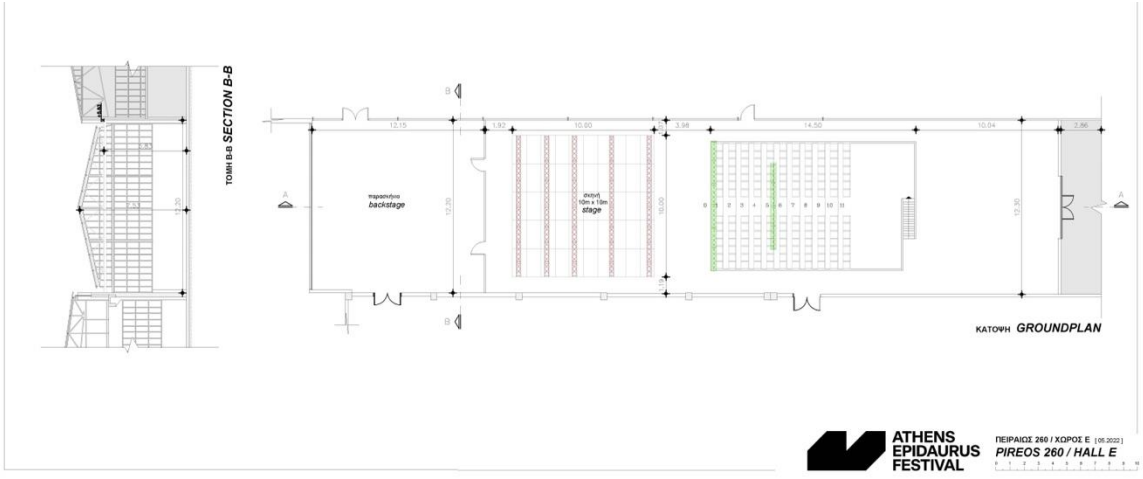

Figure 44: Layout of the theatrical hall and stage chosen for the performance.

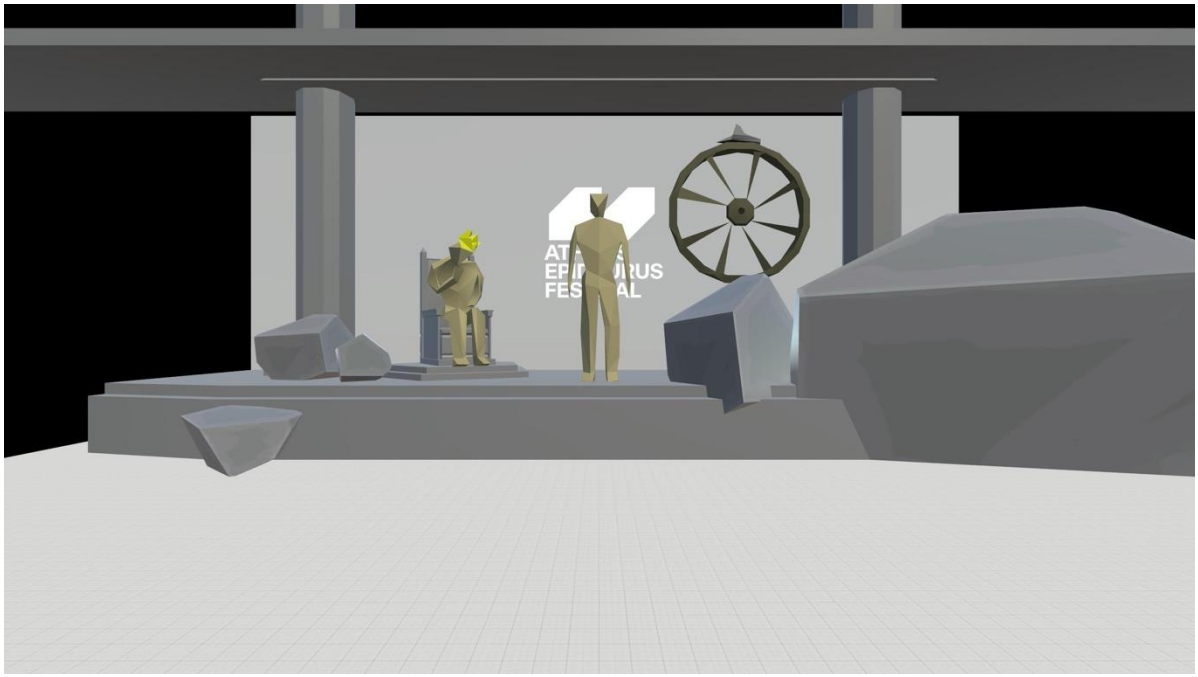

Figure 45: Examples of AR VFX, extension to the physical stage with static and animated objects.

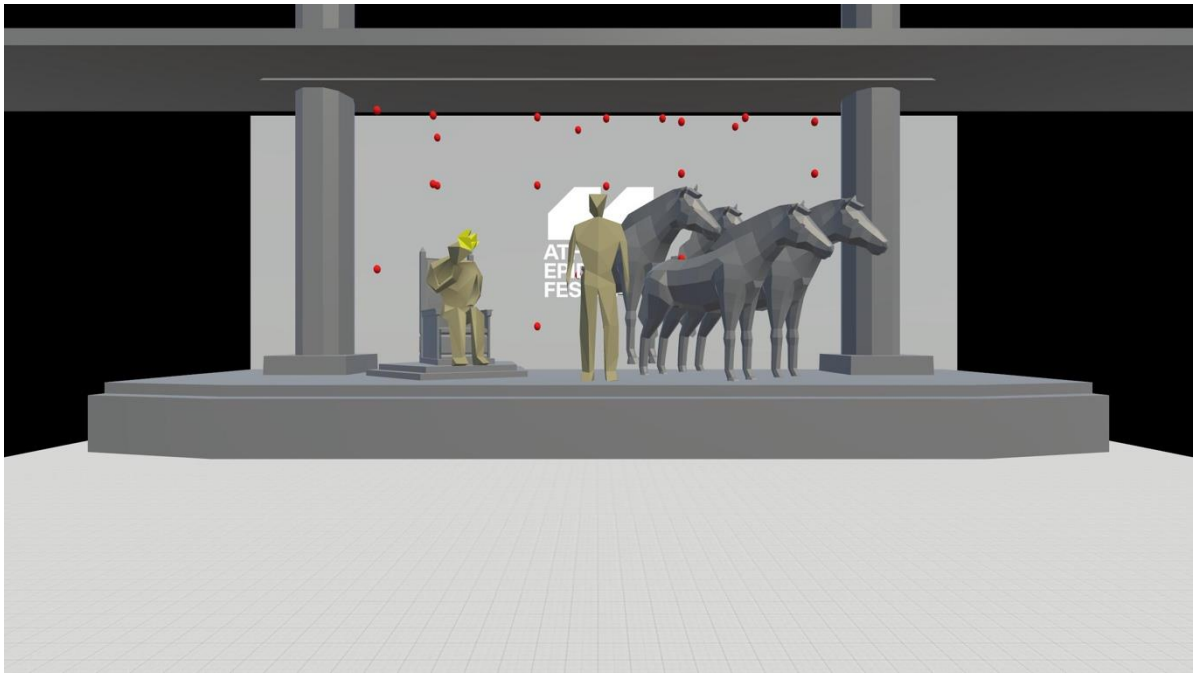

Figure 46: Examples of AR VFX, narrative elements, environmental effects and animated models, as described in the actor's narration.

4.2.1.2 Application Workflow Diagram

The application supports a linear flow across three scenes: from the Home scene to the Introduction scene and finally to the Performance scene which in turn is comprised of four (4) sequential phases: 2D subtitles, 3D subtitles, VFX with 2D subtitles, and User combination phase, where the user can choose to experience any of the above features again or to make new features combinations of their own (e.g. to choose only VFX with no subtitles, or VFX with

3D subtitles). The Home and Introduction scenes are intended to be completed before the theatrical performance starts. The progression from Home to Introduction scene is controlled by the end-user through the user interface (e.g., with instructions to press “Ready” to proceed). The performance scene is intended to start once the theatrical action on the stage begins. The start of the Performance scene and the transition from phase to phase is remote-controlled by the server through a dedicated user interface and a human facilitator. This combination is designed so that users can go through the independent steps of the Home and Introduction scene in their own pace but experience the phases of the Performance scene in sync with each other with the mediation of the human facilitator through the server remote control feature.

In the Home scene, the user is requested to select their language of preference and to proceed to the Introduction scene. Access to the Performance scene is locked until the individual user has completed the Introduction tutorial and/or the facilitator unlocks it through a remote-control command. Figure 47 presents from left to right the language selection screen, scene selection with locked access to the Performance scene (mandatory completion of Introduction scene first) and the scene selection with unlocked access to the Performance scene.

Figure 47: Home scene screenshots.

In the Introductions scene, the content is split into two sections: a short section where the user is shown each of the available input methods of the XR device and is required to try them out (5 screens - Figure 48), and a longer section where the user is shown the user interface for each of the four phases of the Performance and is required to perform any desired customizations in advance of the performance (20 screens - Figure 49). The user customizations include setting the font size, the z-distance (depth) of the user interface panel from the user, the background contrast of text, and the position of the text to accommodate better legibility based on individual preferences and visual acuity. The user customization settings, as determined during the Introduction scene remain active in the Performance scene, so that the user may only need to make minor adaptations during the Performance and remain immersed in the play without the need for further interactions.

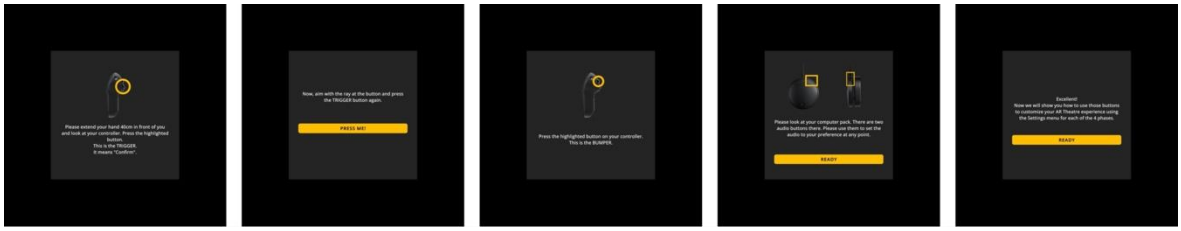

Figure 48: Introduction scene – section A: instructions for input methods.

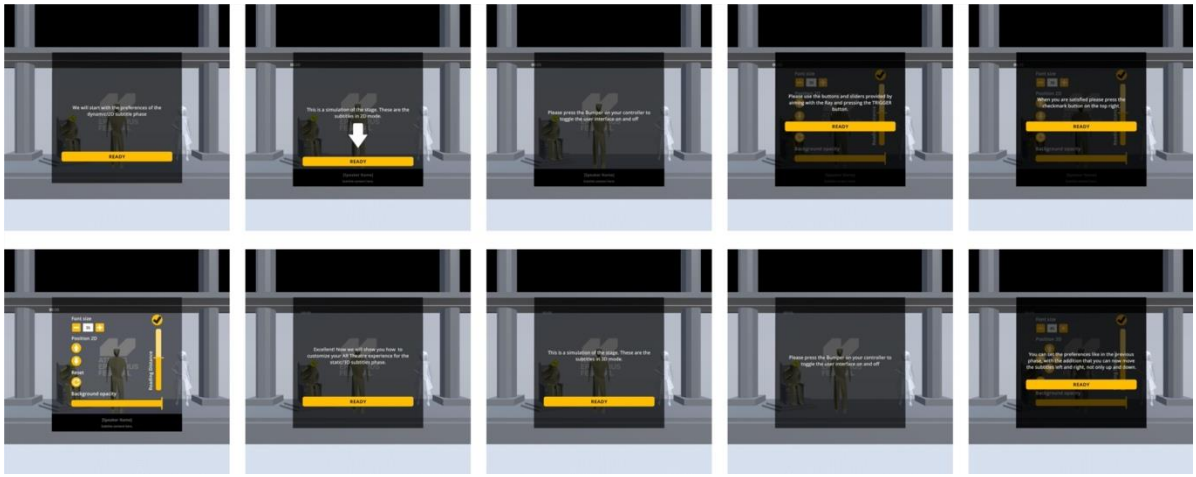

Figure 49: Introduction scene – section B excerpt: instructions for user interface.

The Performance scene (Figure 51) is compartmentalized in four phases to incrementally present the AR features to the users and allow for improved comprehension and isolated examination of each feature. The user combination phase is intended to allow users to experiment freely and create a more informed opinion about their preferences. In the Performance scene, the transition between phases is automated and remote-controlled by the human facilitator using the server user interface (Figure 50). The transition is indicated to the viewer by an informative panel. A small timer with a countdown indicates the duration of each phase to the user. The user may at any point use the controller to bring up the user interface and make adaptations to their settings (e.g. change language, change font size, etc.).

Figure 50: Transition panels.

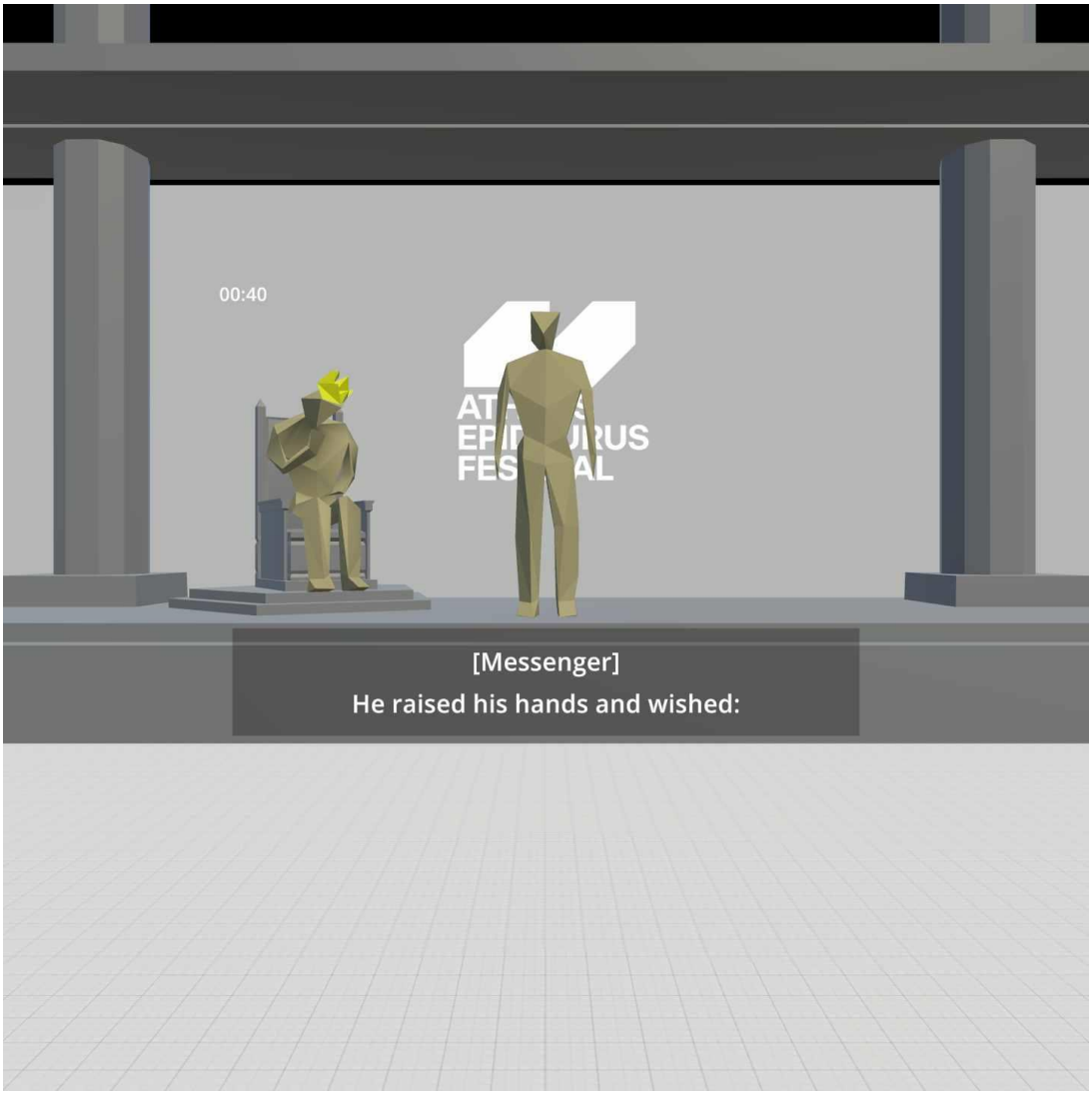

Figure 51: Performance scene sample with user customized 2D subtitles.

When the performance is over, the user is presented with an informative panel and the application closes automatically. In the background, the client application creates a traffic log of the content that it has received and generated (transcript, scene descriptions and translations) with a separate file per service (one transcript log, one translation log and one scene description log). In the next version of the application, the application will also generate a user log with the settings customization, that is generated once after the Introduction scene is completed and updated once again at the end of the performance.

Finally, the server application (Figure 52) currently allows the facilitator to choose an IP/port for the WebSocket connection (with the added feature of locating the IP of the device in the local network and assigning it automatically as the server IP) and proceeds to display a list of all connected clients. The server application can remote controls all clients simultaneously with a set of commands for functionalities such as localization, scene selection, phase selection, etc. This allows for improved testing, coordinating and facilitating the experience across multiple devices. In the next version, the server will be expanded to allow for individual control of each client using a separate port per client, as part of the crash recovery plan. Individual control e.g., is useful in case a client crashes and needs to be brought up to speed

with the rest of the clients and the play. The server should also be able to remote-control a client to load the previously stored user settings to recover from a crash without loss of information.

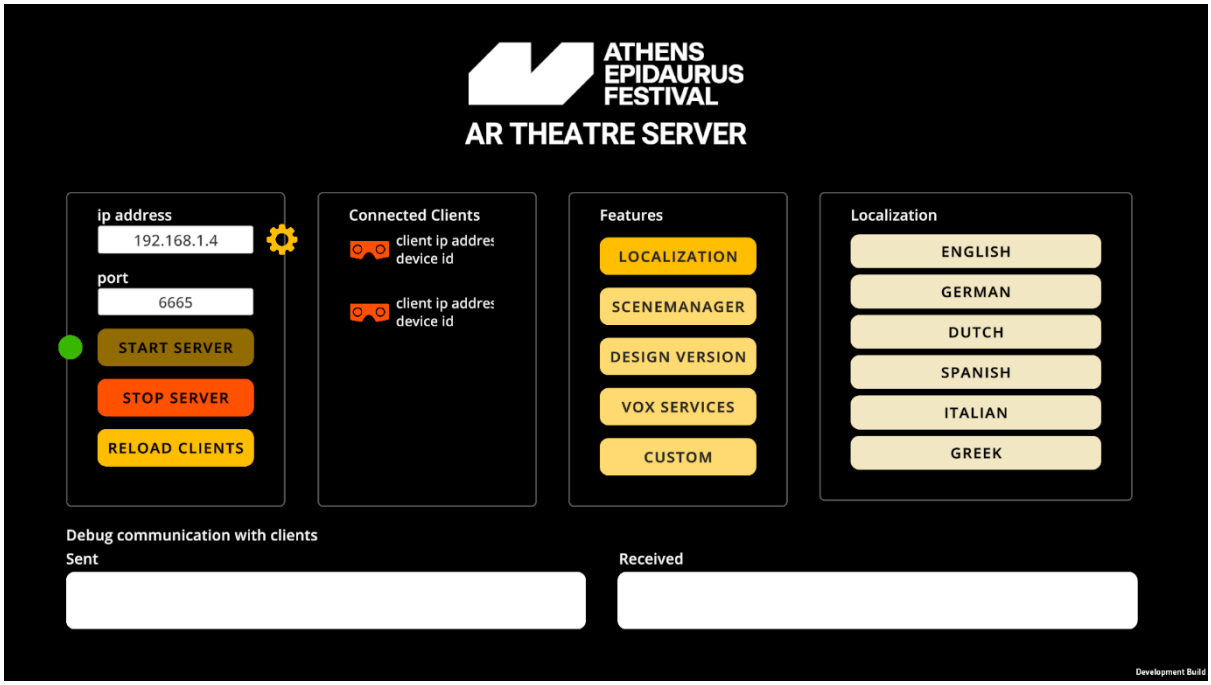

Figure 52: Server user interface for remote control of the XR applications.

4.2.2 Implementation Details

The development of the AR Theatre application follows an iterative approach to adapt to the unique challenges posed by the nature of the use case. In more detail, the first version of the application was implemented running independently on the XR client using the digital simulated stage as visual aid and voice recordings for the actor speech as audio aid. Due to the lack of requiring a server, this approach was favored to provide increased mobility and ease of use for testing with a single XR device and for showcasing and communicating purposes between the various stakeholders during the first stages of the development process. The first version also targeted the high priority user requirements, most notably the subtitles and the VFX.

The second version of the application was implemented using a server-client architecture as described above but maintaining the digital simulation of the stage and the voice recordings as a measure to facilitate user testing independently of the limited access to the theatrical space and/or actor availability. This version covers most of the high and medium user requirements and does not address the low priority ones (e.g. the background information provision).

The third and final version of the application removes the digital simulation and makes use directly of the spatial mapping and tracking of the XR device to project the augmented reality content in predefined locations in the physical space, as well as the live feed from a

microphone input source instead of recordings. This version will cover all the user requirements.

The first version of the application was completed in November 2023. The second version of the application has been completed in February 2024 and at the time of the report is being tested and finetuned. The third version of the application is due to be delivered by the end of September 2024 with a view to a larger-scale pilot testing in spring of 2025.

4.2.2.1 Development Environment Setup

Both server and client applications have been developed using the Unity Game Engine (version 2022) with the extension of appropriate packages, most notably the Unity XR Management plugin and the respective Magic Leap XR Plugin, as distributed through the Unity package manager, the Magic Leap SDK (com.magicleap.unitysdk) and the Magic Leap Setup Tool (com.magicleap.setuptool) as provided through open repositories by Magic Leap. All the relevant dependencies have also been installed. Visual Studio 2022 is the selected IDE for all local code development and debugging. A private repository in GitHub has been used as the version control system for the development process. The development process has been tracked and managed with the help of a GitHub Project page (kanban style tool) and a Miro Board has been created for wireframing, application flow diagramming and user feedback gathering. Finally, development video footage and snapshots from within the Unity Editor have been captured using the Unity Recorder package.

In addition, Android Debug Bridge (ADB) tools[7] have been used for starting, stopping, installing, patching and debugging the applications on the Magic Leap 2, and for other data management tasks, like transferring files. Supplementary, the Magic Leap 2 Hub software[8] has been used for maintaining and communicating with the Magic Leap 2, namely updating the development SDK, updating the OS in the devices, streaming from the devices in wired and wireless mode.

Streaming is also important in this use case for monitoring user's actions during testing, and for recording footage for communicating and documentation purposes. The Windows "Connect to a Wireless Display" extra feature[9] has been installed in the laptop and is being used for wireless casting from the Magic Leap 2 to a monitor. OBS Studio has been used for recording the monitor feed[10]. Finally, the built-in photo/video recording function of the Magic Leap 2, which can be triggered by voice commands, has also been used in cases where the application is not occupying the microphone feed of the device.

4.2.2.2 3D Models and Scene Creation

As detailed in the 5.2.1.1, the 3D models for the AR Theatre use case are currently placeholders for development and communication purposes. For this reason, minimum required effort has been invested in their production and a high abstraction, low polygon approach with no textures has been adopted. The digital simulation assets, i.e. the 3D models

[7] https://developer.android.com/tools/adb
[8] https://ml2-developer.magicleap.com/downloads
[9] https://support.microsoft.com/en-us/windows/screen-mirroring-and-projecting-to-your-pc-5af9f371-c704-1c7f-8f0d-fa607551d09c
[10] https://obsproject.com/

for the stage and the actors have been created internally using Blender, which is a free and open-source software for 3D modelling. A subtle animation has been designed and assigned to the digital actors to offer a sense of natural movement. The animation is being triggered when the respective digital actor speaks according to the transcript. The placeholder assets for the VFX have been created locally in their majority, but in this case, some assets were also sourced from third parties. In specific, the rigged animal models (horses, bull) originate from a commercial source[11] and the stone models with their textured material were sourced from a 3D model hub and distributed with a creative commons license from their creator[12].

4.2.2.3 Core Algorithms and Techniques

The full pipeline for this application involves the following parallel routines:

Subtitle routine

The steps of this routine are:

1) Server-connected microphone → continuous audio feed → API call to the continuous transcription algorithm of the ASR model → WebSocket message to clients with timestamped transcription result as JSON data.
   It should be noted that for the first and second version of the application, the available transcription algorithm (/transcribe_audio_files method) receives audio recordings instead of an audio stream. To amend for this fact, the current application can perform a microphone recording with a customizable recording duration to simulate the behavior of a stream-based approach. To facilitate testing and development given the absence of actors, synthetic speech recordings of the theatrical text have been produced with each audio file (.wav) containing a single line of the theatrical lyrics (average duration of each file 2-3 seconds, 180 files). Even more, to better simulate the actors, selected lines (~100) have also been recorded by human agents. Therefore, the application during development uses the playback of the pre-generated audio files instead of a live microphone recording as audio data for the transcription step. Finally, an optional addition at this stage is to receive separate microphone recordings per actor/character, so that the generated transcripts can be easily tagged with the character's name.
2) Client message receipt → JSON deserialization of message → addition of transcription to backlog of received messages → client callback with translation request of the latest transcript in the user selected language, using the up to two previous transcript lines and any available scene description as context (/contextual_translate_text). If this method cannot produce adequate quality results for the chosen theatrical text (to be determined after further testing and evaluation), then the alternative is to manually produce a custom override dictionary for selected erroneous terms, use the "/upload_terminology" method to provide the data to the model and the "contextual_terminology_translate_text" during runtime instead of the /contextual_translate_text.
3) Upon successful resolution of the http request → display of subtitles in the currently enabled mode with the current user settings.

[11] https://assetstore.unity.com/packages/3d/characters/animals/low-poly-animated-animals-93089
[12] https://sketchfab.com/3d-models/stylized-lowpoly-rock-6e476441b5614231bf4e8de194c418d9

At this stage, the application also consults the "speaker name" which is tagged to the transcript and logged in the JSON data sent by the server. If the subtitles are in 2D mode, the application displays the speaker's name on top of the subtitles. If the subtitles are in 3D mode, the application enables the respective subtitle panel that is matched to the speaker's position. If the http request fails, then the application should attempt translation request again with fallback cloud server address. If this fails again, the application should display nothing. Both successful and unsuccessful requests should be added to the respective NMT data log.

The selected play for the application testing is in the Greek language and it has been observed that the transcription of Greek contains various spelling errors, which carry over to erroneous translations in the next step. Since the play is not improvised but a fully scripted play, an alternative approach is examined, where the audio recording is matched to an excerpt of the given script, instead of being transcribed live. This approach will be tested in the next version of the application. This approach also makes the tagging of the current speaker feasible with a single microphone channel since this information can exist in the script and be retrieved alongside the excerpt by the matching algorithm. Finally, it should be noted that this approach also increases human overview and control over the resulting translation which is a desired trait for this use case.

Vision language routine

This routine consists of the following steps:

1) Server-connected camera → image capture at configurable intervals (roughly every 2 seconds, similar to the duration of an average lyrics line recording) → Vision language captioning algorithm → WebSocket message to clients with timestamped scene description/captioning result as JSON data.
   The exact resolution of the camera image to be sent for captioning and the frequency of the captioning call should be determined by the difference between the vision language routine latency and the subtitle routine latency aiming for the best possible synchronization. This needs to be determined by further testing, since the distance of the camera to the scene is also an influential parameter.
2) Client message receipt → JSON deserialization → add to backlog of received messages → storage of latest message as contextual information for translation calls. Further testing which should take place in the actual conditions of the theatrical stage should highlight if any finetuning of the model is required to produce adequate quality results given the individual nature of the stage objects.

VFX routine

The VFX are triggered by keywords in the transcript (and not the translation) and/or the scene description. The scanning logic is performed in the clients and is tied to a callback of the WebSocket client upon receipt of a transcript or scene description message from the server. Currently implemented triggers are detecting an individual word or short phrase for the first or Nth time in the transcript (e.g., trigger VFX "a" when the phrase "πατρική κατάρα!" is heard for the first time). The spelling errors that the ASR transcription produces for the Greek language, make this triggering method unreliable and provide yet another reason to highlight the audio matching approach as more suitable to the needs of the use case. Another triggering method that is implemented is based on the index of a specific lyrics line (e.g., trigger VFX "a" when lyrics line 173 has been spoken). This is easily feasible in tightly controlled conditions like

during development where the digital simulation with an audio recording playback per lyrics line is used but can prove unreliable with a live microphone recording where the audio can be cut off at random intervals. This is another problem area which the audio matching approach can circumvent, since the lyrics index can be tagged in the script play and can be retrieved alongside the excerpt with any other desired metadata.

With respect to the rendering of the VFX, it should be noted that the VFX can be split into user space and world space VFX. User space VFX are positioned in a coordinate system relative to the user and are thus consistent in their generation and rendering whether during development with the digital simulation stage or whether during performance in the actual theatre. Examples of user space VFX are environmental effects like rain (i.e. effects without a clear origin point or shape), or objects whose position is sensibly related to the viewer as an individual (e.g. making a flower appear in front of every viewer). World space VFX needs to be positioned in a specific location in the actual physical space and that position needs to be identical across all client applications, regardless of the viewer's position. This requires the existence of spatial mapping and object anchoring logic in the SDK of the XR device, which the Magic Leap 2 device supports. This feature allows the AR VFX categories such as the AR expansion of the scene or the AR VFX actor presence. It follows that the type of VFX and their triggering keyword can be arbitrarily combined.

In the next version of the application the following changes will be implemented in order of priority:

1) The adoption of the audio matching algorithm should be implemented and the transition from an audio file-based to an audio stream-based approach should be completed.
2) The transition from the digital simulation stage to the physical stage should be completed, specifically the code for the generation of the world space VFX.
3) Further resilience and recovery features should be designed and implemented. Two designed features are pending for development: 1) the clients should declare their status at every frame to the server, so that the human facilitator can monitor and manage incidents. 2) the clients should save the user customizations in persistent memory and expose a remote-controlled (by the server) command for reloading them in case of an application crash.
4) The low priority user requirements should be designed and implemented, specifically the background information call.
5) The optional features could be designed and implemented, specifically the scene description call.

4.2.2.4 User Interface Implementation

The user interface is designed for a persona with minimal familiarity with AR technology and the user input modalities are required to be the least intrusive for a theatrical environment. Furthermore, the user interface should also aim to address high accessibility and inclusivity standards. These parameters are detailed in user requirements entries 1, 4, 5, 8, 14, 15 and 27 and are the main factors which informed the design decisions for the user interface.

As a result, the input methods for the application are restricted to two button presses for simplicity (one main and one secondary). The main method is the equivalent of the mouse

input in a traditional setup and is provided by a ray originating from the controller for pointing and a button press (Trigger) for the clicking. This method covers most user interactions. The secondary button press (Bumper) was intended to toggle the user interface on and off at any point for increased immersion in the play. All input methods are shown to the user and practiced in an interactive tutorial before the start of the performance. In addition, the application provided a digital rendering of the controller overlayed on the physical controller, which had the available buttons highlighted in yellow color for further guidance. It should be added that an additional button for accessing a hidden debug menu that is known only to the facilitator has also been implemented.

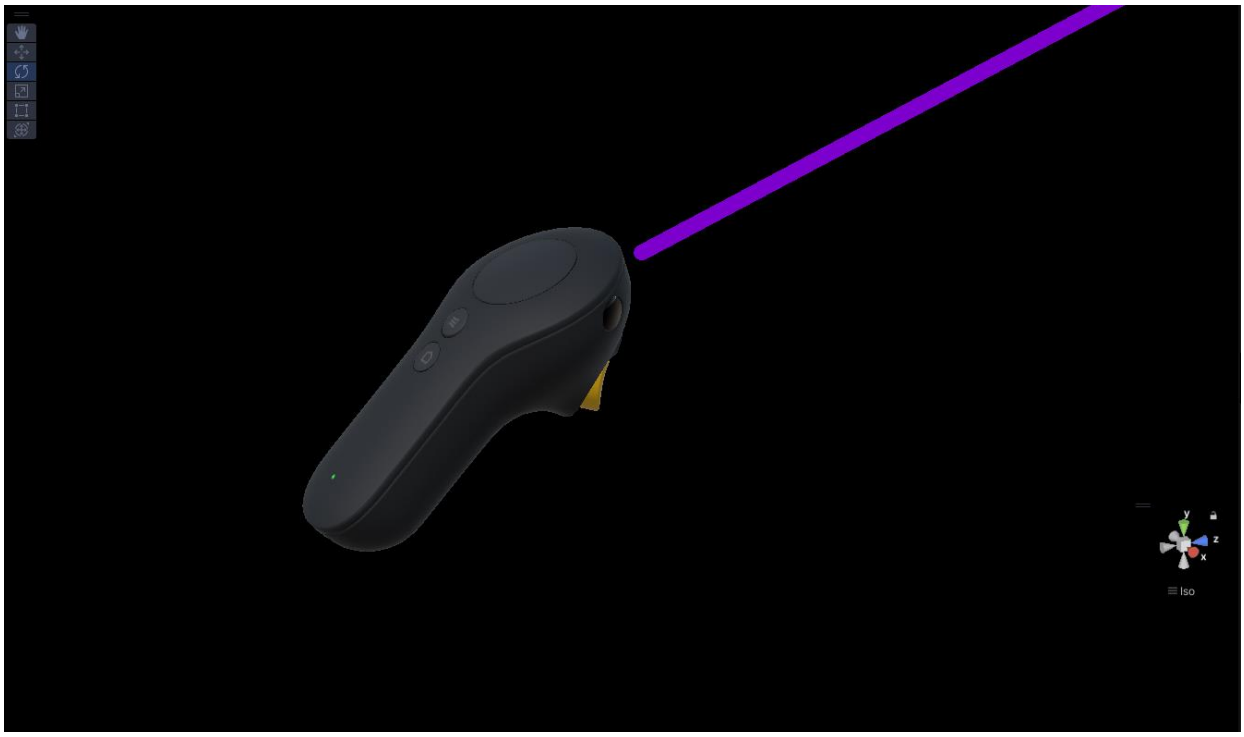

Figure 53: 3D model of the controller with the ray for interacting with the UI and the available button highlighted in yellow color.

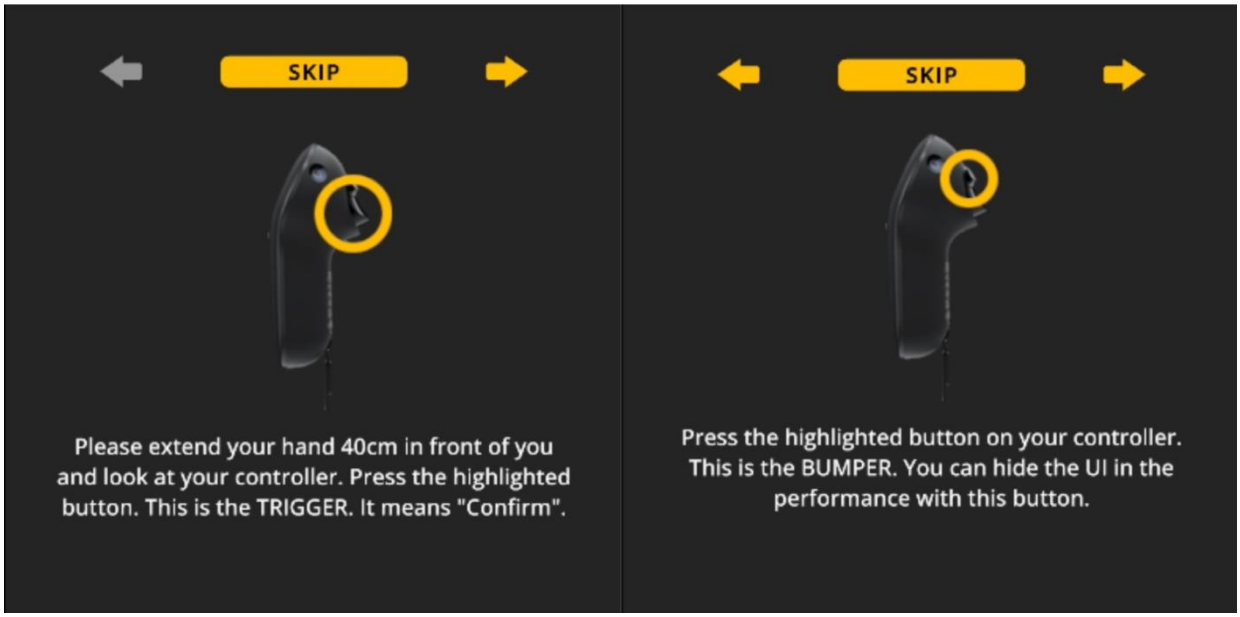

Figure 54: Tutorial covering all the input methods (2) with a hands-on approach.

More importantly, on the user interface design itself, accessibility standards and user customization options were implemented. Typical web-based accessibility standards dictate font size, contrast ratio, font type, line distance, and similar characteristics that relate to legibility. Given the fact that such standards do not exist yet for AR environments, default values for the design of the UI were influenced by the Web Content Accessibility Guidelines (WCAG) international standard and the user was provided with settings to adapt those values to their preference within a reasonable range. This was considered highly important since subtitle provision is one of the high priority user requirements for this use case and legibility is paramount for this feature. In specific, the user was allowed to edit the font size with +/- buttons for incremental changes, edit the distance of the entire panel with a slider, edit the opacity of the background of the subtitles with a slider, and edit the position of the subtitles incrementally either only vertically in the 2D mode, or vertically and horizontally in the 3D mode. Furthermore, usability reasons led design decisions such as the size and placement of the interactive elements (buttons, sliders). For this reason, use of typing, which would require an on-screen keyboard with overall low usability, was minimized. To further support usability, a visual aid for targeting (reticle) was added to the ray controller, the ray had two visual states (active/inactive) to denote if the user was targeting an interactive element or not, and most interactive elements had a visual response to hover events. Finally, recognizable iconography was chosen to render most buttons as self-explanatory and thus maintain a clean look to the UI with minimum label text.

Another important design decision for the UI was the implementation of the Immersive mode. As mentioned, the user could toggle the UI on/off with the help of a button on their controller. The default state of the application for the Performance scene was to start with the Immersive mode toggled on, therefore the user interface was hidden. Given the fact that the user was prompted to make any required adaptations during the Introduction phase, the user could potentially have little or no reasons to bring the UI up again and therefore, view most of the

performance with no interference. Having the UI as a provisional feature during the performance allowed the tradeoff of designing the UI elements with the highest usability level (large buttons, central positioning, bright colors), but also with the highest occlusion level to the theatrical stage.

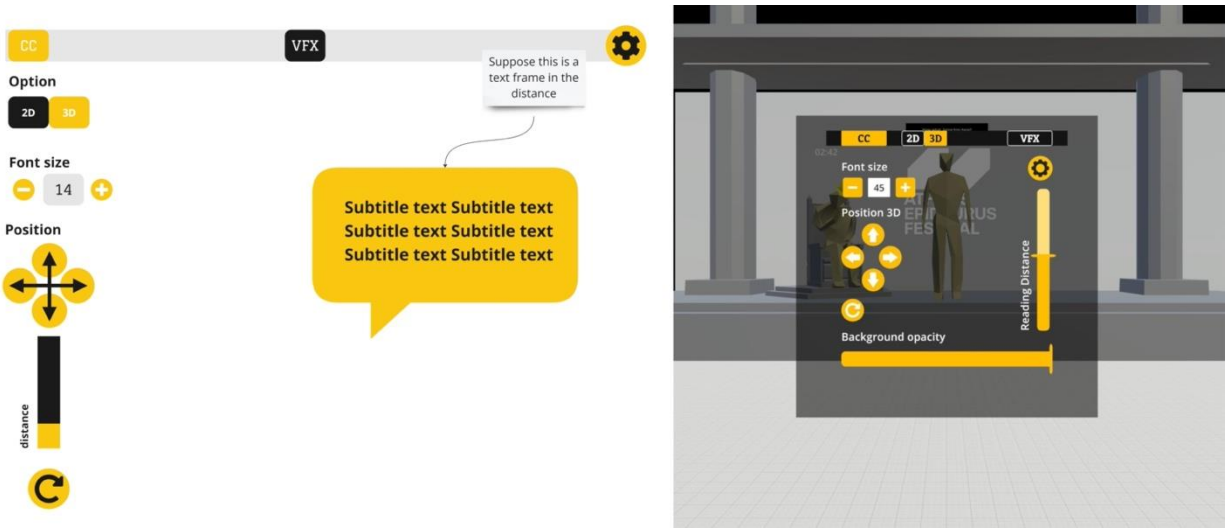

Figure 55: From wireframes on Miro for discussion and feedback to implementation in the application.

4.2.2.5 Summary of Achieved User Requirements

In summary, all the user requirements marked as High or Medium that relate to the XR application and one requirement marked as Low have been met in the second version of the application. The failed requirements marked as High or Medium relate to the lack of actors, director, stage or background information about the play as determined by a writer or director and will be targeted in the third version of the application. Therefore, the development of the XR application for the use case theatre has from a technical point incorporated most of the currently available resources at the time of writing.

4.3 Training Assistant

The use case focuses on the development of an augmented reality (AR) industrial assembly training application aimed at improving the training experience through the integration of VOXReality's automated speech recognition (ASR) model and the dialogue system. Traditional training methods often lack interactivity and adaptability, leading to suboptimal learning outcomes. By integrating artificial intelligence into the AR environment, this use case seeks to create a more engaging and effective training environment. Key features of the application include visualization and manipulation of 3D computer-aided-design (CAD) files (Figure 56) in AR environment, an interactive virtual training assistant with real-time performance monitoring, and a dynamic dialogue system powered by natural language processing (NLP) and speech-to-text capabilities.

The prime constituent of the training assistant technology is the application Hololight Space Assembly (shortly Assembly). The training process begins with the loading of a 3D CAD file, which serves as the basis for the assembly task. Trainees are required to accurately assemble constituent parts within the CAD object's frame. Assembly supports the loading and interaction

with previously created asset bundles that contain all the relevant scene information (CAD file(s), tools, other objects like table or shelves), scripts for interaction with the models, menu interaction and relevant algorithms. Performance metrics, such as time spent and incorrect insertions, are monitored in real-time based on predefined thresholds. Additionally, Assembly utilizes Hololight Stream, to remotely render the application from a laptop with a powerful GPU to AR smart glasses in order to bypass the device's rendering limitations. Combined with the remote rendering and streaming functionalities, the AR training application (server) is hosted on a laptop and streamed to the HoloLens 2[13] (client).

Figure 56: The Raptor engine CAD model which is used for the assembly task in the current use case.

Furthermore, the Assembly application is enhanced through the integration of ASR. ASR assists with the conversion of audio input to text (speech-to-text). The dialogue system comprises of a Natural Language Understanding (NLU) model, responsible for comprehending user input, and a Natural Language Generation (NLG) model, tasked with creating coherent and meaningful responses. This allows for natural verbal interaction with the virtual assistant, eliminating the need to learn predefined keywords. The forthcoming versions of the training assistant will provide contextually relevant information in response to trainee inquiries. Moreover, upon detecting the violation of the industrial assembly protocol, the training assistant intervenes by offering support and training aids through an interactive dialogue system. An additional feature of the training assistant includes the ability to present PDFs, relevant videos, or specific files upon user request. This multimedia capability expands the agent's functionalities, allowing it to provide diverse and contextually rich information in response to user inquiries.

4.3.1 System Architecture and Design

4.3.1.1 3D Models and Scenes Design/Creation

Traditional Training Challenges

Conventional paper-based training methods fail to enhance muscle memory due to their two-dimensional nature, resulting in the failure of comprehensive information transfer. Such training aids primarily in the memorization of assembly sequences at the most. Conversely, practical training utilizing authentic components facilitates the development of muscle memory and imparts all pertinent information to trainees, including spatial placement techniques and

[13] https://www.microsoft.com/en-us/hololens

micro-actions essential for assembly steps. However, this approach demands substantial resources, necessitating the provision of dedicated parts for training purposes, establishment of specific training venues within companies, and allocation of individualized training sessions by instructors, thereby incurring significant time costs. Moreover, on-site training is imperative as large-scale objects, such as automobile frames, cannot be feasibly transported to trainees' residences. External factors like global pandemics can disrupt training activities, halting onboarding progress once restrictions are lifted. To mitigate such limitations, an application for virtual training in XR environment with an interactive virtual training assistant was conceptualized.

Conceptualization of the Application

The application was conceptualized to,

- Closely simulate real-world usage scenarios, incorporating elements such as shelf configurations and object placements to enhance muscle memory, thereby enabling trainees to intuitively identify the next component to retrieve.
- Enable trainees to interact seamlessly with each component required for assembly.
- Integrate pre-assembly steps into the primary training with virtual table, shelve and toolkits as it aids in reinforcing muscle memory.
- Incorporate tool interaction to simulate realistic training experiences while circumventing hazardous accidents which are otherwise probable in the real-life scenario. This involves the interaction with indispensable tools such as a drill with audio feedback. Incorporating sound effects further enhances immersion by compensating for the absence of physical weight and gravity in virtual objects.
- Accurately define the intermediary steps such as visual inspections, precise object placements, and adherence to safety or logistical procedures via QR code scanning to ensure comprehensive training.

Inclusive Training Support Measures:

- Implementation of difficulty modes: The application features different difficulty levels (easy, medium, hard), with easy mode providing assembly preview, visual cues to disclose assembly sequence (Figure 57 and Figure 58), guiding lines and object locking for enhanced guidance. Conversely, hard mode challenges trainees by withholding visual aids and object locking.
- Various accuracy modes: Trainees can switch between simple and advanced snapping modes, catering to varying skill levels. Simple snapping facilitates easier attachment, while advanced snapping necessitates precise alignment for successful object placement. Simple snap is for trainers, as it is the easiest method of attachment. Advanced is more realistic as a nearly perfect alignment is necessary for the object to snap into place.
- Performance indicators: The application employs color-coded highlights to denote correct (green) and incorrect (red) object selections, aiding user comprehension. Additionally, visual cues and colour changes at sequence flags signify completion status, facilitating progress tracking.
- Integration of audio cues: Audio cues alert trainees when objects are correctly placed, further enhancing the training experience.
- Pre-assembly: The application enables the user to perform pre-assembly of the grabbable parts on the table / shelve before starting the actual assembly task.

Figure 57: Parts of the CAD model on the shelve.

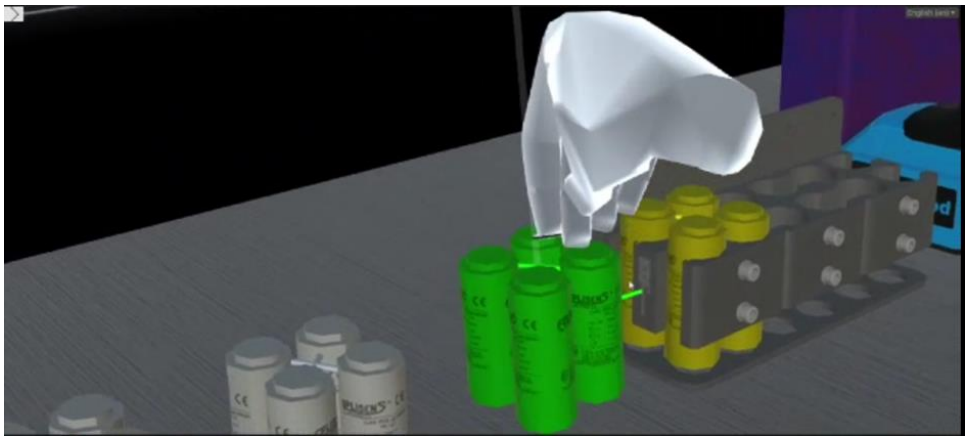

Figure 58: Pre-assembly of the parts with visual guiding cues.

Simplifying user interaction

Efforts are made to streamline user interaction, with features such as a hand menu (Figure 59) to minimize visual clutter in the virtual space. Ongoing and future enhancements, such as the integration of virtual and interactive training assistant with vocal capabilities further reduce the reliance on hand menu interactions, thereby promoting ease of use.

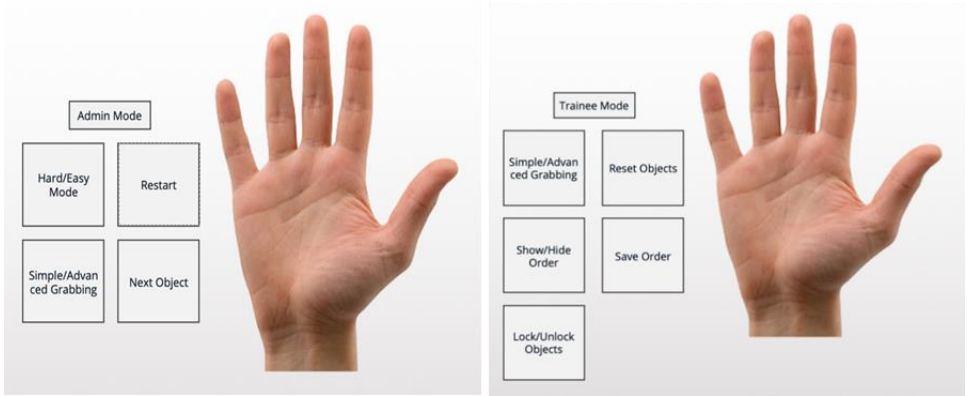

Figure 59: Hand menu design for different difficulty modes.

4.3.1.2 Application Workflow Diagram

The VOXReality Training Assistant (Figure 60) utilizes Hololight Space assembly application for visualization and interaction of 3D CAD files for industrial assembly training tasks. The application will run on a computer and the data is streamed to the client device like the Hololens2. The audio-video communication is enabled through WebRTC. The ASR component will generate textual counterpart of the user speech and transmit it to the Dialogue System. The Dialogue System will use NLU model to comprehend user input, and the NLG model to create coherent responses which will then be delivered as audio responses. These audio responses from the virtual assistant will enhance the user experience and the training efficiency during industrial assembly training in Hololight Space Assembly which is rendered to the client device.

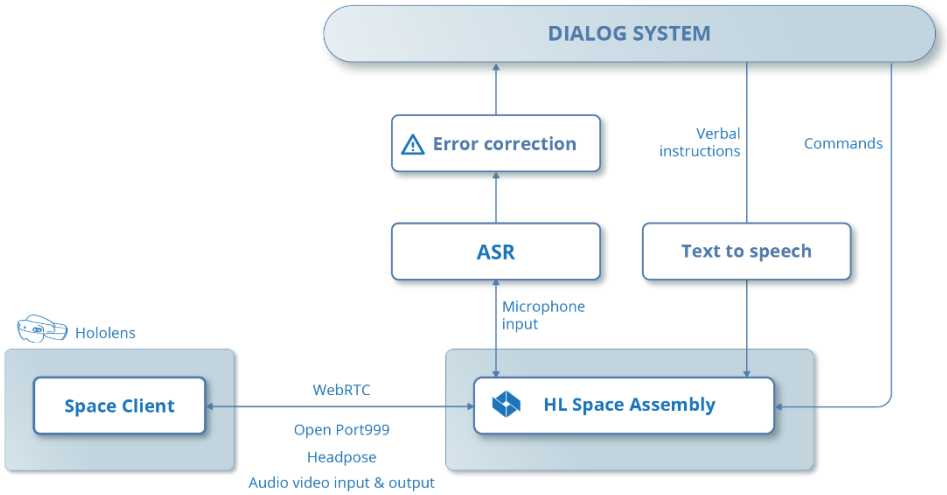

Figure 60: Application workflow diagram of the training assistant.

4.3.2 Implementation Details

4.3.2.1 Development Environment Setup

For the VOXReality project, the requisite hardware and equipment are meticulously selected to facilitate the generation, visualization, and interactive engagement with 3D content, specifically tailored for assembly training scenarios. The foundational elements encompass dedicated augmented reality glasses and an associated server device tasked with hosting the AR Training application. The current choice for augmented reality glasses is the HoloLens 2, a Microsoft product.

The VOXReality use cases demand the orchestration of a seamless flow of data, achieved through the utilization of the AR Training application, which employs remote rendering and application streaming through Hololight Stream. This method culminates in the streaming of the entire application, including rendered content, to the HoloLens 2 device. To meet the computational demands of this application, the requisite laptop must adhere to specifications, which include an operating system compatible with Windows 10 (Build 10.0.17763), Windows 11, or Windows Server 2019, a minimum of 16 GB of memory (with a recommended 64 GB), and a CPU selection ranging from an Intel i5 8th Gen. with 6 Cores or AMD Ryzen 7 3700X at the minimum, to an Intel i7 12 Gen. with 12 Cores or AMD Ryzen 9 3900X for optimal performance. GPU considerations mandate a minimum of NVIDIA GTX 1070Ti or NVIDIA GRID for VMs, with a recommended NVIDIA RTX 3080 TI, while storage requirements necessitate either SSD or NVMe technology.

The critical link in facilitating data exchange between the AR Application and the HoloLens 2 is the WebRTC protocol. This requires a robust 5 GHz Wi-Fi connection, with a minimum bandwidth of 20 Mbit, and a recommended bandwidth of 40 Mbit. Interaction with the AR application is orchestrated through the laptop interface, exemplified in the context of the AI models developed within the VOXReality project. Notably, in the future versions of the application, the microphones embedded within the HoloLens 2 device facilitate the transmission of speech data to the VOXReality ASR model, mediated through the AR Training Application, wherein interactions are confined solely to the main application, rather than the HoloLens 2 device.

4.3.2.2 Core Algorithms and Techniques

The streaming technology

The AR training application deals with computationally intense CAD files and real time manipulation of the CAD objects in the XR environment. The mobile XR devices with their limited computational processing power pose a challenge to the functionality and usability of such applications. An effective approach is to outsource the rendering process, enabling complete XR applications to be streamed from powerful local servers or the cloud. Hence the application will leverage Unity game engine technology and incorporate the Hololight Stream software development kit (SDK). Stream-enabled applications facilitate a client-server environment that offloads computationally intensive rendering to a dedicated server with robust graphics processing, streaming high-fidelity AR visualizations to client devices like the HoloLens. Hololight Stream mediates streaming via WebRTC protocol. The streaming process is two-fold between the server (XR application with integrated Hololight Stream plugin) and a client Hololight Stream app (installed on the XR device). The XR client receives pixel streams composing the content rendered by the XR application and returns sensor data back to the application. This plugin thus enables remote rendering and full application streaming, leveraging edge device computational and graphics processing power. Such remote rendering and streaming circumvent limitations of XR devices.

Animations

The training assistant utilizes animations made with keyframe animations in Unity. Through keyframe animation, the state of the object is recorded and the changes between each keyframe is interpolated. This allows basic animation of objects within the scene which includes object rotations and changing the position or size.

As part of the interactive assembly training, the assistant utilizes avatars to enhance the user experience.

Language model and ASR

The application will employ the automatic speech recognition (ASR) component developed by the consortium to transcribe users' spoken interactions with the digital assistant. The generated text from ASR will feed into the dialogue agent (DA) component where a dialogue system assists users in machine assembly, with English as the mediation language. The DA component will produce English responses that utilize text-to-speech technology to provide audio interaction. Moreover, the agent will be composed of natural language understanding to comprehend users' spoken requests and natural language generation to produce meaningful responses.

Algorithms

Majority of the codes and algorithms cannot be openly distributed. Nevertheless, with regards to certain functionalities, such as the microphone, the custom send functions of Hololight Stream are used. On the Server-Side, *CustomSend* object in the scene in Unity has been enabled. Associated script should be found in Packages under *Hololight.Isar.Runtime* in *CustomSendExample.cs. ServerApi, ApiConfig*, ConnectionCallbacks and *ConnectionHandle* are initialized on the Start. Currently, default message (ping) from Unity is running based on the timer *OnTimerElapsed* function in *CustomSendExample.cs* and can be modified to the use case. The message to be sent can be assigned under HlrCustomMessage defined under *ConnectionApi.cs*. Through the *ConnectionApi*, *PushCustomMessage* the message can be sent to the client. The type of encoding while assigning message as bytes stream must be taken care of. Following is an example to send “OpenKeyboard” message on a button press event to Client.

```
public void OpenKeyBoard()
{
        if (IsConnected)
        {
        byte[] msg = Encoding.ASCII.GetBytes("ShowKeyboard");
        IntPtr unmanaged = Marshal.AllocHGlobal(msg.Length);
        Marshal.Copy(msg, 0, unmanaged, msg.Length);
        HlrCustomMessage message = new HlrCustomMessage();
        message.Length = (uint) msg.Length;
        message.Data = unmanaged;
        _serverApi.ConnectionApi.PushCustomMessage(_handle, message);
        }
}
```

Incoming messages from the client are received under *OnCustomMessageReceived* event handler and can be verified and updated further depending on the use case.

```
OnCustomMessageReceived (in HlrCustomMessage message)
{
        int length = (int)message.Length;
        byte[] managedData = new byte[length];
        Marshal.Copy(message.Data, managedData, 0, length);
        string msg = Encoding.ASCII.GetString(managedData);
        Debug.Log($"Received custom message: {msg}");
}
```

On the Client-Side message is received (for reference) under: Remote Rendering in ImmersiveAppView.cpp under Init function in the callback m_*customMessageCallback*. Furthermore, message can be verified or used to trigger an event by registering in the *register_custom_message_handler* of the *ConnectionApi*.

An Example to open System Keyboard on the HoloLens based on message from Server-Side:

```
m_customMessageCallback = [ ](HlrCustomMessage* message, void* user_data) {
	auto* immersive = static_cast<ImmersiveAppView*>(user_data);
	uint32_t length = message->length;

	std::string msg {
		message->data, message->data + length};
	};

	if (msg == "ShowKeyboard") {
	immersive->RequestUserInput ();
	}
};

m_isarApi.connection.register_custom_message_handler (
	m_connectionHandle,
	m_customMessageCallback, this
);
```

This triggers function *RequestUserInput*() in the client to open System keyboard on the incoming message. Currently, default message from client (pong) to the server is assigned in the *onConnectionStateChanged* in the ImmersiveAppView.cpp running on timer periodically and can be changed according to the usage. To send a message back is similar to the Server-Side code and can be achieved by assigning to *HlrCustomMessage* using *ConnectionApi push_custom_message*. An example of sending every character or a string on a Key down event of the system keyboard from the client.

```
std::string msg;
const std::vector<uint8_t> raw(msg.begin(), msg.end());
HlrCustomMessage message = {};
message.length = msg.size();
message.data = raw.data();
ISAR_ASSERT(m_connectionHandle && m_connectionHandle != HLR_INVALID_HANDLE);
m_isarApi.connection.push_custom_message(m_connectionHandle, &message);
```

4.3.2.3 User Interface Implementation

The trainee's AR experience involves the virtual manipulation of object components in the training environment. For the virtual training, CAD files will be visualized in 3D and the parts of an industrial model will be assembled in the correct sequence on the object's frame. To make the industrial training task efficient and user friendly, a training assistant is introduced instead of the tedious hand menu interaction. The training assistant is equipped to perform verbal interaction using language models. The current capabilities of the training assistant are limited to operating in accordance with the directives of the trainee to skip a step or retrieve the file names. The training assistant, however, will be equipped in the future to process audio inputs from the trainee to launch the application, initiate training, and perform the assembly task with perpetual aid from the training assistant until the end of the task. The response will also be displayed as text at specific locations in the XR visual field. The ultimate output of this use case will manifest as a voice based interactive virtual assistant for industrial assembly training.

Using the 'admin' mode in the application, the user can define the order of assembly and add additional modular steps like scanning and pushing. The order of alignment can be tracked through tool tips linked to the individual parts of the object. Tooltips show the order of attached

parts, modular steps, and screws. The correctly attached or finished steps are marked with green tooltips and the rest are shown in blue (Figure 61). The tooltips can be displaced by the user, in case they overlap, or block the line of sight.

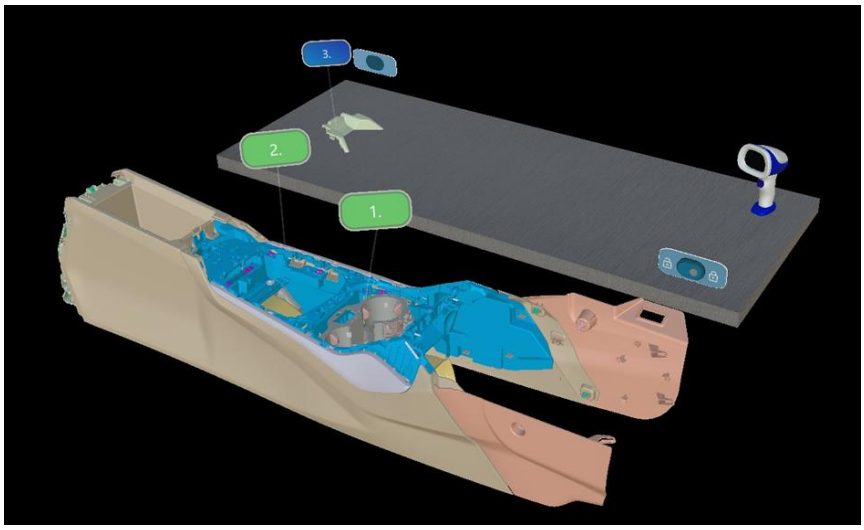

Figure 61: Tooltips displaying visual cues for correctly attached / finished steps.

The user can switch to 'trainee' mode to execute training in 3 different difficulty levels namely, easy, medium, and hard (Figure 62). The precise implication of each configuration was stated earlier.

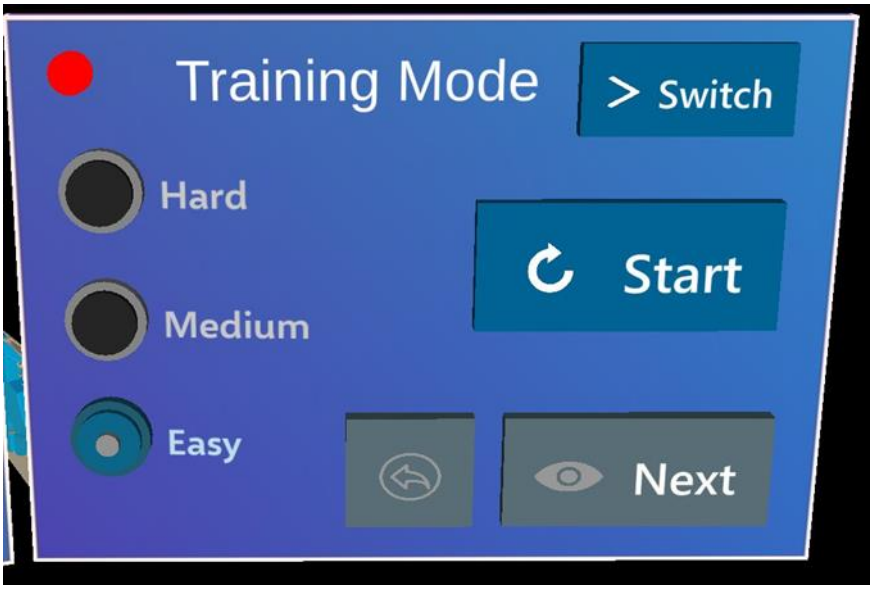

Figure 62: Menu to configure the difficulty level of the training.

Before the assembly task, the user can incorporate the 'table' or the 'shelve' feature which can be added from the asset-bundle and associate all the grabbable parts to it. Moving the table allows the user to move all the grabbable objects together within or outside of the XR visual field. The lock / unlock button located on the right side of the object provides the user with the capability to secure the shelf or table in a fixed position (Figure 63 and Figure 64). It is important to note that the design ensures the prevention of inadvertent shifts. Nevertheless, should it become necessary to modify the height of the table, this can be effortlessly achieved by unlocking the object.

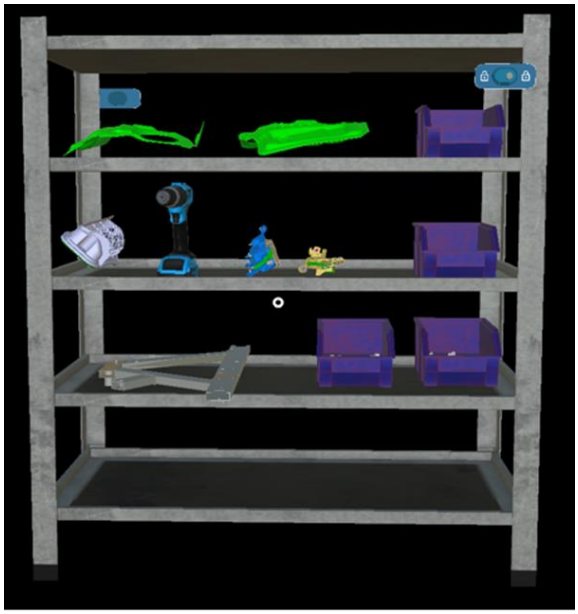

Figure 63: Shelve in the asset bundle, where the object lock feature is displayed on the top-right corner.

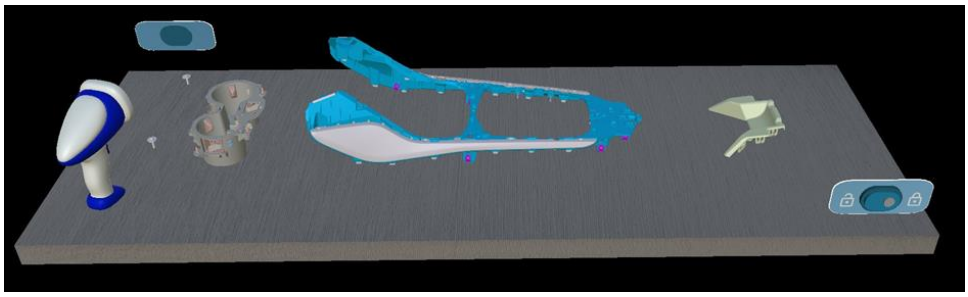

Figure 64: Table in the asset bundle, where the object lock feature is displayed on the bottom-right corner.

The Assembly application provides a ‘grabbing’ feature that facilitates the manipulation of virtual objects in a user-friendly manner. We have devised a grabbing algorithm that enhances the existing default grabbing functionality of the HoloLens 2. It detects the instances when the virtual hand engages with the three-dimensional object's surface and links the object to the hand when it is grabbed. Even if the HoloLens 2 loses visual contact with the hand, the system retains this information. When the hand reappears, the object swiftly aligns with the hand's position stabilizing the interaction with the object. This grabbing functionality is complemented by visual and auditory cues that inform the user whether they have successfully grabbed the object or released it. The visual cues employed depend on the difficulty level of the training, utilizing colour codes such as green and red to indicate the correct and incorrect parts being grabbed respectively (Figure 65).

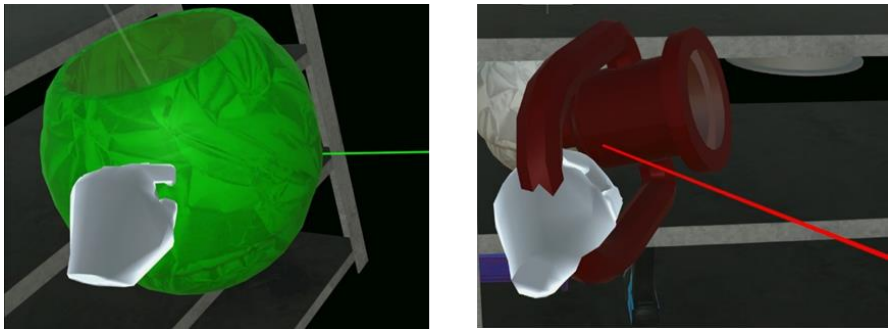

Figure 65: Colour-code visual cues showing the correct (green) and incorrect (red) object selection.

When the level of difficulty is chosen to be ‘easy’, the trace of the object is displayed in yellow at its destination and required rotation (Figure 66). There is furthermore a guiding line that extends from the part to its counterpart thereby aiding the user in identifying the precise location.

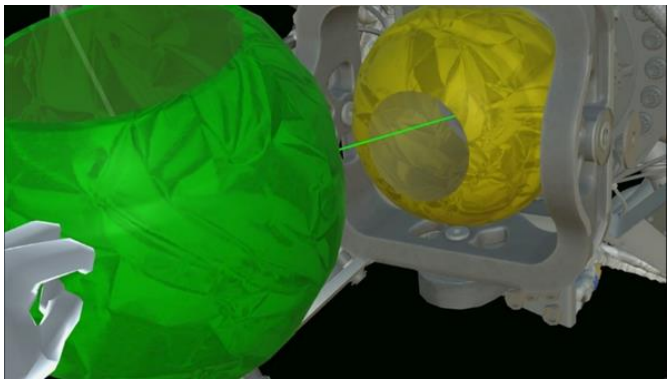

Figure 66: Colour-coded visual cues with the object destination (yellow) and a guing line connecting the grabbed object and its destination.

To assist the user to rotate, align and place the object, the application offers a simple and an advanced ‘snap’ feature. The ‘simple snap’ feature is triggered when the grabbable assembly part is connected to the invisible bounding box around the counter part by a single line. As a result, the simple snap executes an animation that accurately positions the grabbable part to its intended location. The 'advanced snap' feature reduces the bounding box size of the counterpart and requires accurate alignment of the grabbable part and the bounding box with 3 lines. The snapping animation only occurs when all 3 lines are aligned accurately, and the part reaches the fixed threshold of the bounding box. The application offers an object ‘locking’ feature which fixes the grabbable part after being placed in the solid model. The part could however be removed and displaced again using the ‘unlocking’ feature.

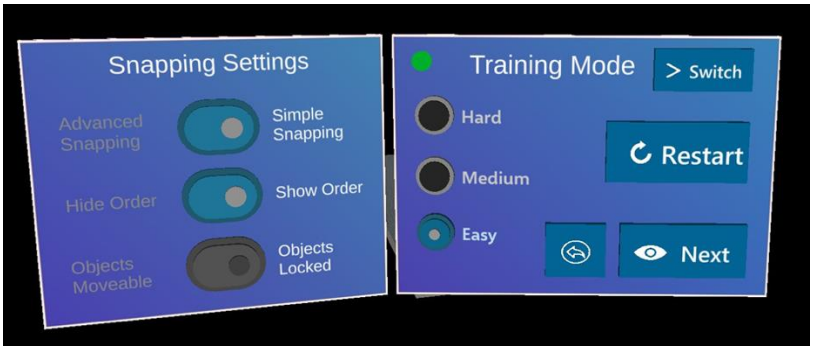

Figure 67: Menu to configure difficulty levels and snap features.

The iterative nature of this process ensures that all components of the CAD object are effectively joined together. Through this process, the user acquires the knowledge and comprehension necessary to successfully assemble the object virtually. Once this understanding is obtained, the virtual assistant can present the user with the opportunity to recommence the training at potentially more challenging difficulty levels. Consequently, the user is encouraged to attempt the different difficulty levels in a sequential manner. By engaging in this serial assembly practice and receiving ongoing support from the virtual assistant, the user is provided with a distinct training environment. Upon completing each difficulty level, the trainee is furnished with relevant performance metrics, such as the time taken to complete the entire task, or the time spent on each individual step. This assisted virtual training enables the trainee to practically assemble the industrial gadgets with efficiency in the absence of assistance. Through the implementation of a practical and user-friendly environment, certain elements of muscle memory for industrial assembly can be imparted. The dissemination of this training can be effortlessly extended to various factories without the need for any hardware equipment apart from the initial setup of this training assistant. In order to conduct ergonomic evaluations, the virtual components have the capability to be superimposed onto an actual framework, thereby facilitating the examination of these aspects.

4.3.2.4 Summary of Achieved User Requirements

The completed tasks in the initial phase of the project have successfully achieved their objectives, focusing on developing an augmented reality (AR) industrial training scenario. This includes setting up the environment, offering multiple difficulty levels, ensuring language compatibility, and integrating interactive features such as object manipulation and user feedback. Additionally, progress has been made in assessing user performance and interaction with the virtual agent, although some aspects remain partially completed, such as implementing assessment metrics and providing options for instructional delivery. However, the overall completion of these tasks sets a solid foundation for the next phase of development.

In the ongoing tasks, there are several open objectives awaiting completion, mainly centred around enhancing user interaction with the virtual agent, refining the user interface, and improving feedback mechanisms. These tasks aim to provide users with a more intuitive and personalized training experience, including features like on-demand assistance, customizable interfaces, and immediate feedback on task completion. Although these tasks are yet to be finished, the completion of the interface to Hololight Space in the initial phase ensures that incorporating these open tasks into the second iteration poses minimal risk to the project timeline. The upcoming pilots will primarily focus on testing the general intractability with the virtual agent, with feedback utilized to further refine usability and functionality.

5 Conclusions

The work presented in this document is connected to the WP4, whose primary objectives are to deploy the VOXReality AI models across all use case along with providing comprehensive documentation on the various deployment methods for these AI models. Additionally, this WP is responsible for the sharing of those models to third parties. Moreover, it involves the investigation of the 'once-for-all' training scheme and the inference optimization methods. To this end, activities regarding the implementation of XR applications are conducted in the WP4.

This document presents an overview of the SOTA methods regarding the "once-for-all" training methods and the AI model optimization techniques. Additionally, it includes some initial results from the application of VOXReality optimization tool. Subsequently, it presents the VOXReality model optimization approach that is applied in VOXReality models. Regarding the deployment of pre-trained models, the document outlines the process for deploying VOXReality models onto a development server following the CI/CD pipeline, for validation and testing purposes. Furthermore, it includes guidelines on the deployment of models via source code and Docker images. The sharing of the model is achieved by Hugging Face. Lastly, it provides the implementation details of the three VOXReality XR applications: VR Conference, Augmented Theater and Training Assistant.

Focusing on the results achieved up to the 17th month, the proposed VOXReality post-training optimization method generally maintains or even enhances prediction quality while reducing the inference time. Additionally, the shift to ONNX with graph optimization further reduces inference time, indicating its effectiveness in streamlining model performance without significantly compromising output quality. Concerning the deployment methods of VOXReality models, two methods are currently presented, those are source code-based and container-based deployment. The container-based deployment involves utilizing Docker images from Docker Hub as well as employing Docker Compose Comprehensive deployment guidelines for these methods are provided to assist users. Additionally, Hugging Face host a dedicated repository where the pre-trained VOXReality models are listed, each accompanied by documentation in the form of "Model Card".

The VOXReality models have been already integrated in the three VOXReality applications to enhance user immersion. The VR Conference application has successfully implemented 25 out of 49 user requirements, including most high and medium-priority ones. It features virtual avatars with dedicated cartoonish agents for interaction and navigation aids like a virtual map and visual cues. Additionally, it offers real-time translation in five languages with customizable subtitles for each speaker, enhancing the immersive experience. The Augmented Theater application has fulfilled all high and medium priority user requirements and one low priority, except those related to theatrical elements. The application can provide personalized subtitles, virtual effects, and background play information. The Training Assistant application establishing an environment with various difficulty levels, language support, and interactive elements like object manipulation and feedback. Some parts, like assessment metrics and instructional options, need completion, but the groundwork is laid for further development.

In the upcoming period, the consortium will focus on planning and execution of Pilot 1I, which aims to test the VOXReality XR application in a real-world environment as well as to gather feedback from the users. The planning of the pilot will be included in the D5.1 "Pilot planning

and validation V1” on M18, while the execution details and results analysis will be presented in D5.3 “Pilot analysis & feedback V1” on M21. Moreover, we will continue to work on “once-for-all” training concept and model optimization techniques. Our goal is to provide a tool that facilitates the once-for-all training method and to expand the CLI tool for inference optimization. This has as goal to decrease the model size and processing demands, ensuring efficient and effective use of AI models during the deployment to various hardware environments, including those with limited resources. The deployment methods and their accompanying guidelines will be expanded to provide more options. The XR application will be finalized considering both the user and technical requirements as well as the received feedback from Pilot 1. Any potential changes and additions will be documented in the second version of this deliverable with title “D4.1.2 – Model deployment analysis V2” on M32.

6 References

[1] H. Cai, C. Gan, T. Wang, Z. Zhang and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” *arXiv preprint arXiv:1908.09791,* 2019.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems,* vol. 30, 2017.

[3] W. Kwon, S. Kim, M. W. Mahoney, J. Hassoun, K. Keutzer and A. Gholami, “A fast post-training pruning framework for transformers,” *Advances in Neural Information Processing Systems,* vol. 35, pp. 24101-24116, 2022.

[4] S. Yu, T. Chen, J. Shen, H. Yuan, J. Tan, S. Yang, J. Liu and Z. Wang, “Unified visual transformer compression,” *arXiv preprint arXiv:2203.08243,* 2022.

[5] X. Wu, Z. Yao, M. Zhang, C. Li and Y. He, “XTC: Extreme Compression for Pre-trained Transformers Made Simple and Efficient,” *Advances in Neural Information Processing Systems,* vol. 35, pp. 3217-3231, 2022.

[6] J. Zhang, H. Peng, K. Wu, M. Liu, B. Xiao, J. Fu and L. Yuan, “Minivit: Compressing vision transformers with weight multiplexing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12145-12154.

[7] P. Ganesh, Y. Chen, X. Lou, M. A. Khan, Y. Yang, H. Sajjad, P. Nakov, D. Chen and M. Winslett, “Compressing large-scale transformer-based models: A case study on bert,” *Transactions of the Association for Computational Linguistics,* vol. 9, pp. 1061-1080, 2021.

[8] H. Cai, C. Gan, T. Wang, Z. Zhang and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” *arXiv preprint arXiv:1908.09791,* 2019.

[9] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen and Q. Liu, “Dynabert: Dynamic bert with adaptive width and depth,” *Advances in Neural Information Processing Systems,* vol. 33, pp. 9782-9793, 2020.

[10] O. Zafrir, A. Larey, G. Boudoukh, H. Shen and M. Wasserblat, “Prune once for all: Sparse pre-trained language models,” *arXiv preprint arXiv:2111.05754,* 2022.

[11] M. Chen, H. Peng, J. Fu and H. Ling, “Autoformer: Searching transformers for visual recognition,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12270-12280.

[12] VOXReality, “D3.1 - Advanced AI multi-modal for XR analyis V1,” 2023.

[13] VOXReality, “D2.3 - Development infrastructure and integration guidelines,” 2023.

[14] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik,* vol. 1, pp. 87-290, 1959.

VOXReality

Voice driven
interaction in XR spaces

Funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Directorate-General for Communications Networks, Content and Technology (DG CNECT). Neither the European Union nor the granting authority can be held responsible for them.