



VOXReality

VOICE DRIVEN INTERACTION IN XR SPACES VOICE DRIVEN INTERACTION IN XR SPACES

Model Deployment Analysis V2

WP 4

30/06/2025



Funded by
the European Union

Version 2.0
 WP 4
 Dissemination level Public
 Deliverable lead SYN
 Authors Ioannis Oikonomidis, Ntinos Prousalidis (SYN), Athanasios Ntovas, Georgios Papadopoulos, Sotiris Karavarsamis, Stefanos Biliouris, Dimitrios Pattas, Petros Drakoulis, Alexandros Doumanoglou, Dimitris Zarpalas (CERTH), Yusuf Can Semerci (UM), Leesa Joyce, Gabriele Princiotta (HOLO), Olga Chatzifoti (MAG)

Reviewers Manuel Toledo (VRDays), Spiros Borotis (MAG)

Abstract

Keywords Model Deployment, Model Sharing, Deployment Guidelines, once-for-all Training, Inference Optimization, Interactive XR Application Development

License



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0). See: <https://creativecommons.org/licenses/by-nc/4.0/>

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	Editor(s)	Changes to previous version
0.1	2025-05-01	SYN	Ntinios Prousalidis Ioannis Oikonimidis	ToC released
0.2	2025-05-02	SYN	Ntinios Prousalidis , Ioannis Oikonimidis	Added in Sec. 4.1
0.3	2025-05-28	CERTH	Athanasios Ntovas, Petros Drakoulis	Added in Sec. 2
0.4	2025-05-28	HOLO	Leesa Joyce, Gabriele Princiotta	Added in Sec. 4.3
0.5	2025-06-13	MAG	Olga Chatzifoti	Added in Sec. 4.2
0.6	2025-06-13	UM	Yusuf Can Semerci	Contribution to Sec 2, 3, 4
1.1	2025-06-13	SYN	Ntinios Prousalidis	Added Executive Summary and Introduction; performed editing
1.2	2025-06-30	VRD	Manuel Toledo	Internal review complete
1.21	2025-06-30	MAG	Spiros Borotis	Internal review complete
1.3	2025-06-30	SYN	Ntinios Prousalidis	Revisions Complete



Table of Contents

Version History.....	3
Table of Contents.....	4
List of Abbreviations & Acronyms.....	6
List of Figures	8
List of Tables	11
Executive Summary	12
1 Introduction.....	12
1.1 Intended Audience.....	13
1.2 Relations to the other activities	14
1.3 Document Structure.....	14
2 Model Training and Inference Optimization	14
2.1 Foundational Concepts of Model Optimization.....	15
2.2 Overview of State-of-the-Art Methods.....	17
2.2.1 The Once-for-All Concept and its Variants.....	18
2.3 The VOXReality Model Optimization Approach.....	22
2.3.1 Aiming and Background.....	22
2.3.2 Matrix Decomposition	23
2.3.3 SVD Synthesis.....	27
2.3.4 Future Work.....	29
2.4 VOXReality models ONNX repository.....	29
3 Model Deployment and Sharing.....	31
3.1 Deployment of VOXReality AI Models in Development Server.....	32
3.2 Deployment Guidelines.....	37
3.2.1 Source code-Based Deployment	37
3.2.2 Container-Based Deployment.....	38
3.2.2.1 Deployment using Docker Hub Images	39
3.2.2.2 Deployment using Docker Compose	40
3.3 Model Sharing	41
4 VOXReality XR Applications	44
4.1 Virtual Reality (VR) Conference.....	44
4.1.1 System Architecture and Design.....	44
4.1.1.1 3D Models and Scenes Design	44
4.1.1.2 Application Workflow Diagram	45
4.1.2 Implementation Details	54
4.1.2.1 Development Environment Setup.....	54
4.1.2.2 3D Models and Scene Creation.....	55
4.1.2.3 Core Algorithms and Techniques	57
4.1.2.4 User Interface Implementation	58



4.1.2.5	Summary of Achieved User Requirements.....	63
4.2	Augmented Reality Theatre	66
4.2.1	System Architecture and Design.....	69
4.2.1.1	3D Models and Scenes Design	71
4.2.1.2	Application Workflow Diagram	74
4.2.2	Implementation Details	75
4.2.2.1	Development Environment Setup.....	75
4.2.2.2	3D Models and Scene Creation.....	77
4.2.2.3	Core Algorithms and Techniques	79
4.2.2.3.1	Transcription.....	79
4.2.2.3.2	Vision language	82
4.2.2.3.3	Translation.....	82
4.2.2.3.4	VFX triggering logic	83
4.2.2.4	User Interface Implementation	85
4.2.2.4.1	AR Theater control server.....	85
4.2.2.4.2	AR Theater AR client.....	92
4.2.2.4.3	AR Theatre audio player client.....	97
4.2.2.5	Summary of Achieved User Requirements.....	98
4.3	Training Assistant.....	102
4.3.1	System Architecture and Design.....	102
4.3.1.1	3D Models and Scenes Design/Creation.....	102
4.3.1.2	Application Workflow Diagram	104
4.3.2	Implementation Details	108
4.3.2.1	Development Environment Setup.....	108
4.3.2.2	Core Algorithms and Techniques	109
4.3.2.3	User Interface Implementation	112
4.3.2.4	Summary of Achieved User Requirements.....	118
5	Conclusions	122
6	References	124



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 Interface
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)
DFO	: Derivative-Free Optimization
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
SVD	: Singular Value Decomposition
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

Figure 1. Conventional pruning framework VS Proposed pruning framework [3]	17
Figure 2. The compression scheme of [4].	17
Figure 3. The compression scheme of MiniViT [6]	18
Figure 4. Train the network once and extract the appropriate sub-network for each different hardware setup.	19
Figure 5. Illustration of the progressive shrinking process for CCNs to support different depth D, width, W, kernel size K and resolution R.	19
Figure 6. The two-stage procedure to train with DynaBert.	20
Figure 7. The Prune OFA training scheme.....	21
Figure 8. Detailed transformer block in an AutoFormer structure with all changeable parameters.....	21
Figure 9. The values of the changeable dimensions. Tuples of three values in parentheses represent the lowest value, the highest value, and step of each tunable parameter.	22
Figure 10. Classical weight sharing (left) vs. Weight entanglement of AutoFormer (right).	22
Figure 11. The Singular Value Decomposition (SVD) basic principle.	24
Figure 12. NaVQA evaluation accuracy for adaptive SVD and Kronecker decompositions across different Reduction Rates.	27
Figure 13. Loading an SVD Synthesized model in Python.	28
Figure 14. The VOXReality HuggingFace repository.	30
Figure 15. Example of Dockerfile.	33
Figure 16. GitLab CI/CD Add variable.....	33
Figure 17. GitLab CI/CD Repository Variables.....	34
Figure 18. Template of .gitlab-ci.yaml file	35
Figure 19. GitLab CI/CD Pipeline when push to main.	35
Figure 20. GitLab CI/CD Create a new tag.	36
Figure 21. GitLab CI/CD Pipeline when create a new tag.	36
Figure 22. VOXReality DockerHub.	39
Figure 23. Example of docker-compose.yml file.....	40
Figure 24. VOXReality Hugging Face repository.....	42
Figure 25. Workflow for communication with the Virtual Agent.....	46
Figure 26. Workflow for communication with the Virtual Agent.....	48
Figure 27. Weighted Graph with Manhattan Distance and Non-Diagonal Connectivity	49
Figure 28. Workflow of Navigation Request Handling Using LLM and Dijkstra's Algorithm	50
Figure 29. The way Angle and Distance change as the user arrives-to or crossing an object. The correct detection of these two events is crucial for the creation of the navigation dataset.	51
Figure 30. Workflow of the translation system.	52
Figure 31. Blender Interface for room designing.	55
Figure 32. Blender Interface for shading.....	56
Figure 33. Mozilla's Spoke Interface.	57
Figure 34. User Panel element.	59
Figure 35. Map Component for the Trade Show Area.....	59
Figure 36. Help slides.	60
Figure 37. Translate Button element.....	61
Figure 38. Virtual Agent Panel (Left), Translation Panel (Right).....	62
Figure 39. Loading animation.	62
Figure 40. AR Theatre system - high level overview	67
Figure 41. VFX triggering methods using verbal and visual cues	68
Figure 42. AR Theatre architecture - detailed version	69
Figure 43. Use of NMT service in AR Theatre	70
Figure 44. AR Theatre architecture - extended version.....	71
Figure 45. Storyboard excerpts.....	72



Figure 46. From storyboard to final assets over iterations.....	73
Figure 47. Matching the physical and digital space elements in the scene design	73
Figure 48. Data logging plan.....	75
Figure 49. Development tools: Magic Leap Hub 3	77
Figure 50. Development methods: diagrams and mockups for multidisciplinary collaboration..	77
Figure 51. Custom created 3D model using Blender with hand-painted textures	78
Figure 52. Procedural Visual Effects using Unity's Visual Graph.....	79
Figure 53. ASR upload subtitle endpoint.....	80
Figure 54. ASR Upload subtitle csv sample	80
Figure 55. ASR streaming and matching parameter configuration	80
Figure 56. Quality assurance methods: confidence-based filtering and error correction.....	81
Figure 57. Contextual translation endpoint.....	83
Figure 58. Contextual translation code block	83
Figure 59. Sample of translated lyrics across the VOXReality languages	83
Figure 60. Sample of VFX plan in csv format.....	84
Figure 61. Example of a VFX trigger method and its result.....	84
Figure 62. AR Theatre Server - Network tab.....	85
Figure 63. AR Theatre Server - Download tab	86
Figure 64. AR Theatre Server - Content sample	86
Figure 65. AR Theatre Server - Preview tab	87
Figure 66. AR Theatre Server - AI Services tab.....	87
Figure 67. AR Theatre Server - AI-Audio tab: Streaming settings	88
Figure 68. AR Theatre Server - AI-Audio tab: ASR response log & WebSocket messages	89
Figure 69. AR Theatre Server - AI-Audio tab: summary of configurable parameters.....	89
Figure 70. AR Theatre Server - AI-Vision tab – VQA with no stage event detected	90
Figure 71. AR Theatre Server - AI-Vision tab – VQA with stage event detected	90
Figure 72. AR Theatre Server - VFX tab.....	91
Figure 73. AR Theatre - Systems for Pilot 2 conditions.....	91
Figure 74. AR Theatre Server - Interfaces for manual and AI mode	92
Figure 75. AR Theatre Server - manual mode interface setup	92
Figure 76. AR Client - Language selection menu.....	93
Figure 77. AR Client - Introduction to AR device.....	93
Figure 78. AR client – Menu panel with scene selection	94
Figure 79. AR Client - Tutorial to application.....	95
Figure 80. AR Client - Subtitles customization menu	95
Figure 81. AR client - Extras scene content.....	96
Figure 82. AR Client - Start of performance.....	96
Figure 83. AR Client - Sample scene of performance featuring captions and triggered VFX....	97
Figure 84. AR Theatre audio player user interface.....	98
Figure 85. 3D scene with Raptor engine on the left and the table on the right	102
Figure 86. 3D scene with table and interactive objects – grabbed object in green and destination in yellow with a green guiding line. Object has a tool tip.....	103
Figure 87. The screwing logic with audio feedback.....	103
Figure 88. 3D scene with table, Raptor engine and assembly	104
Figure 89. 3D scene with table and display panel with a video being played.	104
Figure 90. Training Assistant application workflow.....	105
Figure 91. The logical scheme of the training steps in Free Mode.	107
Figure 92. The ASR transcription endpoint.....	109
Figure 93. Scheme showing ARTA behaviour based on the user requests (Questions in green and Commands in red) and the other parameters	111
Figure 94. Examples of responses to user's questions	112
Figure 95. DA's response to user's request for help through video and hint.....	112
Figure 96. Example of object tooltip.....	113
Figure 97. The display panel with visual feedback.....	113

Figure 98. The display panel with feedback on action performed by Voxy and the listening feedback with green mic and green Voxy avatar	114
Figure 99. The tutorial with guidance for the user regarding MRTK commands	115
Figure 100. The tutorial with guidance regarding the cheat sheet	115
Figure 101. The tutorial with guidance regarding the application logic	116
Figure 102. The tutorial with practice sessions on voice interaction	116
Figure 103. The blue popup panel giving user feedback regarding the replacement of the misplaced object	117
Figure 104. The blue popup panel giving user feedback regarding an action that cannot be logically performed	117

List of Tables

Table 1. The theoretical maximum Reduction Rate of various VOXReality models.	25
Table 2. Evaluation accuracy (BLEU) of Uniform and Adaptive SVD.....	26
Table 3. Different compression profiles for NaVQA.	28
Table 4. VOXReality models and their inference-time (ms) across various file formats, weight-quantization schemes and host device-types.	29
Table 5. VOXReality components used in each use case.	44
Table 6. VR Conference. Achievement of user requirements	63
Table 7. AR Theatre - Achieved user requirements.....	98
Table 8. The association between step number, training object, and the related manual instructions	107
Table 9. Development Environment Specifications – Hardware	108
Table 10. Development Environment Specifications – Software.....	108
Table 11. Achievement of User Requirements	118



Executive Summary

This document presents the final outcomes of the VOXReality project in relation to the training, optimization, deployment, and sharing of its AI models, as well as the implementation of the VOXReality eXtended Reality (XR) applications. It provides a detailed retrospective analysis of the adopted “**once-for-all**” (OFA) training methodology, along with the **optimization techniques** that were applied to reduce model size and computational requirements while preserving performance. The VOXReality model optimization approach, developed during the project, successfully supported model pruning, quantization, and export to common formats such as ONNX, facilitating deployment across diverse hardware platforms.

The document also reports experimental results validating the effectiveness of the implemented optimization strategies. Deployment and sharing mechanisms for the pretrained AI models were defined and realized, including **source code-based deployment** and **containerized solutions**, ensuring accessibility, portability, and reproducibility. 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 implemented advanced Natural Language Processing (NLP) models based on transformer architecture. Therefore, it was 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 addressed 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 implemented 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 was a critical aspect. VOXReality addressed 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 was further enhanced by VOXReality's comprehensive deployment guidelines, which assisted the 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 were deployed in three distinct use cases: Virtual Reality (VR) Conference, Augmented Reality (AR) 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 would 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 was 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 was performed in all three tasks of (T4.1, T4.2, T4.3) of WP4 until the end of the 38th 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 multi-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 as 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.

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.

The main scope of our work was 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 smaller networks from a single large one, that are conditioned on the constraints of the targeted deployment platform. In addition to the introduction of our OFA adaptation (SVD Synthesis) and its corresponding Command Line Tool (CLI), we also developed and released a secondary CLI tool that enables the weights-quantization of VOXReality models and their graph-optimization and extraction to the common ONNX¹ file format.

All the models that were developed for the three use cases (Augmented Reality Theatre, Virtual Reality Conference and Training assistant) were 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.

2.1 Foundational Concepts of Model Optimization

Before we get deep into the more advanced concepts, we should first establish some understanding of the broad families of model-compression techniques available. Various model 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 losslessly compresses the remaining elements of the network achieving the smallest possible representation of the model. Subsequently, we will delve into the various optimization components:

Quantization

Quantization can be equally applied at different stages of the model development process:

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

¹ <https://onnx.ai/>

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:

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, in some form, already present in various common model representation formats.

Now that we have set the basis of understanding, we can continue with the prevalent, more advanced techniques found in modern applications.

2.2 Overview of State-of-the-Art Methods

There is a variety of transformers' optimization techniques and frameworks currently under development, signaling 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 1) 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.

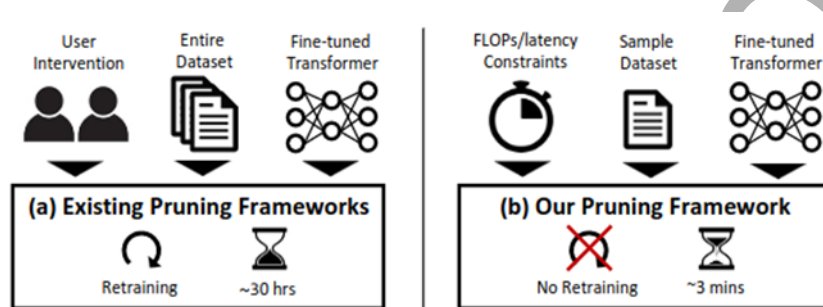


Figure 1. Conventional pruning framework VS Proposed pruning framework [3]

Another work is Unified Visual Transformers Compression [4] which presents a unified framework for compressing Vision Transformers, combining pruning, layer skipping, and knowledge distillation (Figure 2) 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 achieves efficient model compression while maintaining performance on vision tasks.

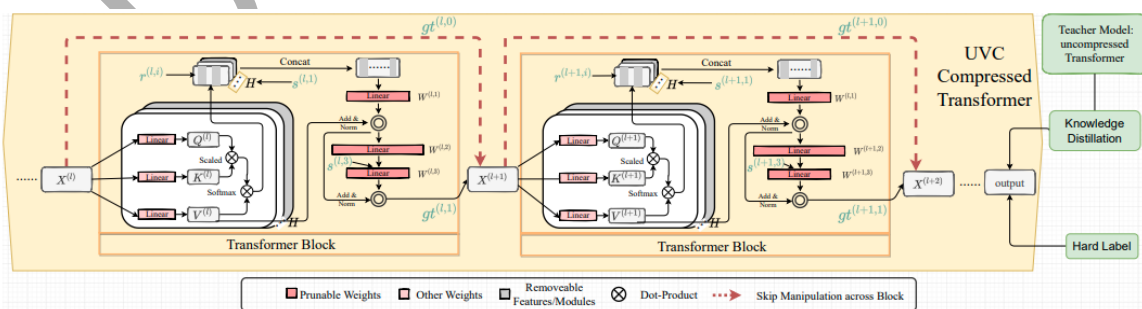


Figure 2. The compression scheme of [4].

An interesting study on extreme model reduction is XTC [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 paper presents a simple yet effective method for extreme compression of pre-trained transformers, demonstrating superior performance and compression rates compared to existing approaches where size reduction matters the most.

Another work is MiniViT [6] that introduces a novel compression framework for Vision Transformers. MiniViT combines weight sharing, transformation, and distillation techniques (Figure 3) 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.

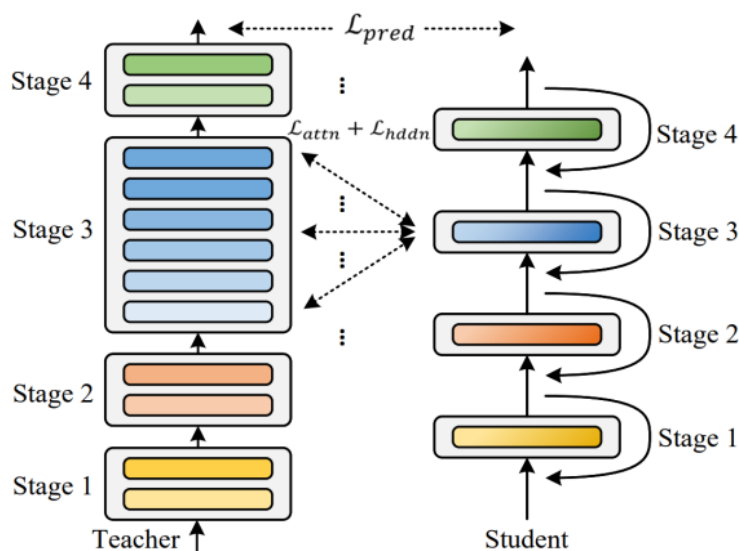


Figure 3. The compression scheme of MiniViT [6]

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.

2.2.1 The Once-for-All Concept and its Variants

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 4). 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 resources needed for model optimization and deployment.

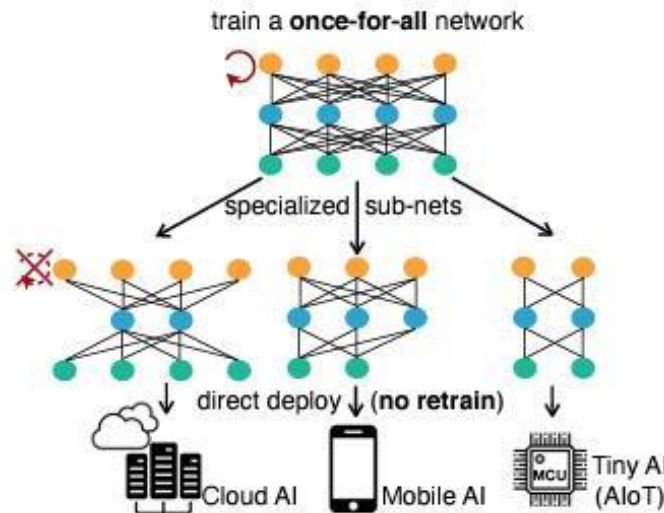


Figure 4. Train the network once and extract the appropriate sub-network for each different hardware setup.

Initially, the idea of OFA [1] was conceived for Convolutional Neural Networks. 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 5). Progressive shrinking works by enforcing training orders 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.

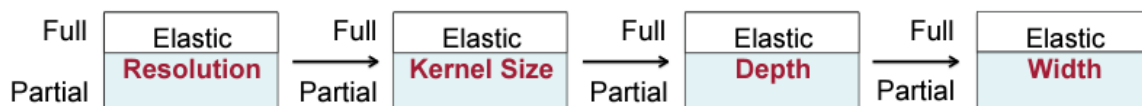


Figure 5. Illustration of the progressive shrinking process for CNNs to support different depth D , width, W , kernel size K and resolution R .

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 in 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 [8] 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 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 6 shows us 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 (DynaBERT_w). Then, using knowledge distillation it transfers knowledge from DynaBERT_w to multiple student sub-networks with adaptive both width and depth (DynaBERT).

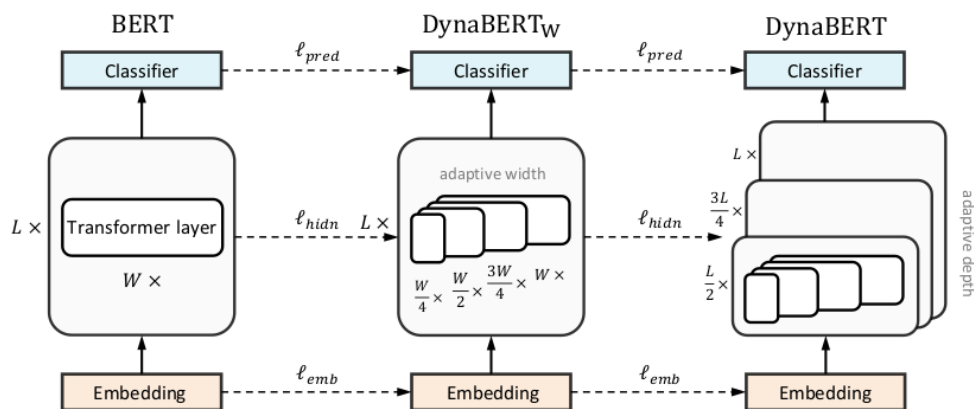


Figure 6. The two-stage procedure to train with DynaBERT².

A different work is *Prune Once For All* [9] (Figure 7). 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.

² Knowledge distillation is used to transfer knowledge from the frozen teacher network (left) to a student sub-network with adaptive width (middle). Then, knowledge distillation is used again to transfer knowledge to multiple student sub-networks with adaptive both width and depth (right).

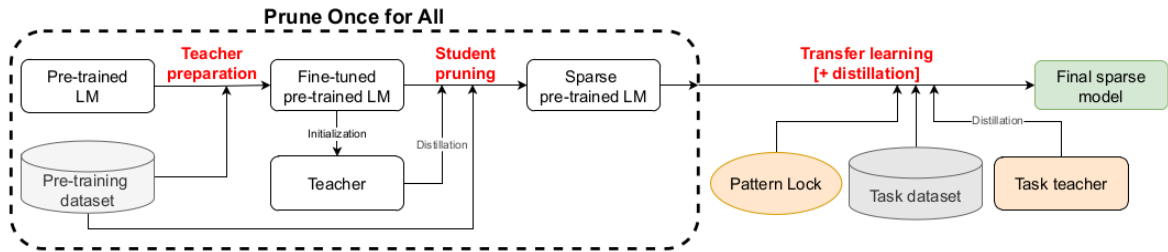


Figure 7. The Prune OFA training scheme.

Finally, the work we initially considered to have the greatest potential for use in VOXReality is Autoformer [10], 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 8). By allowing all these parameters to be adjustable during training, AutoFormer enables the exploration of vastly diverse transformer structures, adapted to specific requirements.

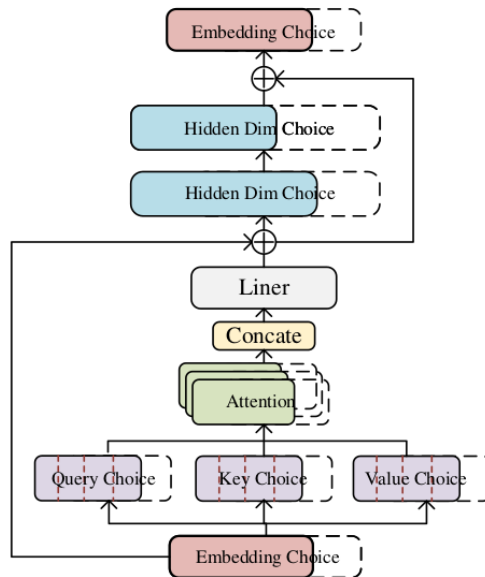


Figure 8. Detailed transformer block in an AutoFormer structure with all changeable parameters³.

The training procedure of AutoFormer involves creating a super-network with adjustable changeable dimensions (Figure 9) 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.

³ Depth, K-Q-V dimensions, embedding dimension, attention's number of heads and Multilayer Perceptron (MLP) ratio

	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 9. The values of the changeable dimensions. Tuples of three values in parentheses represent the lowest value, the highest value, and step of each tunable parameter.

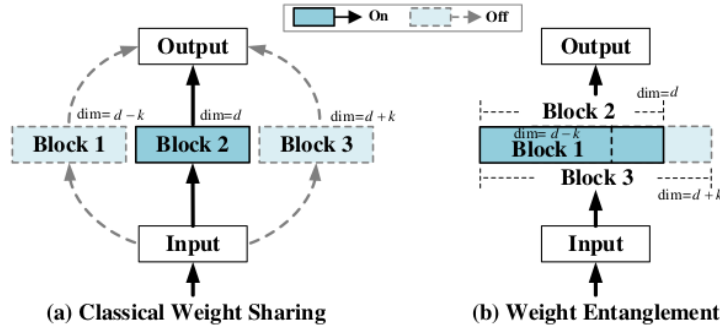


Figure 10. Classical weight sharing (left) vs. Weight entanglement of AutoFormer (right).

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 utilizing 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, a sub-network emerges that meets the desired balance of parameter count and accuracy, ensuring an effective trade-off between model complexity and performance.

2.3 The VOXReality Model Optimization Approach

2.3.1 Aiming and Background

The aim of this endeavor was to provide a framework for the optimization of various VOXReality models, guided by the project DoA which drafts an approach based on the Once-For-All (OFA) concept. Decomposing OFA into its basic elements, it is essentially about training generically one very large network which is not intended for inference, and then for the actual deployment being able to fine-tune and extract a working sub-network adapted to the nuances of the target platform and application.

The original Once-For-All concept which is designed for Convolutional Networks (CNNs) and its more recent derivatives like the Autoformer, rely on diminishing various intermediate model matrices while preserving most of their contribution to accuracy, constrained additionally by other arbitrary requirements like the model's target memory consumption, inference speed on a specific HW, etc. Both these techniques operate partially in training, meaning that the subject models must be designed from scratch with provisions for the support of this schema. These induced design choices, on top of the implicit added human effort, render the conduction of

the large model's generic training very resource-demanding, inefficient, and is deemed reasonable only for models whose extracted sub-models will see broad and intensive use. For all these reasons, we chose to put our efforts on developing solely for the post-training optimization case, enabling us to provide a truly universal method of network compression that is able with minimal provisions to support the treatment of all VOXReality models and more.

In order to provide a truly post-training optimization procedure, while adhering to the core OFA principles of representing the initial large trained model “once” and then combine parts of it, creating sub-models that perform well for various model sizes (corresponding to the capacity of the targeted deployment platform), we chose to build on the fundamental linear-algebra concept of matrix decomposition. There are various ways to decompose a matrix, and we experimented heavily with two of them: based on i) the Kronecker Product⁴ and ii) Singular Value Decomposition (SVD)⁵.

Initially, we invested on the Kronecker Product decomposition because of an observation we made that for the same number of parameters, the Kronecker representation allows for twice the rank⁶ (independent components) of the product matrix, retaining essentially a more expressive embedding space. After long experimentation with our prototype, we concluded that, unfortunately, rank does not correlate well with model accuracy, rendering this form of representation unsuitable for our purposes. So, we resorted to our next option for matrix decomposition, which was SVD. Overall, we created a multi-step process which aims to reconstruct and save an initial model with fewer parameters, subjected to an acceptable drop in accuracy. Another crucial facet of our devised scheme in our effort to borrow elements from the OFA concept, is to create any necessary intermediate data representations or conduct any lengthy operations only once and afterwards use / combine the produced elements on demand to create sub-models adapted to the intended deployment platform.

2.3.2 Matrix Decomposition

Decomposition, as a neural network compression technique, centers on the idea of breaking down large, redundant weight matrices or tensors within a trained model into a set of smaller, more efficient components. The core principle leverages the observation that these large parameter structures often contain significant redundancy, meaning their information content can be accurately represented by a lower-dimensional approximation. Techniques like Singular Value Decomposition (SVD) for matrices or various tensor decompositions (e.g., Tucker, CP and Kronecker decomposition) for higher-order tensors are applied to individual layers (e.g., convolutional or fully connected layers). Instead of storing and computing with the original large weight matrix, the model is reconfigured to use these smaller decomposed factors. For instance, a single large matrix W might be replaced by the product of two much smaller matrices, A and B , where $W = AB$. This replacement significantly reduces the total number of parameters and the computational operations (FLOPs) required during inference, making the model lighter and faster for deployment on resource-constrained devices.

Singular Value Decomposition (SVD) is a fundamental technique in linear algebra that involves breaking down any matrix A into three simpler ones: $A = U\Sigma V^T$ (Figure 11). Imagine it like taking a complex dish and separating it into its core ingredients. Here, U and V^T are orthogonal matrices whose columns (or rows for V^T) represent fundamental “patterns” within the data. The crucial part is Σ , a diagonal matrix filled with singular values, which are essentially numerical “weights” indicating how important each corresponding pattern from U

⁴ <https://mathworld.wolfram.com/KroneckerProduct.html>

⁵ <https://mathworld.wolfram.com/SingularValueDecomposition.html>

⁶ <https://mathworld.wolfram.com/MatrixRank.html>

and V^T is. Larger singular values mean more significant patterns, while smaller ones often represent noise or less vital information. This concept is core to many data compression techniques across various fields. Since the singular values are ordered by their significance, we can create a low-rank approximation of a large weight matrix. Instead of using the entire original matrix, we keep only the top k most important singular values and their associated vectors. This effectively replaces one big matrix with the product of two much smaller ones (e.g., $\approx U_k \Sigma_k V_k^T$). This substitution dramatically reduces the number of parameters and computational operations needed during inference, making the model much smaller and faster to run on devices with limited resources, all while typically maintaining most of its original accuracy. For a given weight matrix $W \in R^{M \times N}$ of the model, SVD decomposes it into $U \Sigma V^T$. When we perform a low-rank approximation by keeping only the top k singular values, the original matrix W is effectively replaced by the product of three smaller matrices: $U_k \in R^{M \times k}$, $\Sigma_k \in R^{k \times k}$, and $V_k^T \in R^{k \times N}$. The original number of parameters in W is $M \times N$. After decomposition, the number of parameters becomes $M \times k + k + k \times N$ (for U_k , Σ_k diagonal, and V_k^T , respectively), which simplifies to approximately $K(M + N)$ when k is small and the diagonal Σ_k is ignored or absorbed into one of the other matrices. For the decomposition to result in effective compression (a reduction in parameters), the new number of parameters must be less than the original: $K(M + N) < MN \Rightarrow K < MN/(M + N)$. This implies that k must be sufficiently small relative to M and N .

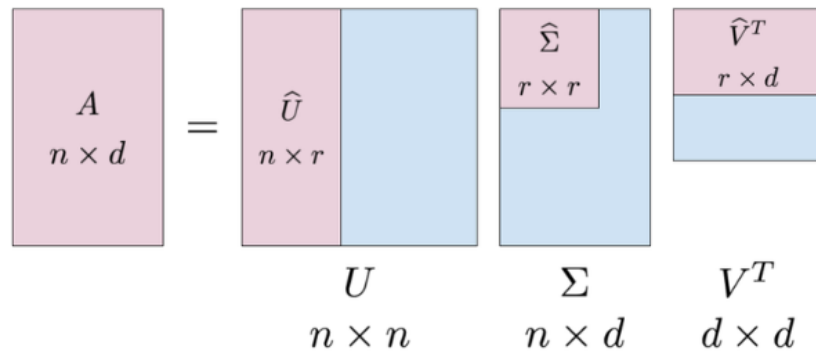


Figure 11. The Singular Value Decomposition (SVD) basic principle.

In Transformer models, such as the Vision-Language (VL) and Large Language Models (LLM) we are developing in VOXReality, SVD is primarily effective for compressing the extensive weight matrices found within the attention mechanisms and Feed-Forward Networks (FFNs) of each encoder and decoder layer. These include the Query, Key, and Value matrices, the attention output matrix, and the MLP layers within the FFNs. Those specific matrices are often massive and exhibit significant redundancy, making them ideal targets for SVD, which can effectively reduce their parameters and computational demands by replacing them with smaller, factorized approximations; thereby making the model lighter and faster for deployment. Having said that, SVD is not guaranteed to be universally applicable to all matrices potentially found in an architecture. Certain layers may be extremely sensitive to changes and are considered notoriously difficult to approximate, deeming them practically incompressible. This primarily includes the embedding layers and the final output layers (e.g., classification or language modeling heads). Embedding layers are crucial as they encode the fundamental semantic meaning of each token; aggressive SVD compression in these layers can severely degrade the model's basic understanding. Similarly, the final output layers directly translate the complex internal representations into the model's ultimate predictions. Compromising these layers through aggressive compression can lead to a disproportionately large and detrimental impact on the overall task performance, as they represent the critical last step in the model's processing pipeline. Taking into account all the above (i.e., excluding

incompressible matrices), we investigated various VOXReality models to estimate their theoretical maximum Reduction Rate. The results of this analysis can be seen in Table 1.

Table 1. The theoretical maximum Reduction Rate of various VOXReality models.

Model	Theoretical Maximum Model Reduction (Reduction Rate)
src_ctx_and_term_nllb_600M	10% (0.1)
src_ctx_and_term_nllb_1.3B	21% (0.21)
nllb_asr_synthetic_robust	11% (0.11)
navqa	78% (0.78)
t5_nlu_intent_recognition	12% (0.12)
whisper_small_el_finetune	29% (0.29)

As it is revealed, the BLIP-VQA-based NaVQA model stands out as the most compressible architecture, and thus we chose to utilize it as our “test mule” for our further concept developments. On the other hand, models with apparently low compressibility, like the “voxreality/src_ctx_and_term_nllb_600M”, face a unique challenge. This specific model contains embedding layers and a final output projection of very large dimensionality (256206×1024), each holding over 262 million parameters. The keeping of these layers is crucial. Embedding layers convert words into the numerical data the model understands, capturing their basic meaning, while the final output-layer translates the model's processing back into language, producing the prediction. Because these specific layers are so fundamental to the model's core understanding and its ability to give accurate answers, they are highly resistant to compression without significant loss in performance. This is the main reason why models like this have a low theoretical maximum reduction rate. So, like it was mentioned before, we are using the NaVQA model for further experimentation.

NaVQA is a fine-tuned BLIP-VQA architecture that integrates components for both vision and language processing. Specifically, its structure includes a vision encoder, a text encoder, and a text decoder. Stemming from the aforementioned compressibility analysis, we targeted and reconstructed the weight matrices comprising the attention and cross-attention mechanisms (i.e., the K, Q, V, and all MLP weight matrices) of the underlying architecture. We performed a low-rank SVD approximation, where the original matrix W was replaced by the product of three smaller matrices (U, Σ, V). In this naive implementation of SVD, we uniformly set the rank k of the decompositions for all compressible matrices to be the same, based on the targeted Reduction Rate of the output model (i.e., setting “ k ” such as the output model to exhibit the targeted reduction in parameters). Based on the previous discussion, understandably, this strategy may be not only sub-optimal but even worse, catastrophic to many network instances, due to the varying sensitivity to changes of the different layers and matrices in the model. Some layers carry less essential information and can be significantly compressed without much impact on the model's performance. On the other hand, other layers, even those structurally similar, may hold crucial features and thus even slight compression can cause a much larger drop in accuracy. This fundamental insight underscores that a uniform compression approach across the entire model is unlikely to be optimal, necessitating a more nuanced strategy to identify the most robust and effectively compressible sections of each specific model instance.



To test this notion, we conducted a systematic exploration (based on exhaustive search) to investigate whether specific layer combinations yield better accuracy (BLEU scores for NavQA). This involved iterating through various configurations, defined by combinations of model components and contiguous ranges of Transformer layers (from 1 to 12, with a minimum span of 5 layers). For every specific combination of model components and layer ranges we explored, we designated all the identified compressible weight matrices for SVD. This included the attention mechanism's Query, Key, Value, and projection weights, as well as the dense layers found in the Feed-Forward Networks, but strictly only within the selected sections and layer indices. This comprehensive combinatorial search allowed us to identify specific subsets of layers for SVD application and then evaluate their impact on model performance. This strategic compression methodology, which we term “Adaptive SVD”, involves selectively applying Singular Value Decomposition to specific subsets of a model's compressible layers. This adaptive approach aims to identify the optimal layer configurations for SVD, balancing overall model reduction with the critical preservation of performance. As seen in Table 2, our adaptive SVD approach yields significantly higher BLEU scores than the naive uniform implementation, especially as we push for greater model reduction. Additionally, based on insight gained through the conduction of such an extensive search of the underlying model configuration space, we can safely conclude that compressibility differs significantly between different components and depths across the model architecture.

Earlier in this chapter, we also mentioned the existence of another type of potentially useful decomposition representation, based on operations of the Kronecker product. Unlike SVD, which factorizes a matrix into singular vectors and values, Kronecker decomposition approximates a large matrix as a Kronecker product of two or more smaller matrices. This method is particularly effective for certain types of structured matrices (i.e., 3D and 4D matrices) and can theoretically offer significant parameter reduction by replacing a single large matrix with a composite of much smaller ones. Putting it to the test, we conducted a similar study using this compression scheme (in the same adaptive manner), reaching to the bitter conclusion that its characteristics, at least for the type of model we experimented with, lead to an inferior than SVD's model size – evaluation accuracy profile, deeming it unsuitable for our causes, as presented in Figure 12.

Table 2. Evaluation accuracy (BLEU) of Uniform and Adaptive SVD.

Model Reduction (Reduction Rate)	Uniform SVD (BLEU)	Adaptive SVD (BLEU)
Original model	91.3	
10% (0.1)	83.5	89.4
20% (0.2)	71.4	81.7
30% (0.3)	43.2	70.8
40% (0.4)	26.5	61.8
50% (0.5)	15.0	48.1

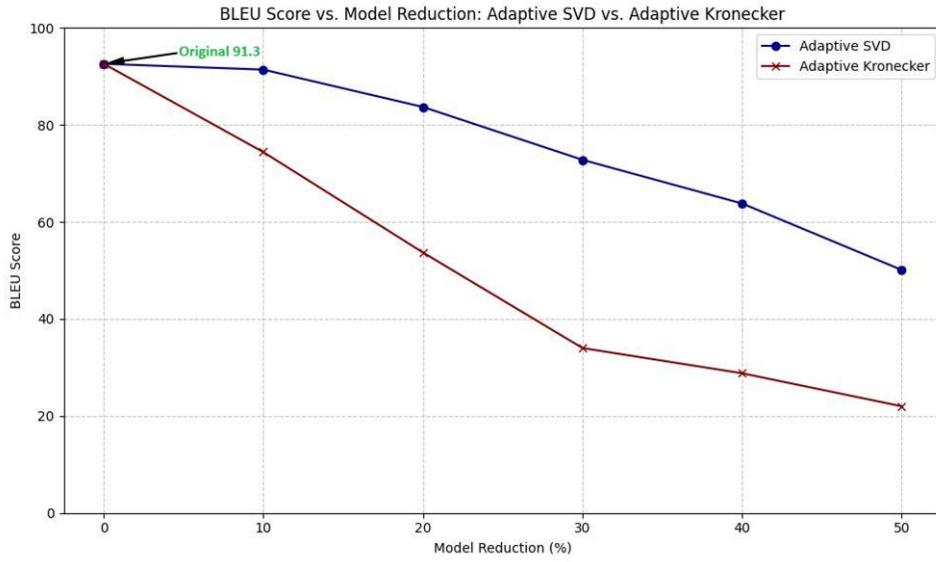


Figure 12. NaVQA evaluation accuracy for adaptive SVD and Kronecker decompositions across different Reduction Rates.

2.3.3 SVD Synthesis

First step in our process is the calculation of the two SVD factors of all model matrices, for each one up to the rank which yields a smaller number of parameters than the initial matrix. We call this meaningful max rank “m”. Since our aim is compression, there is no meaning in reconstructing the matrices with equal or more parameters than their original form. If this is the case, we should better utilize the matrices as they are, avoiding any reconstruction loss. Compression is essentially achieved by potentially replacing each model matrix with its two SVD factors UM (implying the simplified SVD notation where $M = \Sigma V^T$) of low rank “k”, such as the whole model achieves the targeted Reduction Rate. For each individual matrix A_i of dimensions $N_i \times D_i$ of the initial model, the aforementioned replacement can be formally expressed as:

$$A'_i = \begin{cases} A_i & \text{if } k_i > m_i \\ U_{ik_i} M_{ik_i} & \text{if } k_i \leq m_i \end{cases} \quad \text{with} \quad \begin{aligned} k_i &= \lceil (m_i + 1)(1 - RR_i) \rceil \\ m_i &= \left\lceil \frac{N_i D_i}{N_i + D_i} - 1 \right\rceil \\ RR_i &\in [0, 1), \quad k_i, m_i, N_i, D_i \in \mathbb{Z}, \geq 1 \end{aligned}$$

This essentially uniformly maps low-rank factors to RR values, with RR values close to 0 yielding the original matrix.

Next stage of our pipeline is the selection of the individual Reduction Rate of each matrix “ RR_i ” so as they all together conform with the targeted RR of the model. This is a very tricky decision to take, as we discussed extensively in the previous chapters, since not all model matrices are equally compressible, sensitive to changes or have the same contribution to the output model’s evaluation accuracy. Worse than that, trained models of even the same base-architecture, appear to behave differently. In the previous chapter, we referred to an early, exhaustive search-based approach (i.e., Adaptive SVD) dealing with the need to search for the individual matrices’ compression parameters required to consistently achieve a favorable size reduction-evaluation accuracy trade-off. Having set that, one can confidently claim that the decomposition-parameters versus evaluation-accuracy landscape is very complex, highly nonlinear and we have no clue on how to calculate derivatives on it, making it a suitable candidate for the conduction of Derivative-Free Optimization (DFO) [11].

DFO is a family of mathematical optimization techniques which do not require derivative calculations on the implicit loss / energy function they optimize, operating exclusively in a Black-Box manner. At their initialization, the only arguments required are: i) the problem variables, ii) the mathematically formulated constraints, iii) a “budget”, meaning how many steps or how much time they can spend in their search, and iv) whether to maximize or minimize the implicit function. They operate in an open-loop manner, requiring feedback at every iteration. At each step, the optimizer proposes a set of values for the problem variables which do not lead to constraints violations. Then, these proposed values are applied to the system and an evaluation phase follows where their effect on the system’s performance is measured. To conclude the iteration, the system returns to the optimizer the value of its latest evaluation. Considering this, the optimizer proposes a new set of values to be tested in the next iteration, until the whole budget has been depleted. The evaluation function, dataset, and metric, of course, can vary between use-cases and needs to be defined and implemented for the model(s) one cares about. In our current implementation, we include in our tool code and data that support the optimization and decomposition of our NaVQA model, as a means of demonstration.

A powerful aspect of our concept is the conduction of the aforementioned optimization, automating the selection of these otherwise very fragile and crucial decomposition parameters. The external Python library we relied on for the efficient conduction of DFO is Facebook Research’s Nevergrad⁷. Bridging our concept to DFO, we consider our problem variables to be the individual Reduction Rates “RR_i” of all model matrices in [0,1) so that the whole model’s RR to be at least equal with the targeted RR. For the NaVQA case, we set our optimizer to maximize the model’s generated navigation directions’ BLEU score. Once the optimization is finished, we synthesize and export our output model using the best yielded matrices configuration for our targeted Reduction Rate. On top of that we can always apply weights-quantization to further reduce the subject model size. In Table 3, four distinct compression profiles for NaVQA of varying size and accuracy (using a combination of SVD Synthesis and Weights Quantization) can be seen.

Table 3. Different compression profiles for NaVQA.

Preset	Reduction Rate	Quantization	Size	BLEU
Original	-	-	1.54 GB	91.3
Good	0.1	-	1.39 GB	88.3
Light	0.2	FP16	0.61 GB	80.5
Tiny	0.4	FP16	0.46 GB	60.2

We should note here that in order for our exported models to be used in Python applications, they need to be loaded in a different way, using a module we are providing. A snippet of such a tweaked model loading can be seen in Figure 13.

```

13 # model = BlipForQuestionAnswering.from_pretrained(model_path).to("cuda")
14 model = load_patched_model(model_type='blip',
15                             architecture_pretrained_path=model_path,
16                             compressed_state_dict_path=compressed_state_dict_path).to("cuda")
17

```

Figure 13. Loading an SVD Synthesized model in Python.

⁷ https://facebookresearch.github.io/nevergrad/getting_started.html

2.3.4 Future Work

Recent works, namely ASVD [12], SVD-LLM [13] and SVD-LLM V2 [14] attempt to improve naïve SVD with promising results. ASVD is a post-training singular value decomposition method that compresses large language models by transforming weight matrices based on activation distributions and optimizing layer-specific decomposition based on sensitivity. SVD-LLM appears to offer slightly better results by incorporating a truncation-aware data whitening technique and adopting parameter updates with sequential low-rank approximation to mitigate accuracy degradation. SVD-LLM V2 optimizes this pipeline further by assigning unique compression ratios to each weight matrix based on a calculation of the theoretical truncation loss.

Our vision is to develop an advanced compression-rate selection algorithm extending the SVD-LLM2 approach by grouping the model weight matrices not only by their type, but also by layer or a combination of those. We also plan to explore the utilization of Fisher information FW-SVD [15] into our envisioned SVD-LLM2 modified architecture, instead of plain SVD, potentially improving performance further. The full range of options to be investigated for further development include several SVD-based techniques (SVD, FWSVD, ASVD, SVD-LLM, SVD-LLM-v2) applied on BERT and DeBERTa models using the GLUE and Squad datasets. As long as a viable research-path has been identified, we will apply our findings to the VOXReality models, furthering the advancement of our offered ecosystem, and in parallel prepare the release of a new journal paper.

2.4 VOXReality models ONNX repository

As mentioned earlier, we have developed a command-line interface tool (CLI) able to quantize and export various VOXReality models into the common cross-platform ONNX model format. Our tool's source code can be found in the project's GitLab repository⁸. Using the tool, we pre-exported to ONNX format the majority of the VOXReality models, and uploaded them for convenience, ready to be used, in the project's HuggingFace repository⁹ (Figure 14). Additionally, in Table 4 we also provide indicative performance measurements for all the provided embodiments of our models, on various devices.

Table 4. VOXReality models and their inference-time (ms) across various file formats, weight-quantization schemes and host device-types.

Model \ Configuration	FP32-CPU	FP32-GPU	FP16-GPU	ONNX-FP32-CPU	ONNX-FP32-GPU	ONNX-FP16-GPU
rgb_language_cap	4217 ms	1591 ms	578 ms	4007 ms	1693 ms	498 ms
rgb_language_vqa	525 ms	73 ms	56 ms	318 ms	128 ms	48 ms
src_ctx_and_term_nllb_600M	532 ms	98 ms	65 ms	525 ms	206 ms	110 ms
src_ctx_aware_nllb_600M	514 ms	327 ms	66 ms	517 ms	125 ms	57 ms
whisper-small-el-finetune	1636 ms	182 ms	156 ms	1310 ms	657 ms	168 ms
whisper-small-el-adapters	1767 ms	817 ms	175 ms	1291 ms	436 ms	223 ms
vit-gpt2-image-captioning	609 ms	100 ms	77 ms	478 ms	124 ms	72 ms

⁸ <https://gitlab.com/horizon-europe-voxreality/model-training-and-inference-optimization/post-training-optimization-tool>

⁹ <https://huggingface.co/voxreality>

src_ctx_aware_nllb_1.3B	1073 ms	386 ms	145 ms	1036 ms	202 ms	91 ms
t5_nlu_intent_recognition	179 ms	59 ms	52 ms	135 ms	55 ms	42 ms
nllb-asr-synthetic-robust	539 ms	105 ms	66 ms	506 ms	95 ms	51 ms
navqa	2335 ms	939 ms	397 ms	485 ms	109 ms	85 ms

Models24

^ Collapse

Sort: Recently updated

<div><div></div><div>voxreality/src_ctx_aware_nllb_1.3B_onnx</div><div>Updated 15 days ago · 1</div></div>	<div><div></div><div>voxreality/rgb_language_cap_onnx</div><div>Updated 28 days ago · 2</div></div>
<div><div></div><div>voxreality/src_ctx_aware_nllb_600M_onnx</div><div>Updated 28 days ago · 3</div></div>	<div><div></div><div>voxreality/whisper-small-el-finetune-onnx</div><div>Updated 28 days ago · 3</div></div>
<div><div></div><div>voxreality/t5_nlu_intent_recognition_onnx</div><div>Updated 28 days ago · 27</div></div>	<div><div></div><div>voxreality/rgb_language_vqa_onnx</div><div>Updated 28 days ago</div></div>
<div><div></div><div>voxreality/src_ctx_and_term_nllb_600M_onnx</div><div>Updated 28 days ago · 4</div></div>	<div><div></div><div>voxreality/whisper-small-el-adapters-onnx</div><div>Updated 28 days ago · 4</div></div>
<div><div></div><div>voxreality/nllb-asr-synthetic-robust-onnx</div><div>Updated 28 days ago · 5</div></div>	<div><div></div><div>voxreality/navqa_onnx</div><div>Updated 28 days ago</div></div>

Figure 14. The VOXReality HuggingFace repository.






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 utilized 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” [16]. 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 was automated through GitLab’s CI/CD pipeline, which had been configured by VOXReality to enable the automation of integration and deployment procedures. The pipeline utilized 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” [17].

Moreover, the trained models were already deployed across three distinct use cases: Virtual Reality Conference, Augmented Reality 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 were provided outlining various deployment options. Additionally, all developed VOXReality AI models were 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 of VOXReality in the development server was 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” [17].

Automatic deployment to the development server was 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 automated packaging and prepared 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 included the creation of the Docker file in the root of the repository. An example of Dockerfile is presented in Figure 15.

```
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 15. Example of Dockerfile.

GitLab CI/CD Pipeline

The CI/CD pipeline was defined by the `.gitlab-ci.yaml` configuration file, which specifies the stages and jobs that make up the CI/CD pipeline. The file was 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 was specified by navigating to **Settings > CI/CD > Variables** in the repository. A new variable could be added as it is depicted in Figure 16. All the repository variables are visualized in Figure 17.

Figure 16. GitLab CI/CD Add variable.

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more.](#)

Variables can have several attributes. [Learn more.](#)

- Protected:** Only exposed to protected branches or protected tags.
- Masked:** Hidden in job logs. Must match masking requirements.
- Expanded:** Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables </> 1

Reveal values

Add variable

↑ Key	Value	Attributes	Environments	Actions
DOCKER_IMAGE	*****	Expanded	All (default)	<div></div> <div></div>

Group variables (inherited)

These variables are inherited from the parent group.

CI/CD Variables </> 6

Key	Attributes	Environments	Group
SSH_PORT	Expanded	All (default)	Horizon Europe VOXReality
DOCKER_PASSWORD	Expanded	All (default)	Horizon Europe VOXReality
DOCKER_USER	Expanded	All (default)	Horizon Europe VOXReality
SSH_SERVER_IP	Expanded	All (default)	Horizon Europe VOXReality
SSH_USER	Expanded	All (default)	Horizon Europe VOXReality
SSH_PRIVATE_KEY	File Expanded	All (default)	Horizon Europe VOXReality

Figure 17. GitLab CI/CD Repository Variables.

Template .gitlab-ci.yml file

In the provided `.gitlab-ci.yml` template file (Figure 18), the only necessary modification was to adjust the final command in the deploy stage to suit the specific application-model. This last command was responsible for running the application as a Docker container:

```
docker run -d -p 8080:80 -name $SCI_PROJECT_NAME $DOCKER_IMAGE:$SCI_COMMIT_TAG
```

The container name was the variable `$SCI_PROJECT_NAME`, which was 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 [[ "$SCI_COMMIT_BRANCH" == "$SCI_DEFAULT_BRANCH" ]]; then
      tag=""
      echo "Running on default branch '$SCI_DEFAULT_BRANCH': tag = 'latest'"
    else
      tag="$SCI_COMMIT_REF_SLUG"
      echo "Running on branch '$SCI_COMMIT_BRANCH': tag = $tag"
    fi
  - docker build --pull -t "$DOCKER_IMAGE${tag}" .
  - docker push "$DOCKER_IMAGE${tag}"
  rules:
  - if: "$SCI_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 18. Template of .gitlab-ci.yml file

Trigger GitLab CI/CD

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

- When **pushed to main**, the first two stages were triggered as shown in Figure 19. The two phases were the build and test. The *build* job also pushed the \$DOCKER_IMAGE to DockerHub with a tag **latest**.

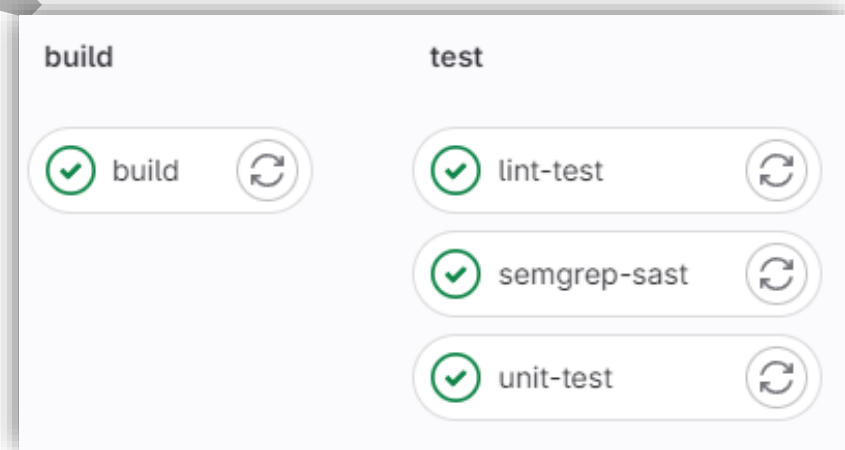


Figure 19. GitLab CI/CD Pipeline when push to main.

- When **created or pushed a tag**, all stages were triggered, including build, test and deploy. A new tag was created by navigating to **code > tag > new tag**, as it is illustrated in [Figure 20](#). The *build-tags* job also pushed the \$DOCKER_IMAGE to DockerHub with tags **latest** and **\$CI_COMMIT_TAG**, which was the commit tag name (e.g., v0.0.1). It was recommended using only Semantic Versioning. The *deploy* job connected to deployment server, pulled the docker image with the specified tag and run the docker run command to start the container. It also removed the previous version of the container if exists. All the triggered jobs are displayed in [Figure 21](#).

Figure 20. GitLab CI/CD Create a new tag.

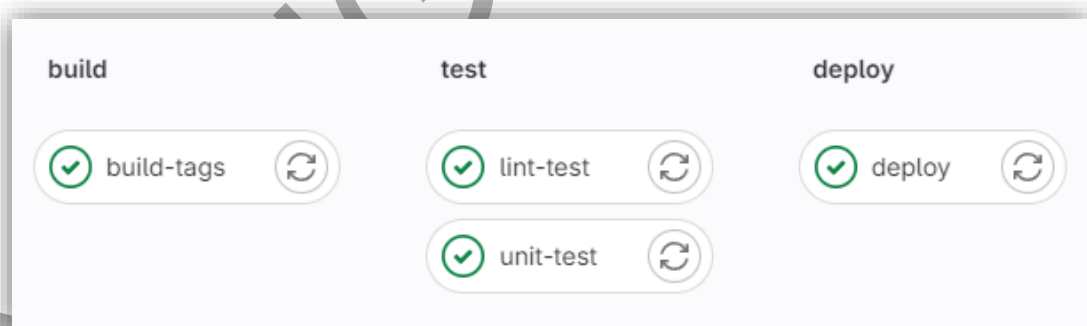


Figure 21. GitLab CI/CD Pipeline when create a new tag.

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 was created in the root of the repository to define the environment in which the application will run.
2. **CI/CD Configuration.** A `.gitlab-ci.yml` file was also placed in the root of the repository. This configuration file defined how the GitLab Runner executed the CI/CD jobs, orchestrating the build and the deployment process.
3. **Security Measures** (Optionally). Enabled the Static Application Security Testing (SAST) to analyze the code for known vulnerabilities.

4. **Repository Variables.** A repository variable named `$DOCKER_IMAGE` was specified under *Settings > CI/CD > Variables*, which was used in the pipeline to refer to the Docker Image.
5. **CI/CD Pipeline Execution.** GitLab CI/CD pipeline was triggered by either pushing to the main branch or creating or pushing a new tag:
 - a. **Push to main.** The *build* and *test* jobs were triggered. After a successful build, the Docker image, denoted as `$DOCKER_IMAGE`, is pushed to DockerHub with the 'latest' tag.
 - b. **Create or push a tag.** When a new tag was created or pushed the *build*, *test* and *deploy* jobs were triggered. This was achieved by `code> tag > new tag`. The Docker image was then built and sent to DockerHub. Following this, the corresponding container started 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 was following, which was the RESTful architecture as it is described in VOXReality Integration Guidelines of D2.3 “Development Infrastructure and Integration Guidelines” [17]. Specifically, FastAPI¹⁰, a modern, fast web framework for building APIs in Python, was employed. In this approach, each service exposes one or more endpoints to which clients can send requests. These endpoints were essentially URLs through which the services were accessible. The endpoints served as the interface for the service, allowing clients to interact with it using HTTP methods. Therefore, the deployment of VOXReality AI models was 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” [16]. Additionally, to enhance scalability, these applications could be containerized using Docker.

The VOXReality AI models could be deployed in various hardware environments following different deployment methods. This section provides general guidelines that were 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

¹⁰ <https://fastapi.tiangolo.com/>

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's 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 22.

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 direct 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.

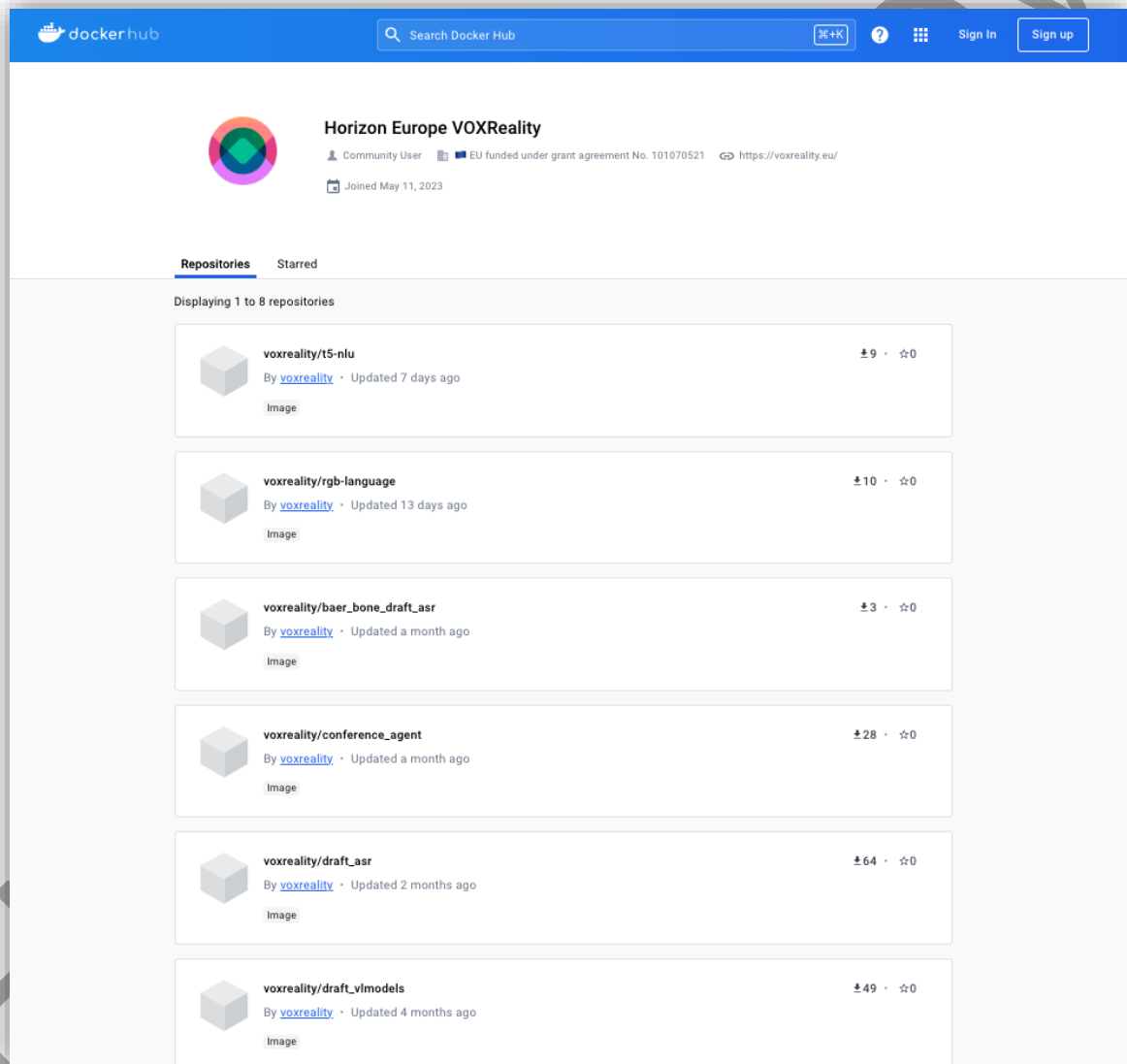


Figure 22. VOXReality DockerHub.

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 23 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 23. 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 24](#) 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.

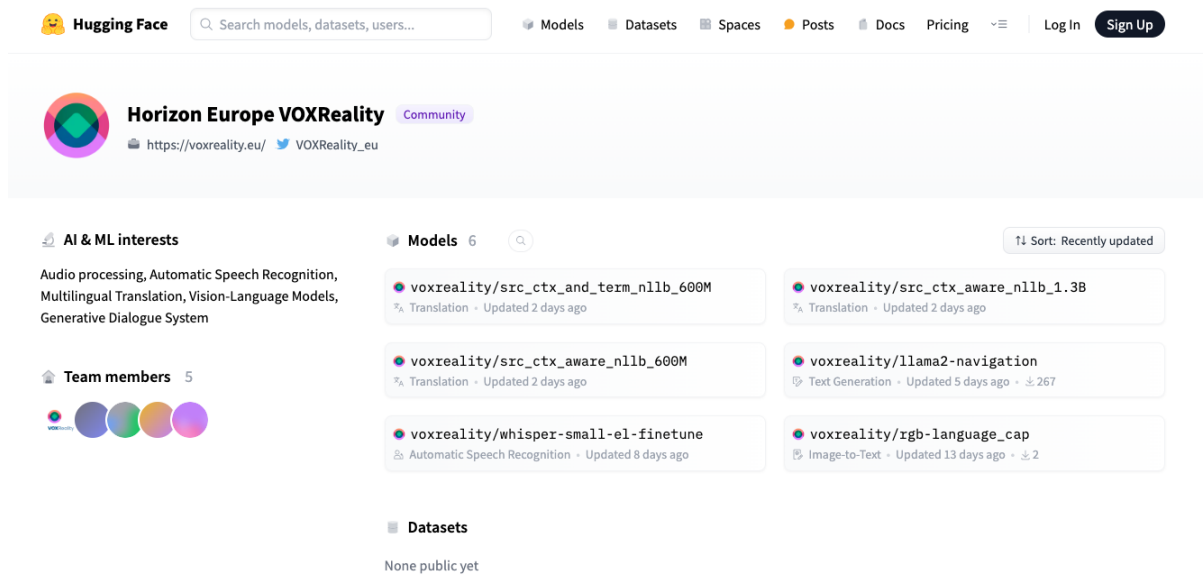


Figure 24. VOXReality Hugging Face repository.

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 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: Virtual Reality Conference, Augmented Reality Theater and Training Assistant. Table 5 presents the VOXReality components integrated into each use case.

Table 5. VOXReality components used in each use case.

Use Case \ Components	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 MX. 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 were met, while the detailed description will be provided in deliverables of WP2 and WP5.

4.1 Virtual Reality (VR) Conference

The VR Conference application emulates the 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 was designed to simulate a real venue in a 3D virtual world, incorporating all essential areas required for hosting a professional conference. It was decided that five distinct rooms would be included to represent typical spaces found in a physical conference venue. These rooms were implemented as follows:

1. The **Lobby Room**, serving as the entrance to the main area of the venue.
2. The **Trade Shows Area**, acting as the main space of the conference, where exhibitors' booths are placed and access to all other rooms is provided.
3. The **Business Room**, intended for one-to-one communication between participants.
4. The **Social Area**, designed for many-to-many communication.
5. The **Conference Room**, where the main sessions are held, supporting one-to-many communication.

To improve performance and reduce loading times, each room was implemented as a separate 3D scene. Navigation between scenes was achieved through virtual doors that connect the individual areas.

The architectural design of each room aligns with its intended use and functional requirements. Spaces dedicated to specific purposes were kept minimal to reduce distractions, whereas multi-purpose areas, such as the Trade Shows Area, feature increased dimensions and complexity. The design promotes the appropriate communication style in each area: one-to-one interactions are encouraged through the use of tables and seating arrangements, while one-to-many communication is supported by amphitheater-style layouts or the presence of a stage.

All 3D models integrated into the application were created following a consistent set of principles. Their size was minimized as much as possible by reducing geometric complexity and lowering texture resolution, in order to control loading delays. The acceptable threshold for this optimization process was carefully balanced against the need to maintain a realistic visual output. To achieve this, a low-polygon aesthetic was adopted, embracing a simpler, lightweight visual style that favors less complex geometrical forms.

4.1.1.2 Application Workflow Diagram

The core functionalities of the application presented to users can be grouped into two main categories. The first category encompasses the communication system between the user and their Virtual Agent, while the second outlines the pipeline supporting the real-time translation system.

Virtual Agent Functionalities

The Virtual Agent is represented by a virtual avatar and designed as a non-intrusive entity, meaning interaction is initiated solely by the user. Communication begins when the user activates the agent by pressing the corresponding toggle button. Once instantiated, the Virtual Agent greets the participant with a welcome message in their language and remains available to assist them throughout their navigation in the VR environment. The workflow is illustrated in Figure 25.

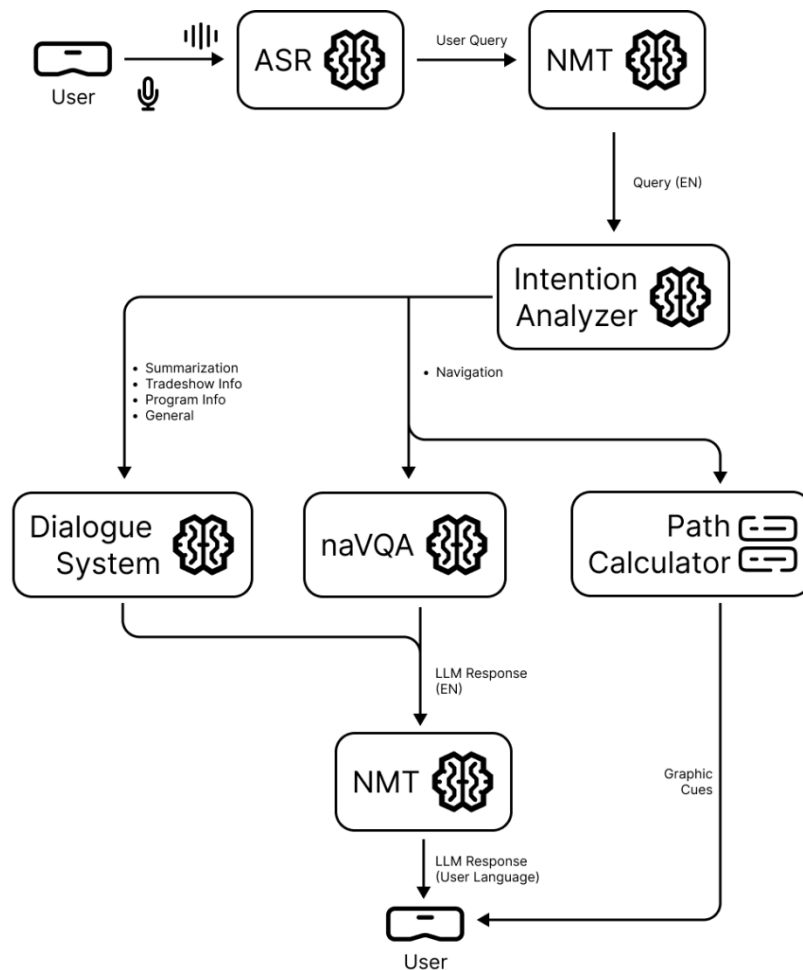


Figure 25. Workflow for communication with the Virtual Agent.

Users can interact with the Virtual Agent using a push-to-talk mechanism that records their voice while the Agent is active. During this time, the microphone output is isolated and not broadcasted to other participants, ensuring a private conversation.

Once the user finishes speaking, the system formats the recorded audio into a WAV file and sends it—along with the user's language and the target language (which is English at this stage of the workflow)—as parameters to the **Translate Audio** endpoint. This service performs both **Automatic Speech Recognition (ASR)** and **Neural Machine Translation (NMT)**.

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. Summary of the presentation
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. Depicted also in [Figure 25](#), for all user intentions except navigation, the human like response is generated directly by the Dialogue Agent component. When the user requests for navigation, their query is propagated to the NaVQA component, responsible for generating correct navigation instructions in a human like format. Accompanied by this response, the application also generates some visual cues for the desired destination, from a system based on the Dijkstra algorithm. 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 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 Training Data

In order to produce the large amounts of ground-truth data needed for the training of the NaVQA model, we devised the following deterministic mechanism, composed of various heuristics and an implementation of the Dijkstra shortest-path finding algorithm. Having already implemented a suitable mechanism, we chose to build upon the navigation system of the previous version of the application that was explained comprehensively in D4.1. In comparison to the previous implementation, a more robust and rich system has been employed this time, generating fluent, human-like textual data describing the navigation steps between two specified start and end points. The output steps composing the NaVQA's training dataset are essentially the product of a navigation route calculated as the optimal solution to a path-finding problem and are subsequently forwarded to the proposed LLM to respond with human-like directional instructions. The navigation model is explained in detail in VOXReality D3.2 – “Advanced AI multi-model for XR analysis”, chapter 3.3.6.

When the application is launched, the navigation system 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 navigation request. The procedure of claiming the shortest path between two points of the scene is effectively managed by employing a version of Dijkstra's path-finding algorithm. Dijkstra's algorithm requires the existence of a weighted graph mapping the environment, physical or virtual, accompanied by information regarding the position of the start and end points. The nodes of the graph, in this proposal, are generated heuristically, by examining if points of an orthogonal field with a predefined distance between them, are placed inside the borders of the environment while not intersecting with other 3D objects. Being a virtual conference, the showcased environment consists of different rooms of the venue. Both the rooms and their objects, treated as obstacles, can be represented as arrays of 2D vectors, assuming that the floor and ceiling levels are known. An estimation of a point's location relative to a polygon is provided by the parity of the number of intersections between the corresponding point and a fixed known point outside the room. As depicted in [Figure 26](#), odd parity suggests that the point is enclosed by the polygon whereas even parity implies the opposite.

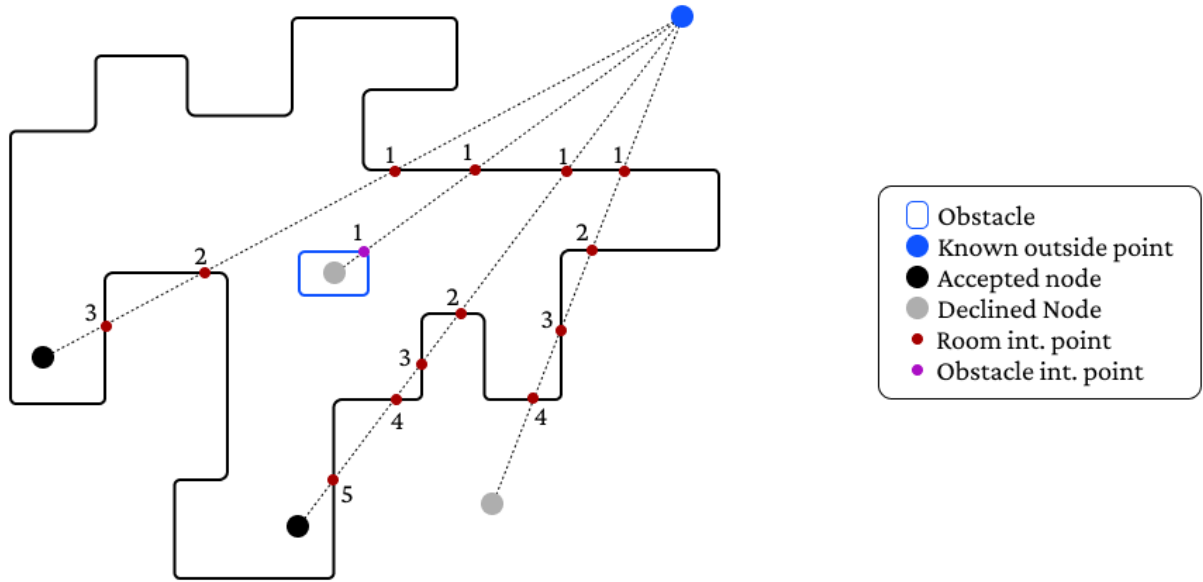


Figure 26. Workflow for communication with the Virtual Agent.

Adjacency between the fabricated nodes is successively resolved following the formula below:

$$Adj = (X_{n1} == X_{n2} \mid |Y_{n1} == Y_{n2}) \&\& D_{Manh}^{n1 \rightarrow n2} < D_{field}$$

X_n, Y_n : Node coordinates

$D_{Manh}^{n1 \rightarrow n2}$: Manhattan distance between nodes,

D_{field} : Field spacing distance

This formula prohibits navigation paths from containing diagonal lines and ensures edges will only connect nodes distant from one another. Furthermore, the weight of each edge matches the Manhattan distance of the nodes it connects to. Additions to the nodes and graphs are applied throughout the development of the graph, such as including nodes to represent the destination points and edges to connect remote nodes to their nearest node. A representation of a produced weighted graph is shown in

Figure 27.

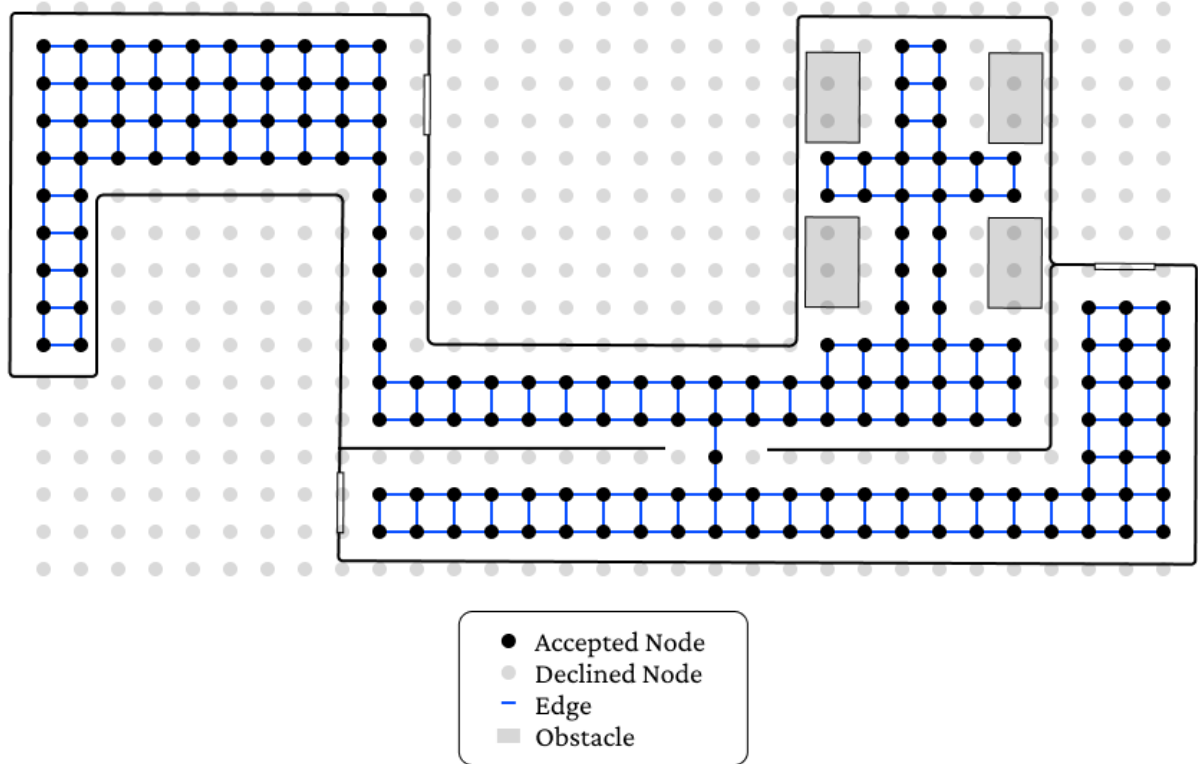


Figure 27. Weighted Graph with Manhattan Distance and Non-Diagonal Connectivity

The geometrical properties of the produced weighted graphs, such as corridors, space openings and corners, are calculated in this respective order. An analytical algorithm determines all straight-line paths of the graph and distinguishes those that meet a predefined width-to-length ratio criterion. Nodes of such paths are classified as corridor members and are excluded from further geometrical analysis. Corners of the virtual space can be defined by identifying nodes which interfere between two consecutive corridors. All nodes that are neither classified as corridor members nor as corner members should geometrically be part of an opening space.

Figure 28 visualizes the workflow of handling requests for navigation from the end user, from the application's perspective. The position of the user inside the 3D scene is accessible by the application, thus, once a request for navigation is recognized by the LLM, the start node is selected to be the nearest to the actual user's position. Consequently, the name of the destination binds to the information about the user's intention by the LLM module, from which the corresponding node is extracted using a lookup table. If a valid destination point is requested, the Dijkstra algorithm forges a sequence of nodes connecting the two points, in the most efficient way, otherwise the application terminates the user's request with a predefined answer.

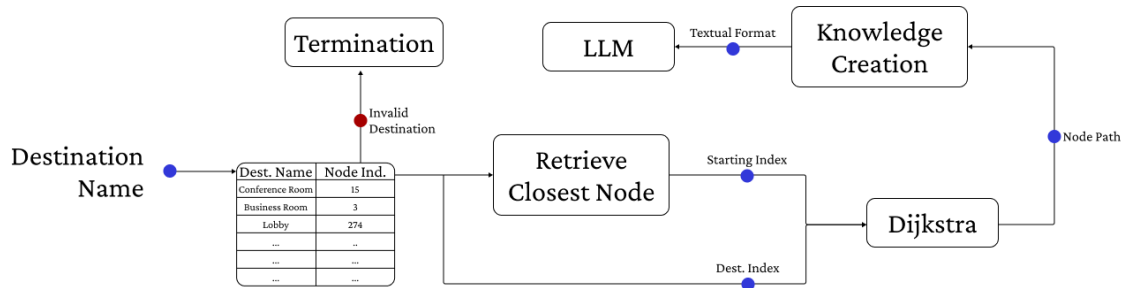


Figure 28. Workflow of Navigation Request Handling Using LLM and Dijkstra's Algorithm

Once a valid node path for the requested destination point is produced, the path is sliced into line segments for further analysis. During translation of the user from the starting to the ending point of a line segment the closest, visible object of each node is collected. For each object the distances and the angles between the examined segment and the line that connects the object with every node of it are calculated. The way those values progress along the path indicates whether the user crosses an object, is moving towards it or arrives at it. This is further explained in Figure 29.

A valid connecting path is transformed into a format valuable for the LLM. Taking into consideration the change in direction during the user's translation along the path, the relative position between each consecutive path node and the objects of the environment and its geometrical properties, an intermediate text following the format below for each line of the path is generated:

`cross [object, direction], pass through [room property], arrive [object, direction]`

An example of such a generated intermediate text:

start, cross chair left, arrive wall, turn left, cross mirror, pass through wall opening, arrive destination left, finish

As a final step, this textual information bundled together with the destination point is propagated to the LLM module in a request to receive human-like navigating instructions. This process took place for every node of the Tradeshow area, and for many different orientations of the user (Body orientation, Head tilt). The resulting training dataset for the NaVQA component consists of pairs of the human like instruction text together with the respective image captured from the user's point of view (PoV).

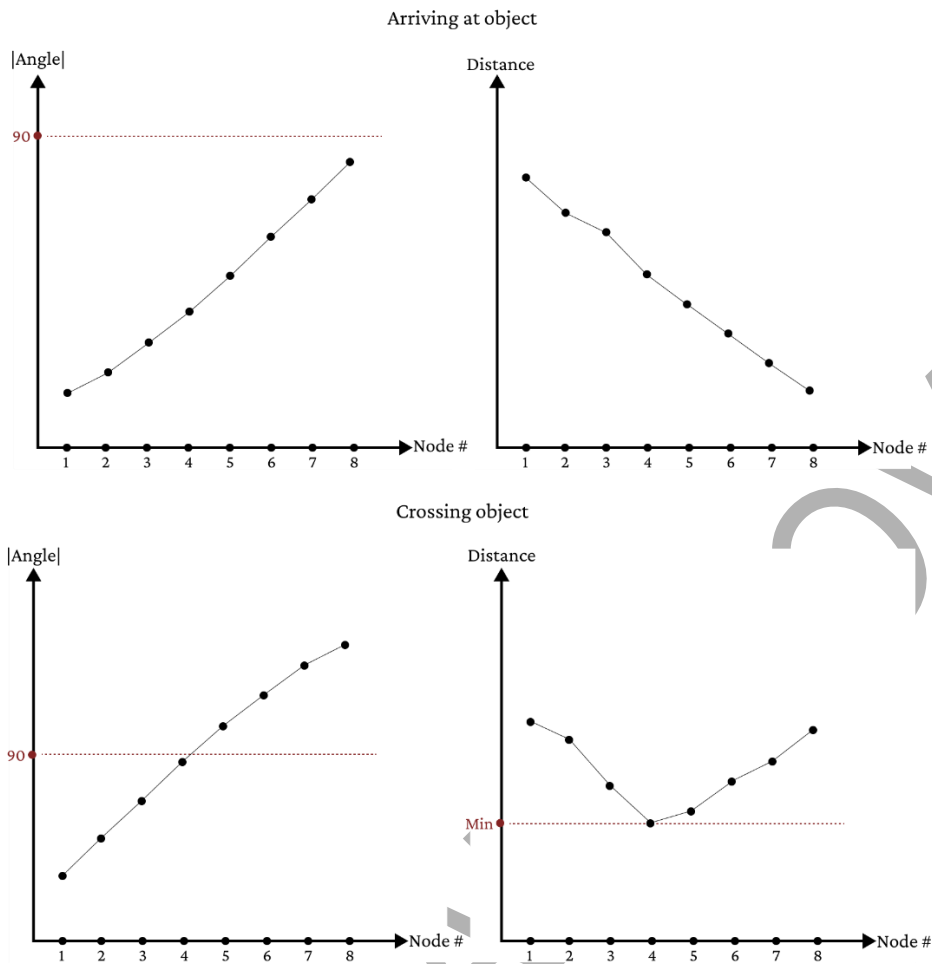


Figure 29. The way Angle and Distance change as the user arrives-to or crossing an object. The correct detection of these two events is crucial for the creation of the navigation dataset.

Translation system (One-to-one and Many-to-many)

The real-time translation system is not managed by the virtual agent entity. In spaces that allow communication between participants (Social Area and Business Room) translation system is activated in stand-by mode.

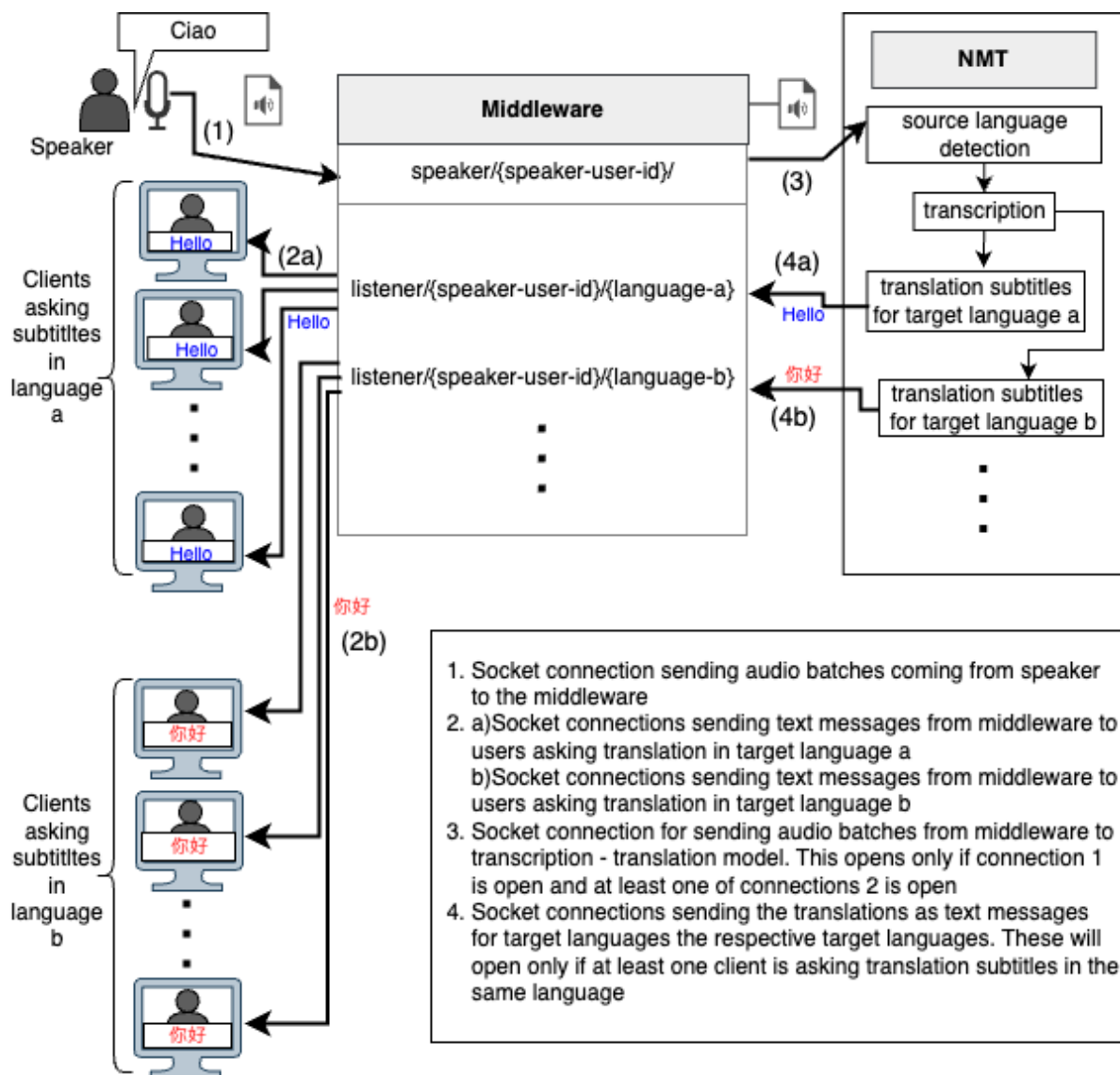


Figure 30. Workflow of the translation system.

The mechanism for initiating translation between users has been enhanced with a modular and resource-efficient architecture (Figure 30) tailored to real-time communication scenarios. When a participant activates their microphone, a WebSocket connection is first established to the endpoint: `/speaker/{speaker-user-id}`. This marks the user as an active speaker and makes their audio stream available for translation. Only after this speaker-side connection is open can other users initiate a listening session by connecting to: `/listener/{speaker-user-id}/language` where `{language}` specifies the desired translation output (e.g., it for Italian, es for Spanish).

Once both a speaker and at least one listener are connected, an **intermediate socket layer** activates a third connection to a **transcription-translation model**, hosted on a separate WebSocket service. The model receives batched audio streams from the speaker, performs automatic language detection, transcribes the content, and dynamically generates text translations for each requested language.

The architecture is optimized for GPU and computational efficiency:

- One transcription process is shared per speaker, avoiding duplicate audio processing.
- Translation is executed once per language, regardless of how many users request it.

- This means unlimited listeners per language can receive real-time translations without additional processing overhead.

Translated messages are delivered to each listener through the `/listener/{speaker-user-id}/{language}` socket. Upon receipt, the translated text appears in the user interface, displayed in a panel directly below the avatar of the speaker (Figure 30) , maintaining spatial and conversational clarity.

Moreover, the architecture **naturally supports multiple speakers** operating in parallel. Each speaker is identified by a unique user ID (`/speaker/{speaker-user-id}`), allowing **concurrent translation sessions** to run independently for different speakers, each with their own set of listeners and translation requirements.

The Translation System is designed to respond dynamically to user actions. When a **speaker deactivates their microphone**, the `/speaker/` socket is closed. When **listeners deselect translation**, their respective `/listener/` sockets are also closed. If no active listener remains for a speaker, the intermediate connection to the transcription-translation model is automatically terminated, conserving system resources.

By decoupling the transcription model from the speaker-listener communication flow, and activating inference only when necessary (i.e., when both a speaker and at least one language-specific listener are present), the system minimizes redundant computation and scales effectively under load.

Presentation System (One-to-many translation, Q&A Session)

The Presentation System manages real-time translation for both the presenter and the Q&A session. It determines the presenter based on spatial positioning within the conference room. Specifically, when a user moves to the stage area, they are automatically marked as the presenter. The system ensures that only one presenter is active at a time—if another user enters the stage, no changes occur.

The system allows audience members (anyone who is not the presenter) to raise their hand to request permission to ask a verbal question. The presenter has the ability to accept or decline these requests. Only upon acceptance is the audience member granted permission to unmute their microphone and speak.

The Presentation Translation System applies the same underlying architecture as the generic Translation System described above, with minor adaptations for one-to-many communication in the Conference Room, where typically only one person speaks at a time (the presenter or an approved audience member).

When a user enters the stage area, the VR-Conference app designates them as the active presenter. At that moment she/he turns on the microphone:

The `/speaker/presentation` connection is opened.

Instead of using user IDs, the Presentation Translation System standardizes the speaker endpoint as `/speaker/presentation` and listener endpoints as `/listener/presentation/{language}`.

If one of the auditors asks for translation by clicking on the translate button in their panel, a socket is established so as to connect to `/listener/presenter/{language}` based on their selected subtitle language.

The intermediate socket layer connects to the model only when both the speaker (the presenter or one of the auditors) and at least another user (the presenter or one of the auditors) has activated translation.

The translated subtitles of the current speaker are displayed for the auditors in real-time above the stage on a large screen. For the presenter the subtitles are displayed in front of his camera view, like the movie subtitles.

All information exchanges—such as hand-raise requests, transcriptions, and event changes—are managed through the Dialog server.

4.1.2 Implementation Details

4.1.2.1 Development Environment Setup

The VR Conference application was developed based on Hubs¹¹, an open-source web application for interacting in networked 3D spaces, powered by Mozilla. For the development process, the **Community Edition** of Mozilla Hubs was hosted on a **Kubernetes cluster**, which automatically managed all the necessary services of the application. The cluster ran on Google Kubernetes Engine¹², using a **2x2vCPU** virtual machine with **4 GB of RAM**. In parallel, minor changes to the application were applied directly using Mozilla's **Development Server** to reduce 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 Headsets 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 and testing, Meta Quest 2 and Meta Quest 3¹³ were used. Due to compatibility reasons streaming the application to the headsets required a MS Windows OS client computer.

The server side of the application, named Reticulum, is written in the Phoenix framework of the Elixir programming language. Reticulum handles all networking functionality except for communication media (e.g., microphone and camera), which are managed by a Node.js-based webRTC server built on the open source MediaSoup project, called Dialog. Another worth mentioning project, that is part of the application, is Spoke, a lightweight **JavaScript-based scene editor** used for creating 3D environments.

All additional features in Mozilla Hubs were developed by modifying and extending the existing Hubs client code, which was written in a mix of JavaScript and TypeScript. The user interface was developed using *React.js*, while the 3D scenes were built using a combination of *THREE.js* and *Networked A-FRAME*, a framework that wraps *THREE.js* components. All physics simulations in the 3D world were handled by the *ammo.js* library.

¹¹ <https://hubs.mozilla.com/>

¹² <https://cloud.google.com/kubernetes-engine>

¹³ <https://www.meta.com/quest/quest-3/>



4.1.2.2 3D Models and Scene Creation

The virtual environment of the application was designed primarily using **Blender**, an open-source 3D creation suite, to sculpt geometry and customize the shading of 3D objects. The creation of scenes representing the various rooms of the virtual conference benefited from the **Archimesh extension**, an architectural modeling tool in Blender that enabled simple and geometrically lightweight room designs (Figure 31).

Decorative elements of the environment, including objects required for each room—such as the booths in the Trade Show Area—were either sourced from **freely licensed 3D model libraries** available online or **created specifically** for the VR conference application.

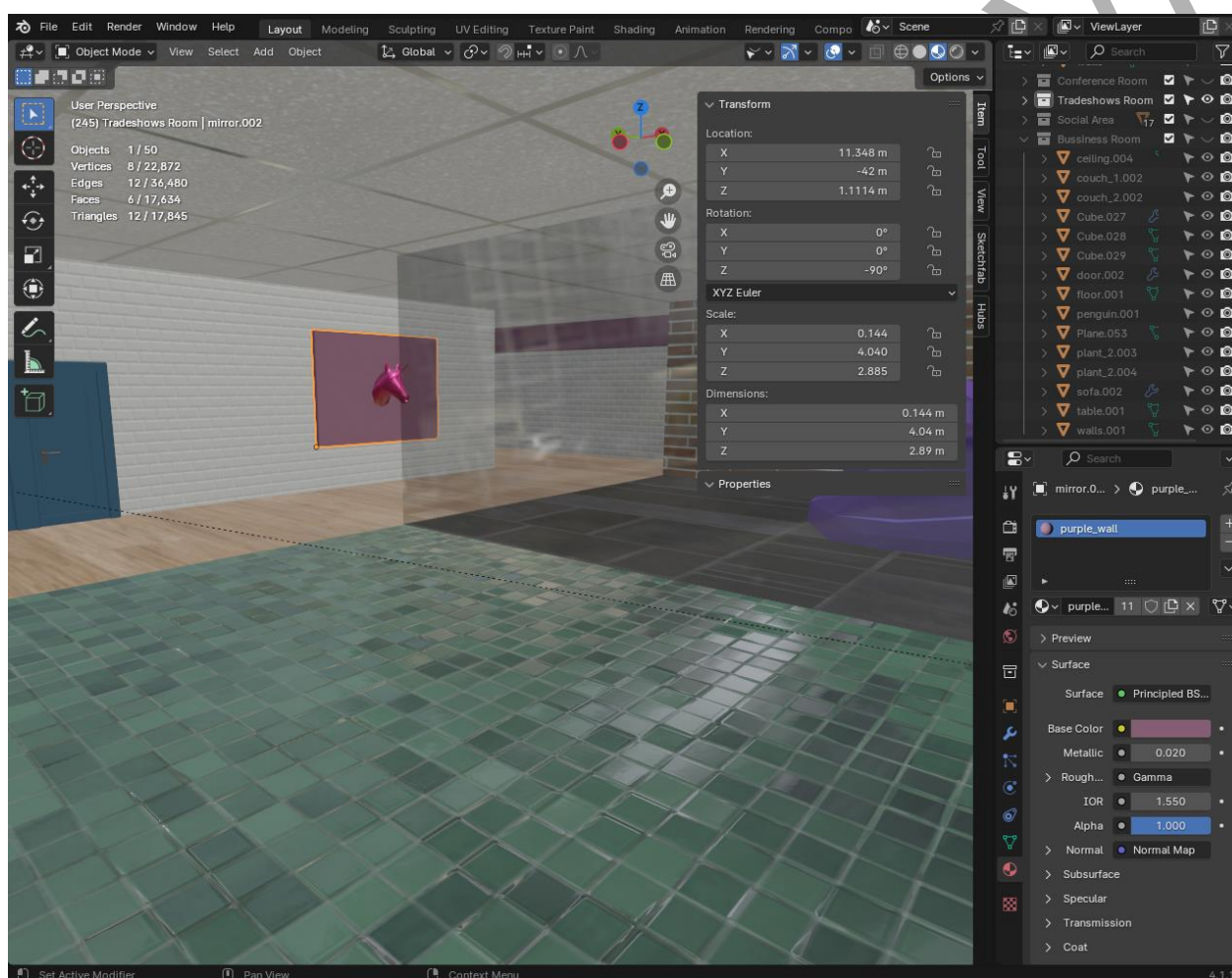


Figure 31. Blender Interface for room designing.

Shading elements without predefined materials was carried out using freely licensed texture files sourced from the internet. New textures were created and assigned to the corresponding 3D models as needed (Figure 32). To maintain optimal runtime performance, all textures were kept at medium to low resolution.

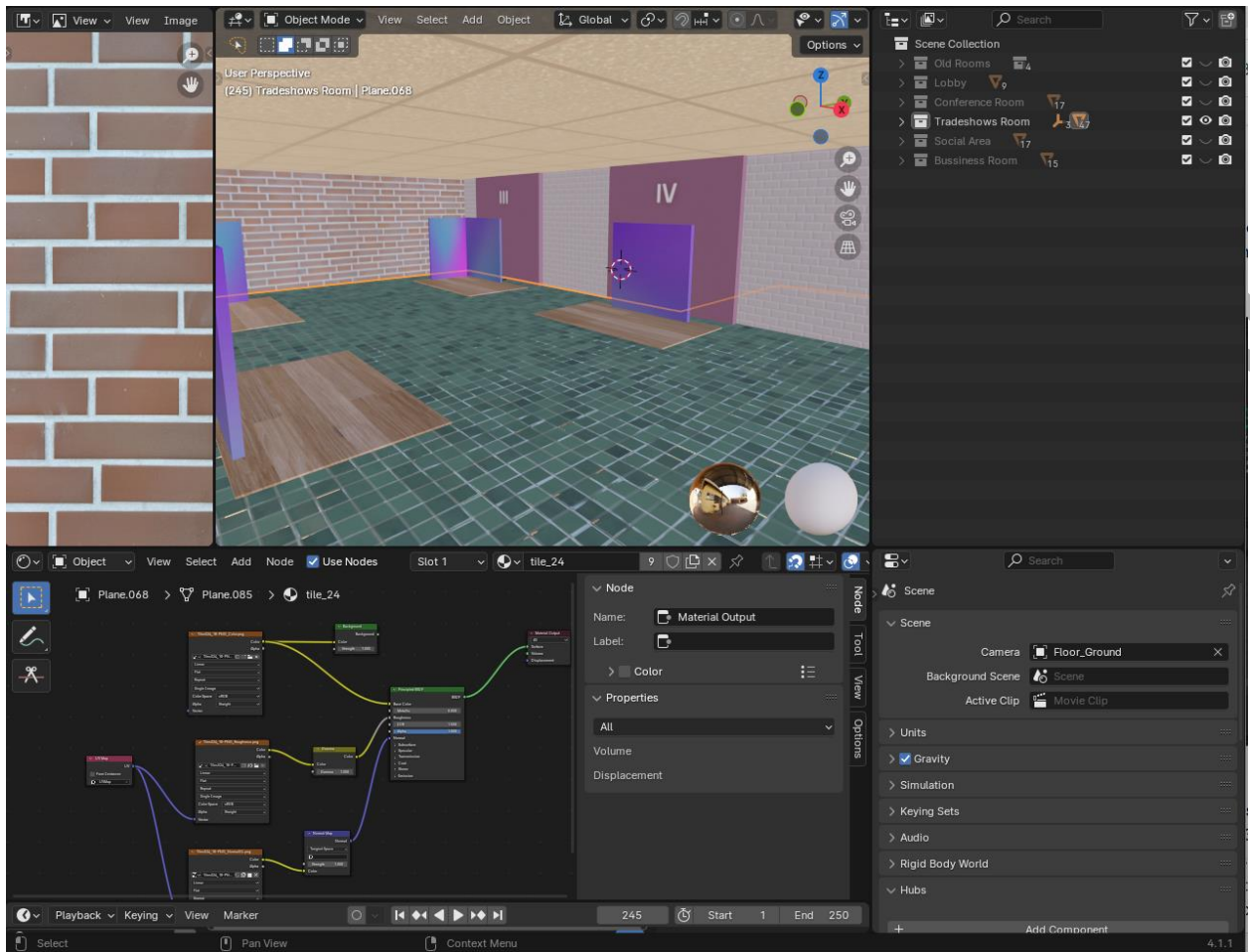


Figure 32. Blender Interface for shading.

Once a space was completed in terms of design and shading, the models were exported as a single file using Blender's GLTF/GLB exporter, since GLB is the only supported format for 3D model importing in Hubs. Final editing took place in Spoke, where the 3D model was imported into a new scene and supplemented with lighting sources (Figure 33). Additional Hubs-specific components—such as connection gates to other preconfigured rooms and user spawn points—were added to the scene before it was published.

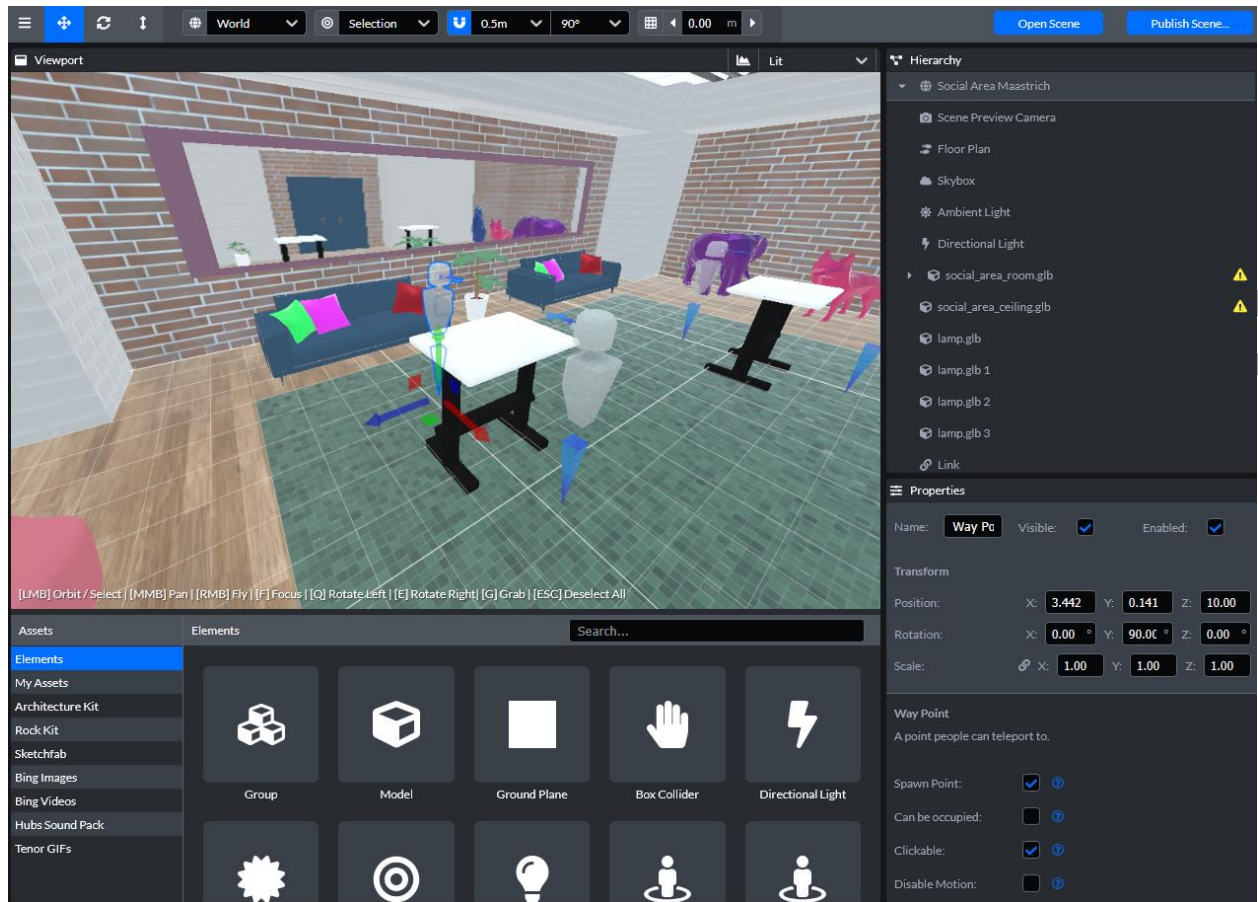


Figure 33. Mozilla's Spoke Interface.

When a participant enters 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, all different VR conference rooms have been created associated respectively with the Lobby Room, the Trade Show Area, the Conference Room, the Business Room, and the Social Area 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 [18], 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.

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 its 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 displaying output once available. Additional functionality was 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 could not be avoided.

User Panel

The user panel (

Figure 34) 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. From the user panel, it is possible to enable/disable the presence of the Virtual Agent in the 3D environment, the Map component, the help slides, and during the main presentation ask for a question or enable/disable subtitles. 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 icon is not shown.

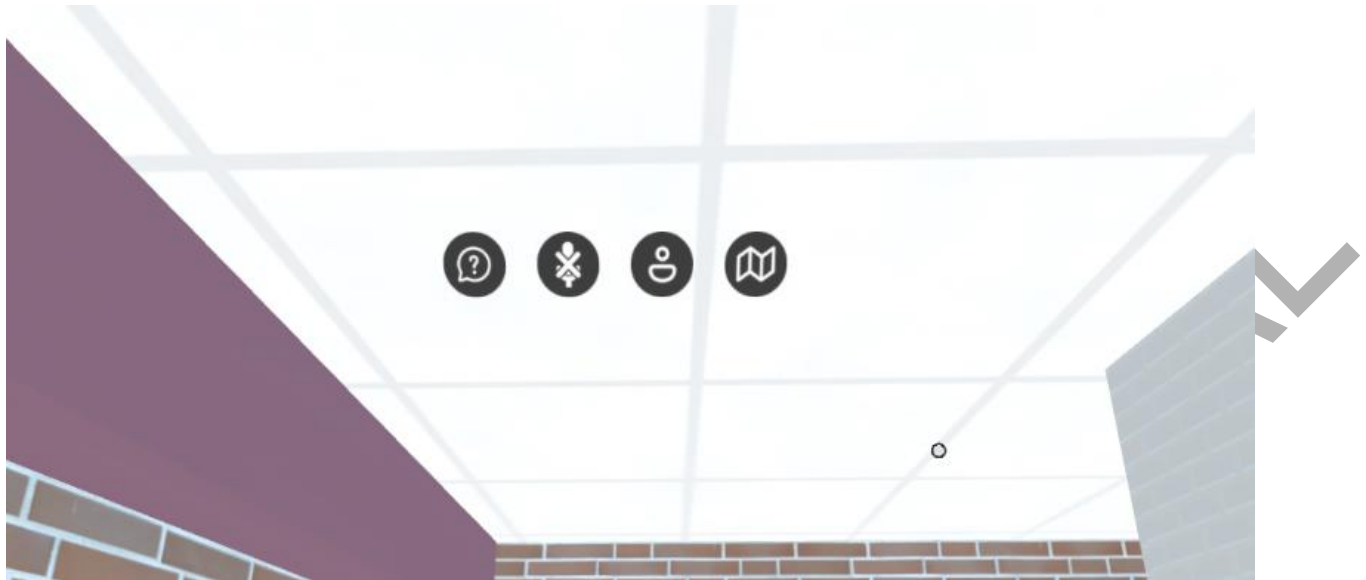


Figure 34. User Panel element.

Map Component

If enabled from the user panel, a 3D map renders in the 3D scene, configured to change its position and orientation according to the user's. The Map panel (Figure 35) 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.

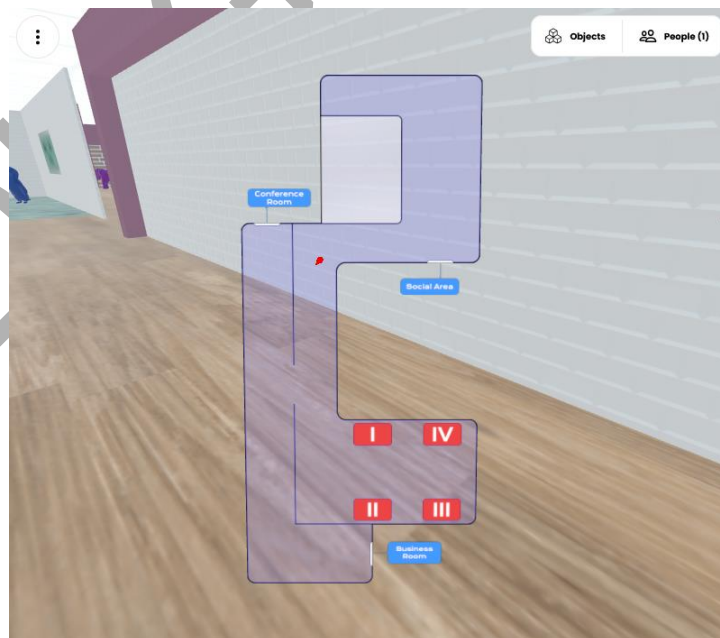


Figure 35. Map Component for the Trade Show Area.

Help Panel

Behaving in a similar way to the Map component, when selected from the User Panel, a Help Panel (

Figure 36) that provides help slides on how to interact with the application, is appended to the 3D scene. By clicking on the arrow buttons, the user can move to the next or previous slide. These slides are accessible from all rooms of the application.

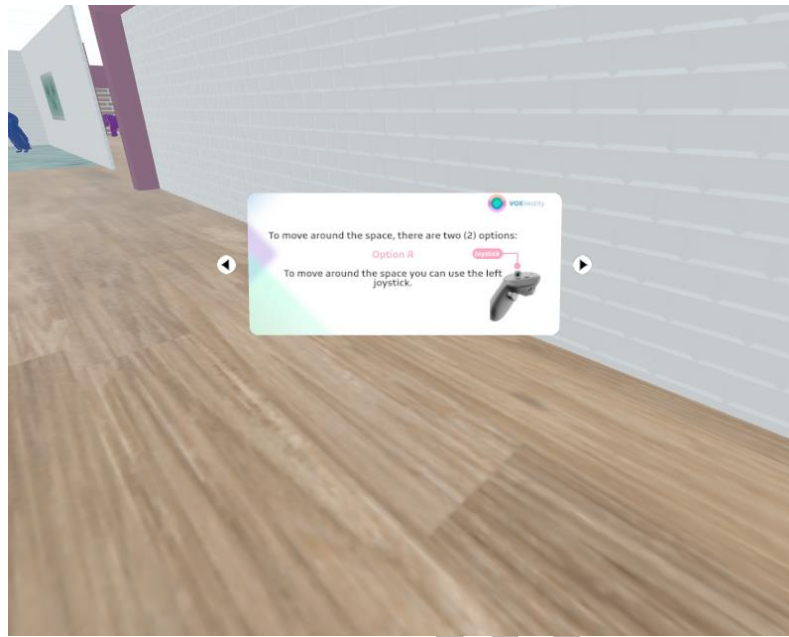


Figure 36. Help slides.

Translate Button & Panel

In rooms where the application permits one-to-one or many-to-many 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 on top of every avatar's head with the translation icon appears in the 3D scene (Figure 37). Pressing this button, it 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. When translation is activated for a user, the translation button transforms to a cancelation button that by pressing it, disables the translation for this specific target. If a target is selected, a text panel (Figure 38 Right) contains initially a phrase in the user's language, informing them that the translation output will appear on this element.



Figure 37. Translate Button element.

Lobby Tutorial Slides

The first room users encounter upon entering the application is the lobby room. Its primary purpose is to help users familiarize themselves with the VR space and learn the application's controls. To achieve this, a dedicated interactive tutorial system was developed.

The tutorial is presented through a series of slides containing images and GIFs, and it is displayed in the user's selected language. Users are guided through essential tasks such as moving, teleporting, rotating, interacting with the user panel, and engaging with the virtual agent.

A built-in task detection system monitors user actions and verifies when each task is successfully completed. Once a user finishes the tutorial, they have the option to either restart it from the beginning or proceed to the next room, the main area of the application.

Virtual Agent

When the virtual agent (VOXY) is enabled, a text panel appears in front of it, displaying the textual output of the dialogue system ([Figure 38 Left](#)). The information that gets displayed on this panel contains greeting messages, responses of the dialogue agent module and potential error messages. The panel is scrollable,

maintaining a fixed size, and users can navigate through the text using the scroll wheel (in laptop mode) or the controllers (in VR mode).

Above the agent, up to two suggestion buttons may appear. These buttons are generated by a suggestion system that tracks the user's progress and remaining tasks, providing relevant queries for the virtual agent. When a user selects a suggested query, it triggers the same process as a voice command but bypasses the transcription model. Once a suggestion is used, it is marked as resolved and will not be shown again.

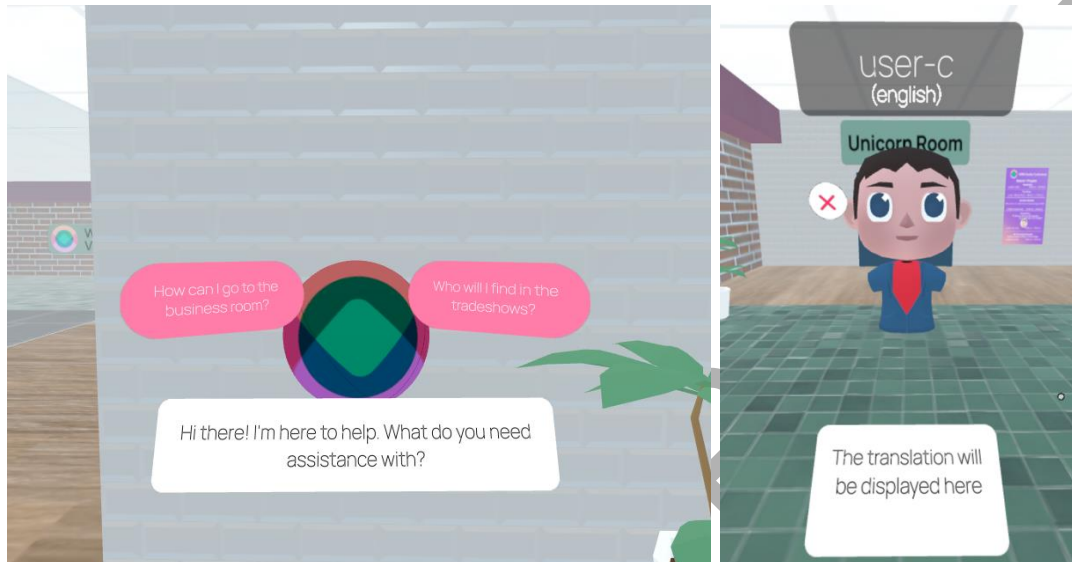


Figure 38. Virtual Agent Panel (Left), Translation Panel (Right).

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.

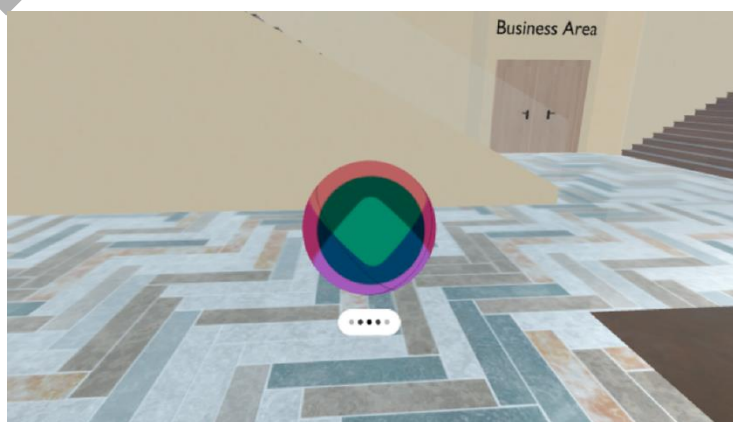


Figure 39. Loading animation.

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 fulfilled 40 of the 49 user requirements for the VR Conference application, including all of the 32 high-priority, 7 of the 10 medium-priority and one of low-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 in the form of drawn lines on the floor, guiding users to their destination. Furthermore, the system provides real-time translation in five languages (German, Dutch, Italian, Spanish and Greek). 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.

Table 6. VR Conference. Achievement of user requirements

#	Type	Requirements	Priority	Final Status	Reasoning
1	General	The experience is in VR.	High	YES	
2	General	Provide user navigation and language translation at the virtual conference.	High	YES	
3	General	Develop a virtual assistant providing users with relevant information to navigate virtual spaces, interact with other visitors, and exchange relevant information.	High	YES	
4	General	The operational output language is English, and it can be translated into other languages.	High	YES	
5	Scenario	The conference venue should imitate the environment of a professional conference setting.	High	YES	
6	Scenario	The users will be represented as virtual avatars.	High	YES	
7	Scenario	The virtual avatars of the participants will offer predefined set of gestures.	High	YES	
8	Scenario	The presentation should be in a predefined conference format.	High	YES	

#	Type	Requirements	Priority	Final Status	Reasoning
9	Scenario	The suggested duration for a single VR session is 30 minutes at maximum.	High	YES	
10	Assessment	The target user includes conference partners and visitors with diverse backgrounds.	High	YES	
11	Assessment	The suggested number of participants per session is ~10 to 20 people.	Medium	YES	
12	Assessment	The quality of experience will be assessed by a questionnaire followed by a semi-structured interview.	High	YES	
13	Virtual Agent	A dedicated virtual agent will be assigned to each user and will stay with him/her during the complete duration of the conference.	High	YES	
14	Virtual Agent	The virtual agent should provide welcome greeting.	High	YES	
15	Virtual Agent	The virtual agent should provide help, FAQs and prompt the users about the context/content of the venue and potential to-do (action) items in advance or relative to the current activity.	High	YES	
16	Virtual Agent	The virtual agent will help user to navigate the space, answer programme-related questions, and deliver relevant information.	High	YES	
17	Virtual Agent	The users should have a "skip" option for the virtual assistant help.	High	YES	
18	Virtual Agent	The communication with virtual agent should be enabled by voice typing.	High	YES	
19	Virtual Agent	The communication with virtual agent should be enabled by text.	Low	NO	Effort was allocated to oral communication instead
20	Virtual Agent	Virtual agents deliver personalized info feeds related to the conference programme-related activities.	Medium	NO	Focus was placed on high-priority requirements instead.
21	Virtual Agent	The virtual assistant can be accessible via a smart-watch / wrist-band.	Low	NO	Focus was placed on high-priority requirements instead.
22	Virtual Agent	The virtual-assistant should auto-save relevant information and provide them in an exportable file.	Medium	NO	Focus was placed on high-priority requirements instead.

#	Type	Requirements	Priority	Final Status	Reasoning
23	Virtual Agent	The virtual agent should present users with most relevant	High	YES	

		available options at the end of every session.			
24	Virtual Agent	The virtual agent should interact with the users on-demand, and should not be intrusive.	High	YES	
25	Virtual Agent	Virtual agent should be likeable, friendly, pleasant, customizable, realistic and complete.	Low	NO	Focus was placed on high-priority requirements instead.
26	Virtual Agent	Virtual agent could look like a cartoony avatar.	Medium	YES	
27	Navigation	Quick get started tutorial option should be available.	High	YES	
28	Navigation	Virtual map should be available to navigate the conference venue.	High	YES	
29	Navigation	Flyover view of the venue should be available with zoom in/out options.	Low	YES	
30	Navigation	Navigation to the target location should be guided by visual cues, arrows, marks/lines on floor etc depending on the context.	High	YES	
31	Navigation	Users should be assisted for quick and easy navigation between different places in the venue.	High	YES	
32	Subtitles	Translation should be available in textual format.	High	YES	
33	Subtitles	Translation should be available in voice (audible) format.	Low	NO	Focus was placed on high-priority requirements instead.
34	Subtitles	Option to mute the complete room or individual speaker(s) should be available.	High	YES	
35	Subtitles	Users should be able to control the volume +/-	High	YES	
36	Subtitles	Users should be able to turn on/off the program, subtitles, auto-translation and voice using interactive buttons.	Medium	YES	
37	Subtitles	For multi-speakers, the subtitles should highlight and differentiate the active speaker among group of speakers.	Medium	YES	
38	Subtitles	The standard/default subtitles should be displayed on the bottom of the screen and in particular cases the subtitles should be placed relative to the virtual environment context.	High	YES	

#	Type	Requirements	Priority	Final Status
---	------	--------------	----------	--------------

39	Subtitles	For head-mounted display, the subtitles should be visible and placed without masking other important parts in the display/screen.	High	YES	
40	Interface	There should be a customizable dashboard visible at all times.	Medium	YES	
41	Interface	A help button should be visible on the corner of the screen.	High	YES	
42	Interface	In the conference room, users can choose the in-room view options i.e., full screen and/OR conference room view.	Medium	NO	Focus was placed on high-priority requirements instead.
43	Interface	Questions/feedback from the audience should appear textually/visually near the presenter's screen.	Medium	YES	
44	User Interaction	Users should be able to engage with speakers and guests during Q&A session(s).	High	YES	
45	User Interaction	Users can interact with the agent and ask the agent questions during the navigation.	High	YES	
46	User Interaction	Users should have options to use hand-gestures to interact with the system.	Low	NO	Focus was placed on high-priority requirements instead.
47	User Interaction	Users should have options to use interactive-buttons on the screen to interact with the system.	Medium	YES	
48	Extra	Dialogue Agent should be available on demand.	High	YES	
49	Extra	Users should be able to make digital and crypto payments in the conference items shop.	Low	NO	Focus was placed on high-priority requirements instead.

4.2 Augmented Reality Theatre

The Augmented Reality Theatre application is intended for use in AR-enhanced theatrical performances, providing audience members with personalized augmented reality audiovisual and textual content to address both accessibility and entertainment needs. According to gathered user requirements, the application must deliver timely, translated captions in the user's preferred language (high priority) and trigger visual effects (VFX) at specific moments in the performance, based on both verbal and visual cues predefined by the theatrical director (high priority). Smooth onboarding for non-experienced users and extensive personalization support were also required.

Given that the application's primary aim is to provide a synchronized, but also personalized experience across the audience, a server-client architecture has been implemented, which

can ensure centralized control, quality assurance, and synchronization. The server communicates with the VOXReality services and distributes the appropriate messages to the clients in the audience as illustrated in Figure 40.

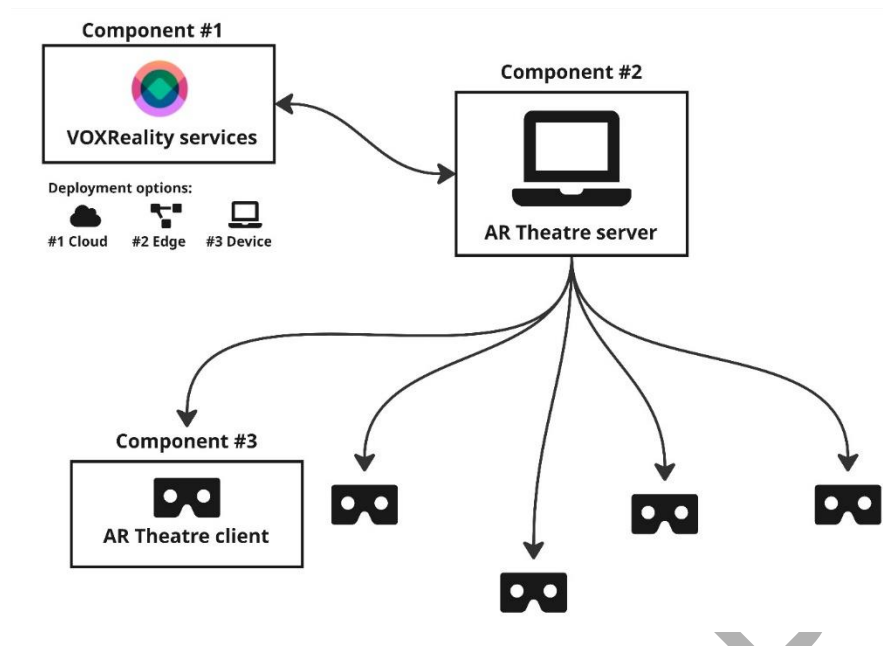


Figure 40. AR Theatre system - high level overview

In more detail, the server receives live audio input from the actors via microphones, generates transcripts using VOXReality speech processing models, and streams the resulting caption to each client. The AR Theatre use case adopts a streaming and matching approach for the transcripts to adhere to the high-quality standards of artistic text. In parallel, a visual feed from a camera is processed to produce scene descriptions via VOXReality vision-language models. The scene descriptions themselves are not streamed to the clients but rather used by dedicated logic in the server to detect the occurrence of specific stage events - like the appearance of an actor on stage. Finally, both the received transcript and the detected stage events are processed on the server-side as triggering conditions for sending keywords to the AR clients, which activate associated audiovisual effects (VFX) when received, as illustrated in Figure 41.

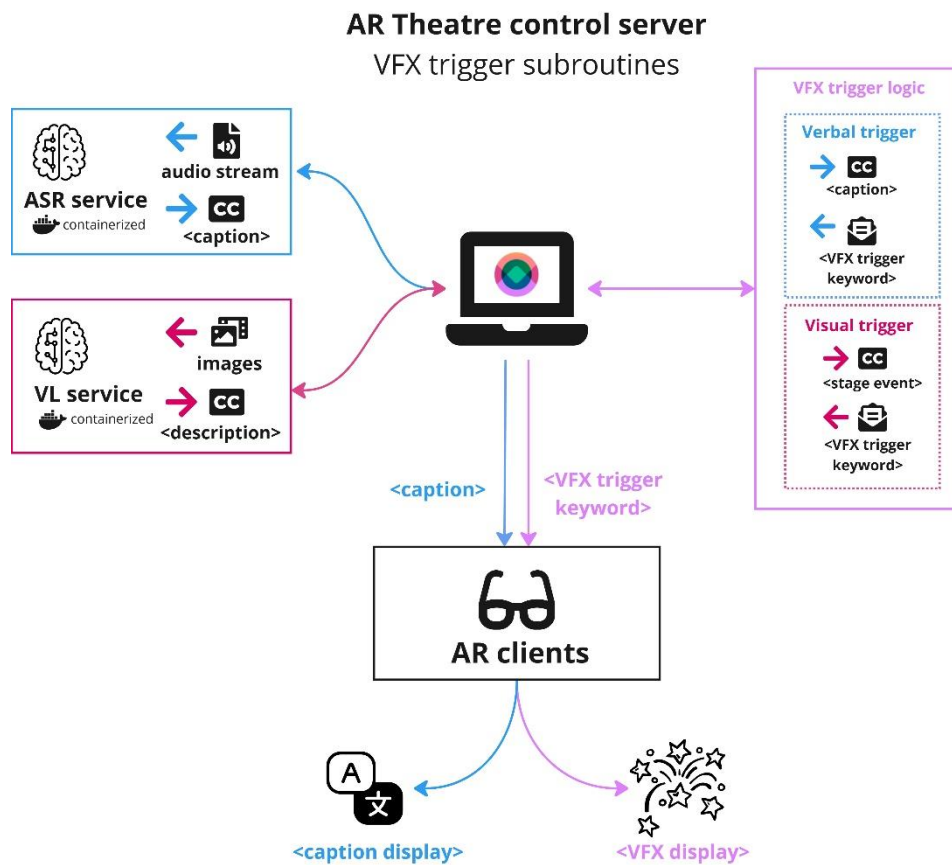


Figure 41. VFX triggering methods using verbal and visual cues

On the client side, the captions can be translated based on the user's language preferences. In the first version of the system, translations were generated in real time via HTTP requests from the clients to VOXReality language services. However, to minimize latency, reduce redundant simultaneous requests, and support literary quality control of the translations by the theatrical stakeholders, the second version of the system uses pre-generated (offline) and human proofread captions. The end-product has been curated by literary experts and stored as cloud resources, which AR clients retrieve during the performance to deliver linguistically appropriate content to each viewer.

The detailed architecture of the AR Theatre system, detailing the positioning of the VOXReality ASR, VL, NMT services, as well as the detailed of the network communication, is illustrated in Figure 42.

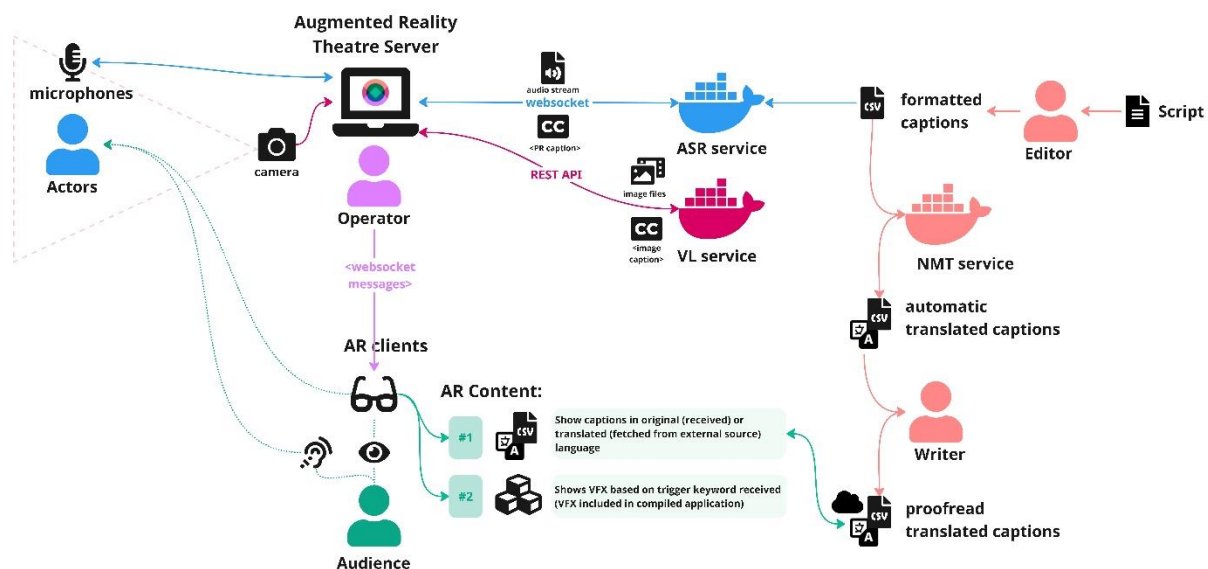


Figure 42. AR Theatre architecture - detailed version

The user requirements documented in D2.2 Definition and Analysis of VOXReality Use Cases instruct not only the system architecture, but also aspects of the AR application design. Specifically, since the application is designed for audiences with little or no prior experience with augmented reality, it must include an onboarding sequence before the performance begins. This phase should provide familiarization with AR technology, introduce key features of the application, and include a step-by-step tutorial on the interface and available customization options to ensure accessibility and ease of use. In addition, in terms of hardware requirements, the system should allow the user to view the theatrical stage overlaid with the digital 3D content with the maximum possible field of view and impose the least distractions and obstructions from the physical stage events. Furthermore, the application should operate without voice commands, in low light conditions, and with minimal requirements in terms of somatic components, like gestures. Therefore, suitable hardware options for this use case are only considered to be in the format of augmented reality glasses (e.g. not camera-based AR on smartphones) and input modalities to the application are restricted to using a dedicated controller.

4.2.1 System Architecture and Design

As presented in short above, the AR Theatre system architecture is designed as a modular, client-server framework that enables synchronized delivery of multilingual subtitles and audiovisual effects (VFX) to the end-user through wearable augmented reality devices. The system is comprised of three components: (i) the AR client, deployed on AR headsets; (ii) the control server, deployed on a laptop device, operated by the venue staff; and (iii) the AI inference services, hosted within Docker containers and accessed via APIs, with multiple deployment options.

The **AR application** runs as a native Unity-based XR experience on the Magic Leap 2 AR device, which supports controller-based input, optical see-through with a large field of view with waveguide rendering. The AR application renders captions, VFX, and graphical user interface elements. Each AR application subscribes to a WebSocket-based communication channel with the control server, enabling real-time synchronization and remote configuration. Thus, the AR application is foremostly controlled by the audience but can also be remote controlled by the operator for synchronization purposes, such as announcing the start of the play. User preferences, such as language and caption display settings, are locally stored, do

not persist across sessions, and are applied at runtime, thus allowing for personalized experience for multiple changing users.

The **control server** serves as a command hub and status monitor. Operators can remotely initiate performance phases, trigger director-defined cues for VFX overlays, and monitor headset connectivity and system health. The detailed functionality of the control server and the respective user interface is presented in section AR Theater control server4.2.2.4.1.

The server communicates with the **VOXReality AI components** – i.e. the Automatic Speech Recognition (ASR) and Vision-Language (VL) services - through containerized services. Communication with the services is configurable on the user interface to accommodate flexibility in the deployment options, such as on edge or on cloud. Neural Machine Translation (NMT) is also used as part of the overall solution, specifically for offline generation of captions, and is therefore not used during the delivery of the performance and not connected to the control server.

The above system uses two more assets:

- a csv file containing the script segmented into captions, optimized for AR display. This asset is created by a human editor through a process of segmenting the theatrical play's script into caption-sized segments manually. The average length of a segment is suggested by the audio sampling length of the ASR service – for a recommended sampling duration of 3 seconds, a segment length of about six (6) words is recommended. The exact length of each segment can vary slightly based on semantics and grammatical rules of the current language. The asset is provided to the ASR service by the control server, to receive back a transcription free of any potential spelling errors and formatted to the level of detail desired by the theatrical stakeholder. The ASR streaming and matching service used in the AR theatre is described in detail in deliverable D3.1.
- a csv file containing the translations of the original captions in all the available VOXReality languages. This file is generated using the NMT service and is edited by a human editor (a professional translator) to achieve the desired literary quality and ensure human oversight of the content. This asset can be hard coded in the AR client application and can be used during the runtime for retrieval of the desired translation based on the current original caption received by the control server. Alternatively, this asset can be downloaded from a cloud location at initialization of the AR client application to allow edits to the content after application compilation for higher flexibility. Figure 43 illustrates the above-described path of information for NMT and ASR in the AR theatre use case and highlights the interplay between human and AI elements.

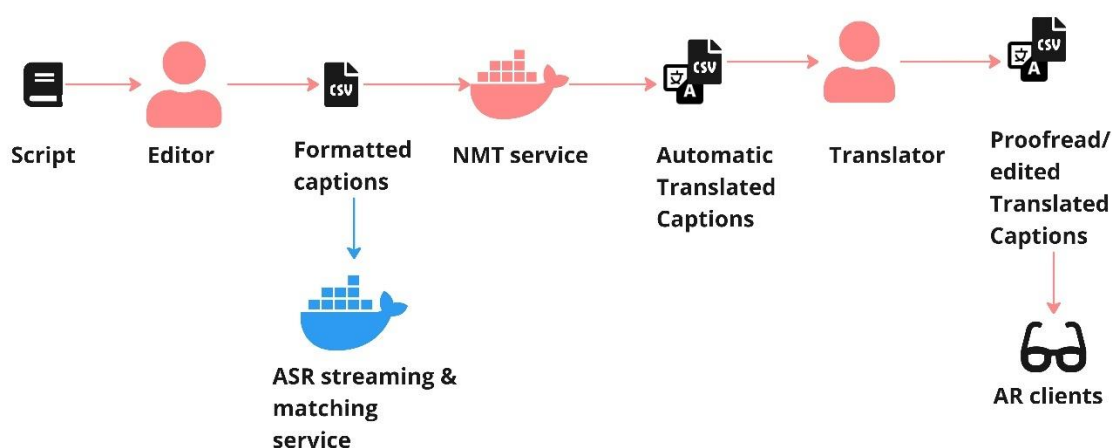


Figure 43. Use of NMT service in AR Theatre

The initial system architecture which was presented above can be extended to include one more component, the audio player client, which can respond to trigger words emitted by the control server, just like the AR clients. The audio player handles the audio dimension of the virtual effects and is connected to the audio system of the theatrical stage to produce a surrounding audio effect in sync with the virtual effects delivered through the AR glasses. This allows for a collective yet personalized audiovisual artistic experience.

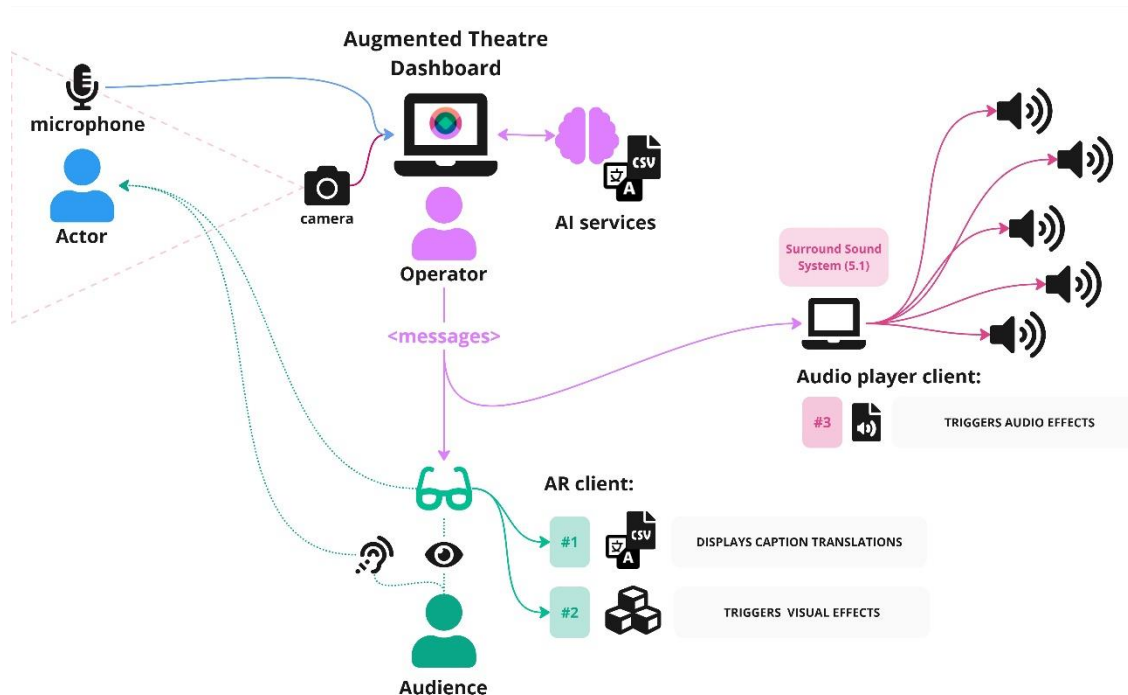


Figure 44. AR Theatre architecture - extended version

The above presented architecture has been designed to minimize the network traffic, thus accommodating scalability for diverse theatrical venues, with improved network robustness and system performance.

4.2.1.1 3D Models and Scenes Design

The only component in the AR Theatre system that has a 3D virtual environment is the AR Theatre client. The AR Theatre client application has five distinct virtual environments:

1. “Main menu”, which allows the user to choose their preferred language and access the other scenes in a recommended order.
2. “Introduction to device”, which presents the basics of the chosen AR device to the user, and demonstrates hands-on the available input methods and interaction patterns
3. “Tutorial to application”, which demonstrates in a step-by-step and hands-on approach all the features of the AR Theatre application
4. “Extras”, which contains information about the play and the performance
5. “Performance”, which is the scene of main interest hosting the audiovisual content and related functionalities
6. “Credits”, which displays the ending credits for the theatrical performance.

Scenes 1, 2 and 3 are mainly equipped with graphical user interface elements and simple 3D models for illustration and comprehension purposes, as detailed in the section 4.2.2.4.2. Scene 4 is the main performance scene, rich in visual content.

The visual content for the AR Theatre application was designed and created by a dedicated team of multimedia digital artists under the guidance and supervision of the theatrical director appointed by AEF. The design process was iterative and took the technical limitations and affordances of the AR hardware and physical stage setup into account. Initially, ideation began with hand-sketched storyboards used to align the creative vision with the theatrical director and to establish a preliminary layout of the key scenes, as sampled in Figure 45. These visual drafts provided a shared reference for both artistic intent and spatial composition.

Following this, the scenes were decomposed into a concrete asset list, with each item categorized as either a 3D or 2D element. Assets were then distributed among expert collaborators according to their respective domains—such as 3D modelling, VFX, or UI design. Prototype assets were produced to validate feasibility on the target AR platform and to assess their aesthetic integration with the live stage. These prototypes underwent refinement through iterative in situ testing with the Magic Leap 2 headset, allowing for adjustments in scale, position, animation timing, and user visibility. This process ensured that all digital augmentations harmonized with the dramaturgical flow and respected the physical affordances of the venue, as shown in a sample in Figure 46.

Parallel to this process, detailed measurements of the physical performance space were conducted to inform spatial constraints, anchor calibration, and user viewpoint considerations, with the end-result illustrated in Figure 47.

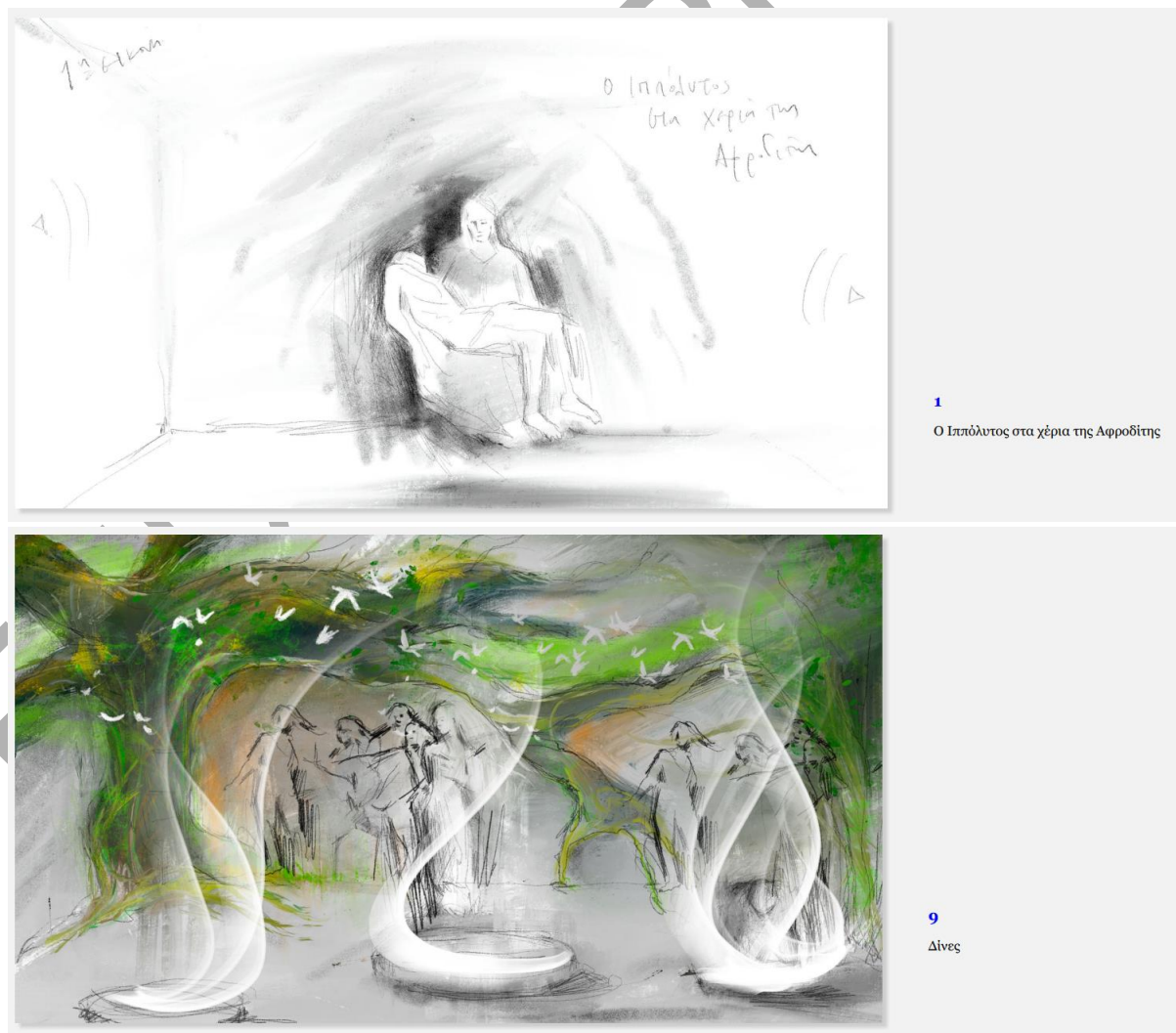


Figure 45. Storyboard excerpts



Figure 46. From storyboard to final assets over iterations

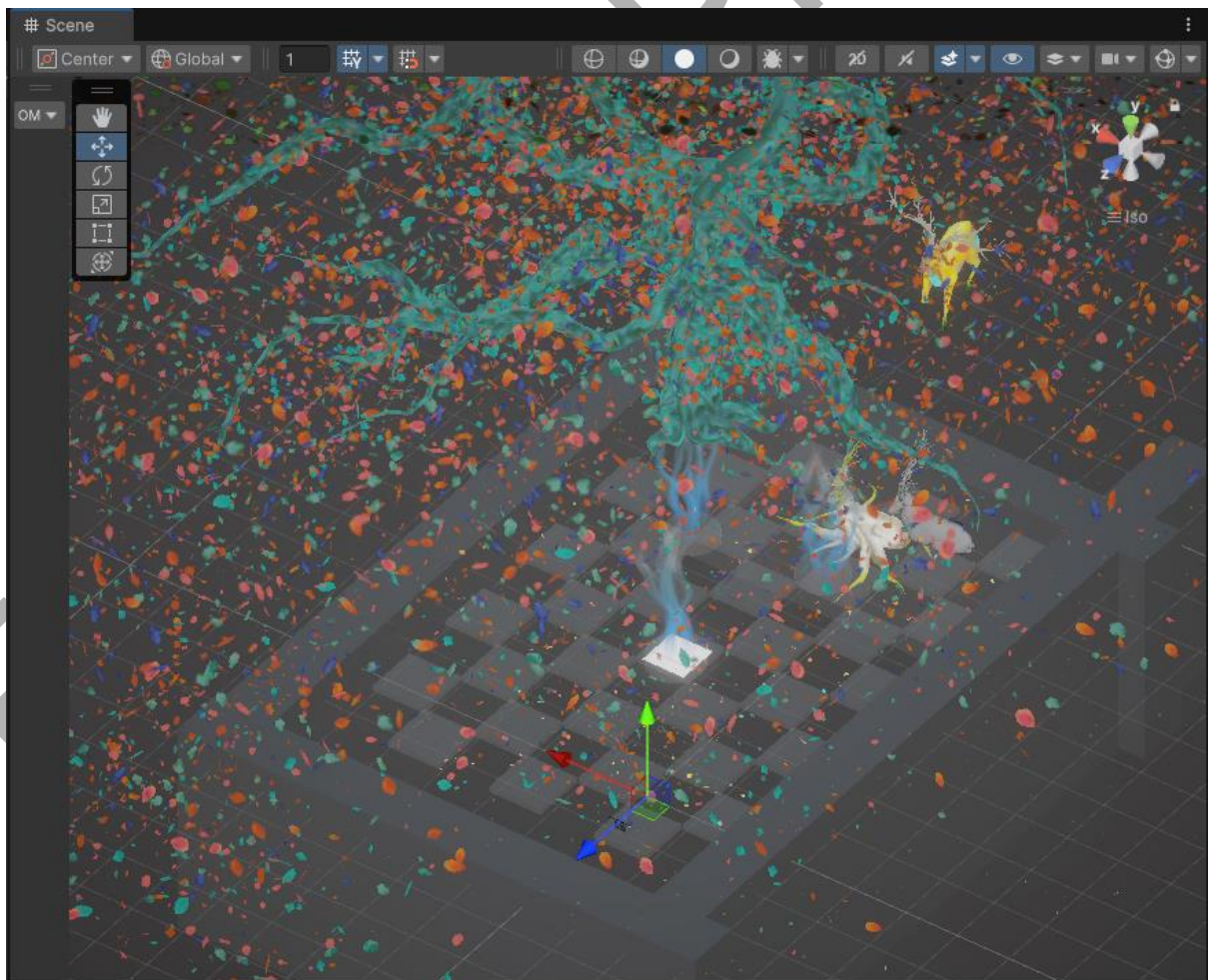


Figure 47. Matching the physical and digital space elements in the scene design

4.2.1.2 Application Workflow Diagram

The AR Theatre application follows a client-server model orchestrated through a series of coordinated runtime stages. There are two main user stories: for the venue operator, handling the control server, and the audience, handling the AR client. The system initiates with the server-side configuration phase managed by the operator.

Venue operator user journey and application flow – prep phase

The venue operator connects the microphones (wireless Bluetooth connection) and camera (wired USB connection) to the device running the control server and positions them accordingly on the stage and actors. The operator configures the communication settings with the AI services on the control server based on the current deployment, e.g. on device (localhost), on edge or on the cloud. The operator configures the ASR streaming and matching service parameters based on the current play (e.g. spoken language should be Greek) and starts the Websocket connection with the service. The operator can perform some mic tests to establish the average confidence value of the results given the microphone quality and acoustics. After microphone check is complete, the venue operator can mute the microphones until the play starts. For lab conditions with RODE Wireless ME microphones, an average value of 35 confidence rating is recommended. In addition, the venue operator can set the query interval to the VL services that will receive the camera feed. A frequency of one query every 3 seconds is recommended. Lastly, the operator can start the WebSocket connection available to the AR clients. When the performance is ready to start, the operator will unmute the microphones, start the VL querying routine, and remote control all AR clients to initiate the performance scene.

Audience user journey and application flow – prep phase

Simultaneously, the audience initializes the AR application in the Magic Leap 2 devices, which automatically upon initialization establish a secure WebSocket connection with the server. The WebSocket connection IP and port are configured remotely by the operator using the Unity Remote Config service, providing flexibility to the system network. The users are prompted first to complete a tutorial introducing the AR device components and input methods (buttons), as well as a second tutorial introducing the AR application interface and functionalities. This process should happen in advance of the performance and with instructor support, if required.

Venue operator and audience user journey and application flow – performance phase

When all members of the audience have completed the onboarding process, the performance is ready to begin, and the system enters the live synchronization phase. The operator uses the remote-control interface to initiate the AR client's performance scene, unmutes the microphones and starts the automatic photographing routine. The operator's main task after this point is to monitor the smooth performance of the system, e.g. detect any system failures or client disconnects, since the rest of the application flow is automated. Specifically, the server streams the audio feed to the ASR services and receives a message with the transcription response (caption). A filtering of the responses based on the confidence value is applied to suppress any responses triggered through background noise or improper audio stream segmentation. The approved captions are broadcast over the network to all clients, which in turn update their rendered overlays accordingly. If the user has requested for translated captions, the AR client displays instead the pre-generated translation, which is retrieved from a disk-stored resource, using the just received caption as key. This allows the entire audience to remain in sync while accommodating language personalization.

The server also automatically uses the connected camera to take photographs of the stage with the preconfigured frequency and provide them to the VL service for processing. For the piloted play, the director asked to detect the entrance and the exit of an actor on the stage.

Therefore, the desired physical stage events to look for were programmatically interpreted as changes to a VL query of “Is there a human present in this photograph?”. A change from “yes” to “no” or vice-versa provided a notification for a “actor entrance” or an “actor exit” event.

Finally, both the received captions from ASR and the stage events from the post-processing of the VL queries are checked by a subscribed subroutine called “VFX trigger manager”. This subroutine requires at initialization a “VFX plan”, i.e. csv with pairs of conditions and associated keywords. This asset can be retrieved from disk from the control server, and it can optionally be downloaded from a cloud location for fast and easy updates to the content, which is especially useful during rehearsals. Upon detecting a keyword associated with the current caption or stage event, the VFX trigger manager uses the WebSocket connection to transmit to the AR clients the associated keyword(s). These keywords are in turn associated with artistic VFX in the AR clients (or any other programmatic response).

A diagnostic loop on both client and server layers monitors system health, connection state, and logs any anomalies. In the event of a disconnection or synchronization issue, the server can issue reconnection commands or switch clients to fallback modes (e.g., preloaded subtitle track). Finally, a data logging plan was installed across the system to capture the behaviour and performance in detail, as shown in Figure 48.

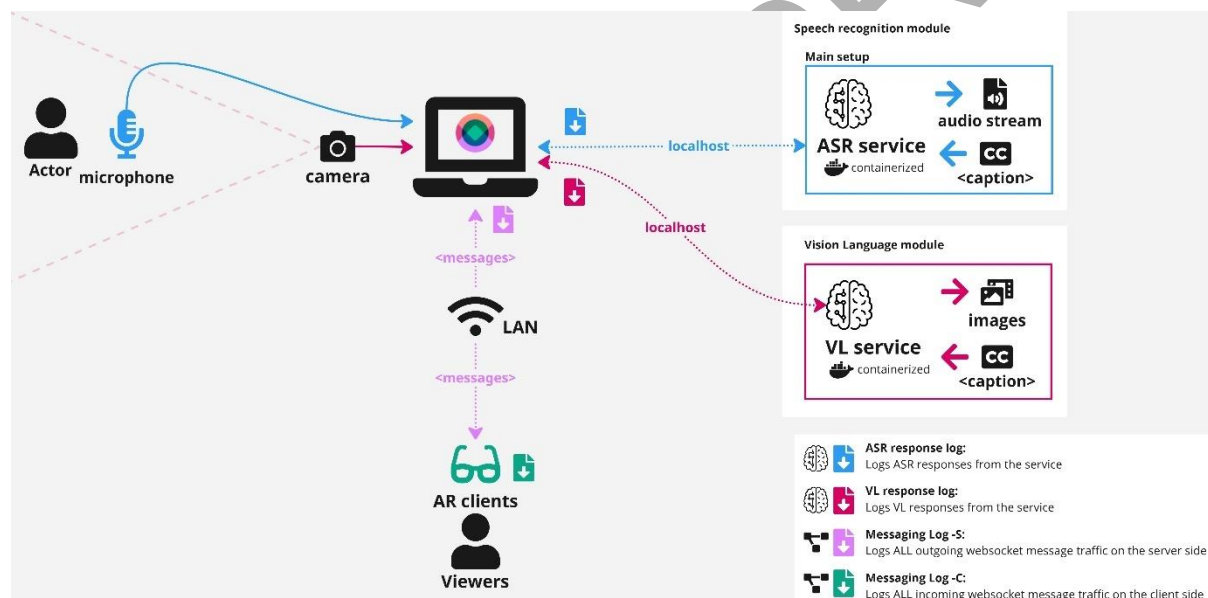


Figure 48. Data logging plan

4.2.2 Implementation Details

4.2.2.1 Development Environment Setup

The AR Theatre system comprises two main components: the control server and the AR client application, both developed within a unified technology stack centered around the Unity game engine (version 6000.031f1). This environment was selected for its broad compatibility with Magic Leap 2 hardware, support for real-time 3D rendering, and seamless integration with extended reality (XR) toolkits.

For communication between server and client, the system employs WebSocketSharp for bi-directional messaging and .NET System.Net.Http for REST-based asynchronous data exchanges. All message payloads are serialized using Newtonsoft JSON, selected for its reliability, performance, and wide adoption within .NET ecosystems. Communication between

components is secured via TLS, with certificate validation layers enabled during deployment to ensure data integrity across the transport layer.

For AR client development needs, the OpenXR framework with the Magic Leap 2 feature extension was used, as provided by the Unity XR Core package and the Magic Leap 2 SDK. The XR Interaction Toolkit was adopted for handling interactions and input abstraction. A variety of commonplace tools, like TextMeshPro for high-fidelity text rendering (notably for multilingual caption overlays), Unity Localization for localizing the user interface, Sirenix's Odin Inspector, and Unity Services' Remote Configuration, were utilized to streamline development workflows and enhance inspector usability during rapid prototyping and debugging. The Magic Leap 2 AR headset served as the primary deployment target. The build target was configured for Android x86-64 with Magic Leap-specific runtime options. The build used Unity's IL2CPP scripting backend, which ensures performance and binary-level optimization for deployment on Android-based AR systems. Android Debug Bridge (ADB) tools were used extensively during development for deploying, managing, and debugging application packages directly on the devices. In addition, the Magic Leap Hub (version 3) provided additional installation tools, firmware update support (version 1.12) and live streaming for application debugging and documentation.

From a version control perspective, the development team employed Git as the source control system, hosted on GitHub, with Git LFS enabled to manage large binary assets such as textures, models, and animation files. A conventional branching model (main, feature branches) was used to isolate development stages. All builds were versioned manually, and build artifacts were tagged and archived for traceability and reproducibility.

To support developer onboarding and reproducibility, Unity dependencies were managed via the Unity Package Manager and the NuGet package tool, and internal documentation was maintained using structured README files and diagrams in whiteboards. These resources enable transparent replication of the development environment across devices and contributors.

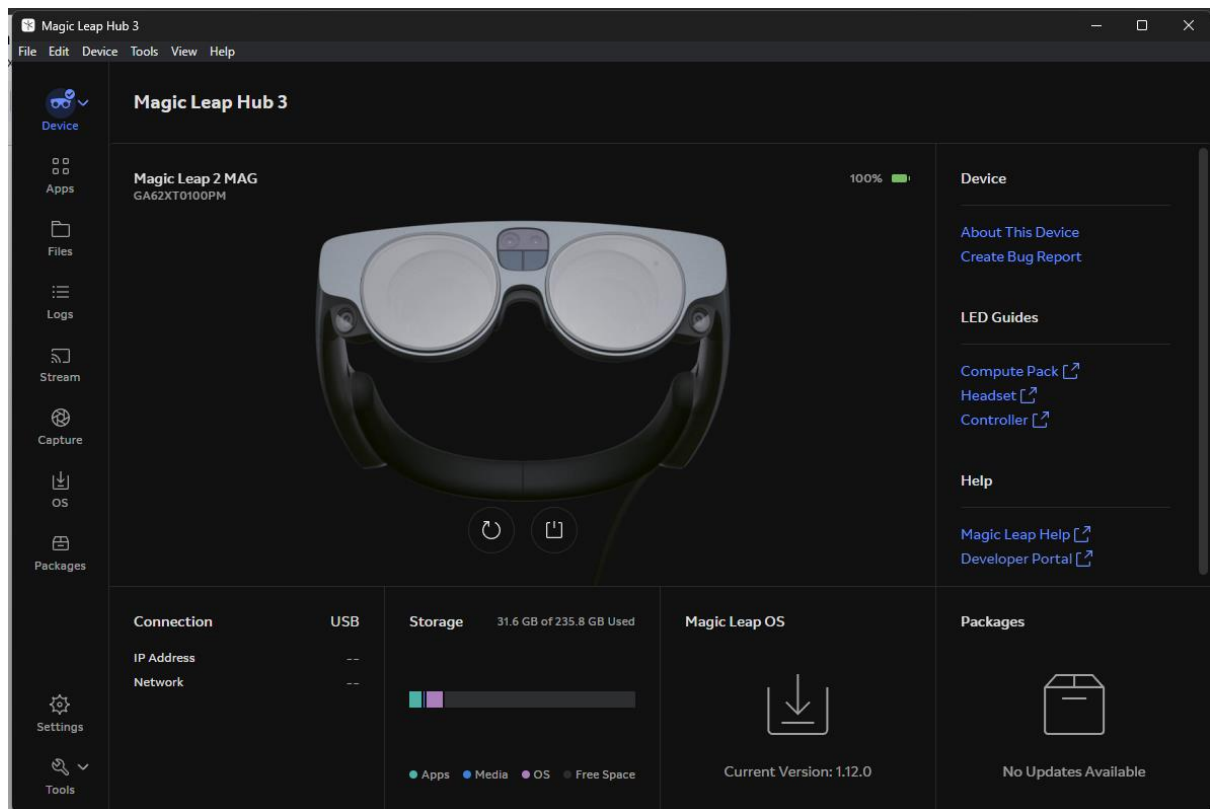


Figure 49. Development tools: Magic Leap Hub 3

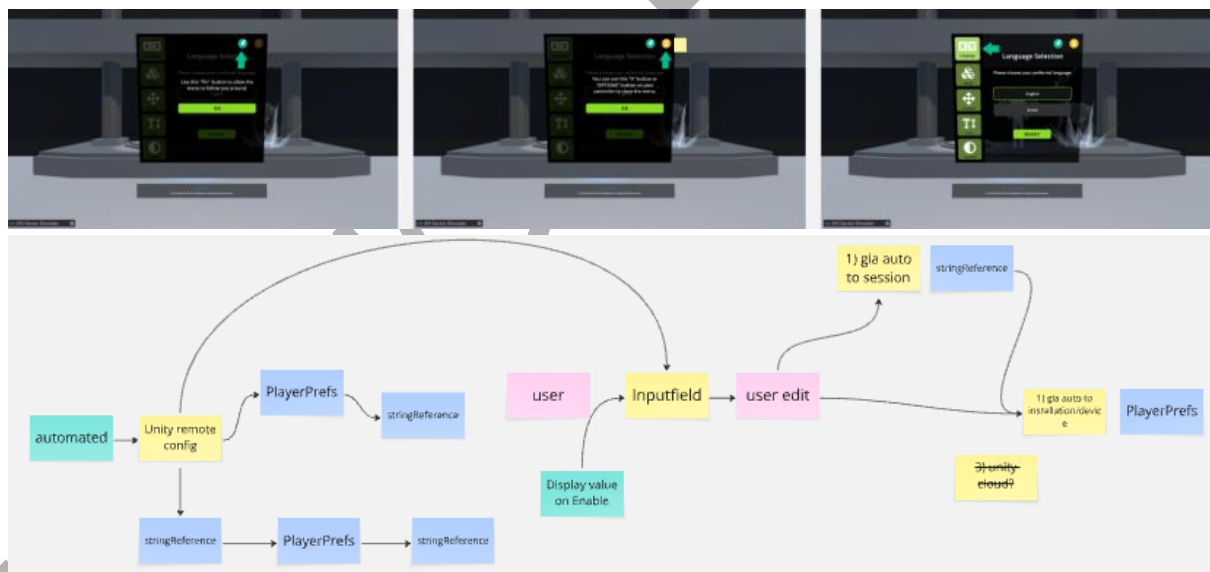


Figure 50. Development methods: diagrams and mockups for multidisciplinary collaboration

4.2.2.2 3D Models and Scene Creation

The visual assets for the AR Theatre experience were produced through a combination of digital techniques selected to balance artistic quality with technical feasibility on standalone AR hardware. A significant portion of the assets - including character motifs, environmental overlays, and symbolic visual effects - were modelled, textured, and animated using established 3D content creation pipelines, with a particular preference for free and open-source software such as Blender. This choice reflected both cost-efficiency and compatibility with the team's distributed production environment.

To achieve a distinctive visual identity though, a hand-painted aesthetic was also employed. Textures for 3D models were manually illustrated using digital painting techniques, providing a stylized, expressive finish. For animated 2D elements, a hybrid production methodology was followed: base layers were created through digital illustration, while animation was carried out either through traditional frame-by-frame workflows (optimized at 12 fps for performance reasons) or via digital keyframing. These animations were developed using professional-grade software such as Adobe After Effects and Adobe Photoshop. The use of proprietary tools in this context was necessitated by both the advanced feature set required and the creative team's existing expertise, given the limited availability of equivalent free/open-source alternatives supporting high-quality 2D animation pipelines. In addition, particle-based VFX were created using Unity's Visual Effect Graph tool.

From a technical standpoint, significant attention was given to asset optimization. All 3D models were evaluated for polygon and triangle count and were subject to mesh simplification procedures. Textures were consolidated into atlases and compressed appropriately to minimize memory load. Shader complexity and rendering parameters were tuned to ensure the application maintained consistent performance under standalone AR conditions, with particular emphasis on maintaining a stable frame rate and responsive rendering.

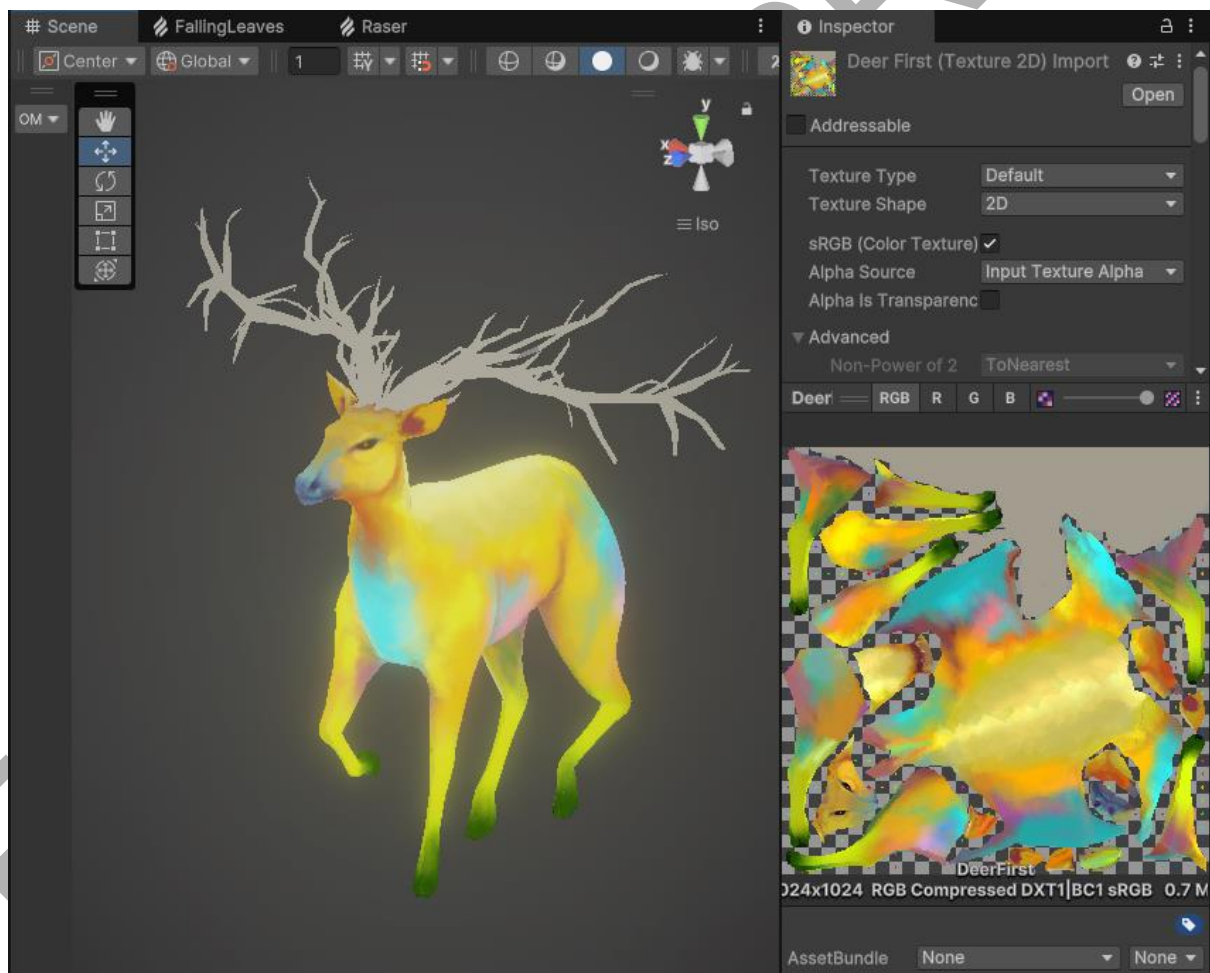


Figure 51. Custom created 3D model using Blender with hand-painted textures

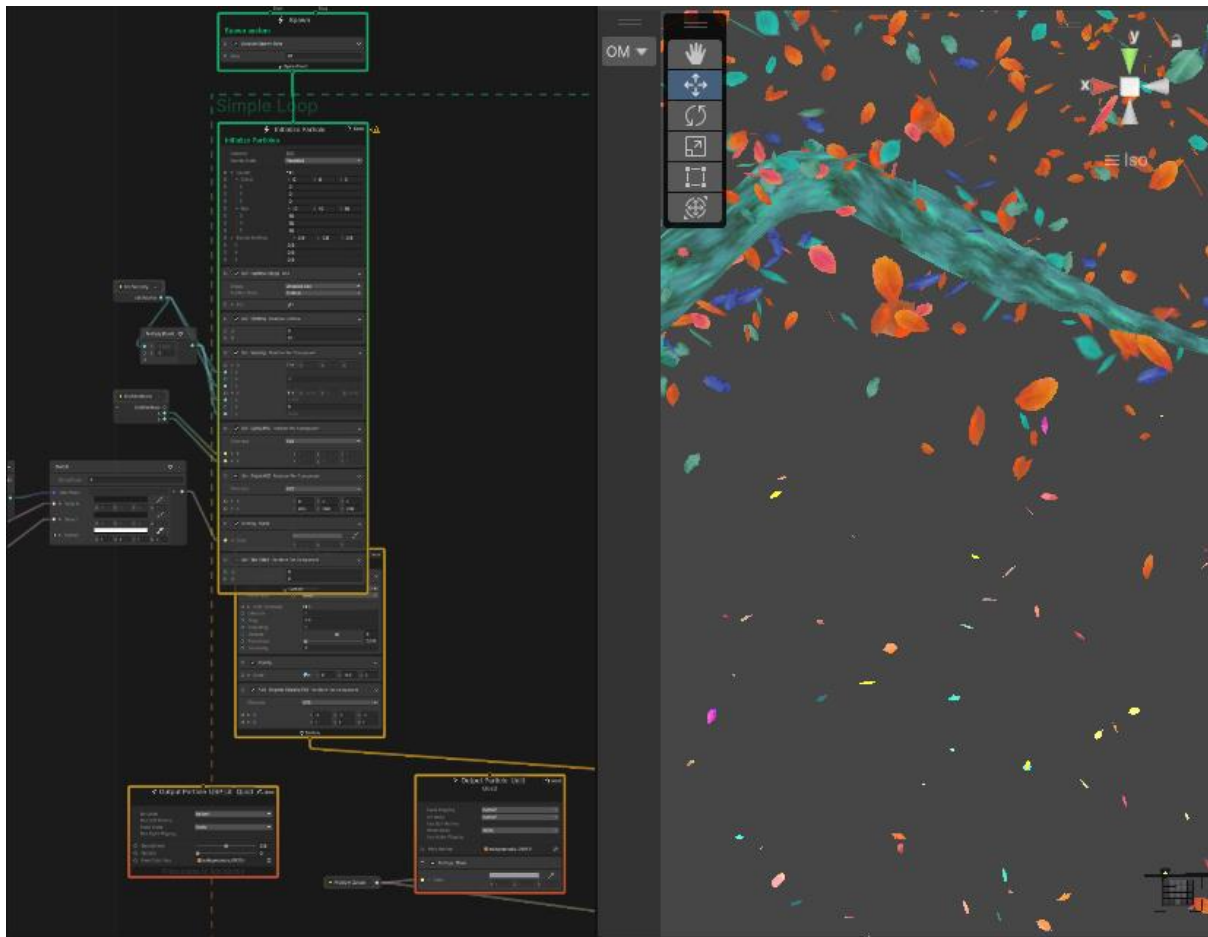


Figure 52. Procedural Visual Effects using Unity's Visual Graph

4.2.2.3 Core Algorithms and Techniques

The AR Theatre system has various algorithms to support the functionalities described in the above sections and visualized in detail from an end-user perspective in the User Interface Implementation section. In this section, the core VOXReality algorithms are chosen for a more detailed and technical presentation:

4.2.2.3.1 Transcription

As mentioned, the AR Theatre system targets the ASR streaming and matching approach. This service is provided as a pre-built docker container, available for downloading and running from the VOXReality Dockerhub profile (docker pull voxreality/draft_asr:audio_streaming_subtitle). Alternatively, it can be downloaded and composed from scratch from the “speech-translation-demo” repo hosted in VOXReality’s GitLab service, targeting branch “whisper_streaming_subtitles”. (https://gitlab.com/horizon-europe-voxreality/multilingual-translation/speech-translation-demo/-/tree/whisper_streaming_subtitles). In both cases, a Hugging Face token is required to run the service, which can be issued from the HuggingFace user account page.

The ASR streaming and matching service requires to be provided at initialization with a csv file (according to a provided template) through the endpoint “/upload_subtitles”, as shown in Figure 53. The csv template needs to have clearly labeled headers for the referenced language (e.g. “el”) is shown in a sample in Figure 54. Uploading the csv to the service can be achieved with the user interface of the control server (Figure 66).

POST /upload_subtitles Upload Subtitles

Parameters

No parameters

Request body **required**

multipart/form-data

file * **required**
string(\$binary)

Figure 53. ASR upload subtitle endpoint

A	B	C
el	speaker	index
Δεν είμαι άγνωστη.	Aphrodite	0
Είμαι η Αφροδίτη· η θεά.	Aphrodite	1
Και σας εξουσιάζω όλους.	Aphrodite	2
Τιμώ εκείνους που με σέβονται.	Aphrodite	3
Μα εκείνους που θεωρούν τον εαυτό τους ανώτερο...	Aphrodite	4
Τους τσακίζω.	Aphrodite	5
Ο γιος του Θησέα και της Αμαζόνας,	Aphrodite	6
ο Ιππόλυτος,	Aphrodite	7
είναι ο μόνος μες στη χώρα αυτή που με θεωρεί κατώτερη θεά, ανάξια.	Aphrodite	8
Δεν έχει χρόνο για έρωτες,	Aphrodite	9
αγάπες, σεξ, και γάμο!	Aphrodite	10
Την Άρτεμη την κυνηγό, που είναι παρθένα, αυτή μόνο τιμά.	Aphrodite	11
Όλο μαζί της με κυνηγόσκυλα γυρνά και λιγοστεύει τα αγρίμια μεσ' στα δά...	Aphrodite	12

Figure 54. ASR Upload subtitle csv sample

Following that, a WebSocket connection is initialized. The first message should set up the service configuration parameters, targeting among others the desired model, language used, recording sampling duration, as well as smart duration flexibility based on speech activity, illustrated in Figure 55. The user can configure these parameters using the user interface of the control server (Figure 67). Afterwards, a WaveIn event provided by the NAudio library for C# is used to stream audio data from the chosen microphone interface to the AI service.

```
function sendAudioConfig() {
    let processingArgs = {};

    processingArgs = {
        chunk_length_seconds: parseFloat(document.getElementById('chunk_length_seconds').value),
        accumulation_chunk_length_seconds: parseFloat(document.getElementById('accumulation_chunk_length_seconds').value),
        max_accumulation_seconds: parseFloat(document.getElementById('max_accumulation_seconds').value),
        chunk_offset_seconds: parseFloat(document.getElementById('chunk_offset_seconds').value)
    };

    const audioConfig = {
        type: 'config',
        data: {
            model_name: model_name,
            language: language,
            target_language: tgt_language,
            top_k: top_k,
            processing_args: processingArgs,
        }
    };

    console.log("Sending config:", JSON.stringify(audioConfig));
    websocket.send(JSON.stringify(audioConfig));
}
```

Figure 55. ASR streaming and matching parameter configuration

The service responds with the raw transcription, as well as an array of matched entries from the provided csv, alongside additional useful metadata, such as confidence rating and processing time. A sample response is provided below for illustration purposes:

"Message Received: {"text": "Ο γιος του Θησέκη της Αμαζόνας", "transcriptions": ["Ο γιος του Θησέα και της Αμαζόνας.", "Είμαι η Αφροδίτη· η θεά.", "η Αφροδίτη θα πληρώσει για το θυμό της.", "Αχ Αφροδίτη! Δεν είναι θεός αυτή.", "Ο Ιππόλυτος τόλμησε να βιάσει τη γυναίκα του Θησέα!"]}, "lines": [[6, 1, 96, 33, 61]], "translations": [[null, null, null, null, null]], "confidence_metric": [[41.40753979849631, 37.59546781842276, 29.08701829520962, 24.030168019455232, 23.228756905025154]], "processing_time": 0.7023236751556396, "processed_chunk_size": 3.06}"

The value of using the matching approach is evidenced by comparing the raw transcription (tagged "text") with the first matched result (tagged "transcriptions"): "Ο γιος του Θησέκη της Αφροδίτης" is matched to "Ο γιος του Θησέα και της Αμαζόνας", thus removing both spelling mistakes and grammatical errors.

Lastly, quality assurance steps are implemented after the generation of a response. The first step is based on the confidence rating and is of highest importance and reliability. Through the control server, the user can configure a threshold value for filtering of the responses based on their confidence rating (Figure 67). Failed responses do not get forwarded down the data flow, i.e. do not get distributed to the AR clients or are examined for associated VFX. This effectively suppresses erroneous responses which can occur due to background noise in the microphones, improper segmentations of the audio streams or verbal speech unrelated to the theatrical play that is captured accidentally.

The second step examines the logical sequence of the responses and is based on the fact that the script of the play is performed linearly. To illustrate with a simple example, if line 19 was delivered before, the system expects to receive line 20 in the next response. If it does not, an optional error correction routine can be triggered, which is exposed on the user interface of the control server (Figure 68). The error routine increases the line index by one increment forcibly if the ASR-received index does not meet the expectation and forwards this result to the rest of the application data flow. It should be noted that none of these methods can guarantee a successful result, since they can fall on edge cases that are hard to capture with deterministic code. Examples of correct and wrong cases are illustrated in the branching diagram in Figure 56 below.

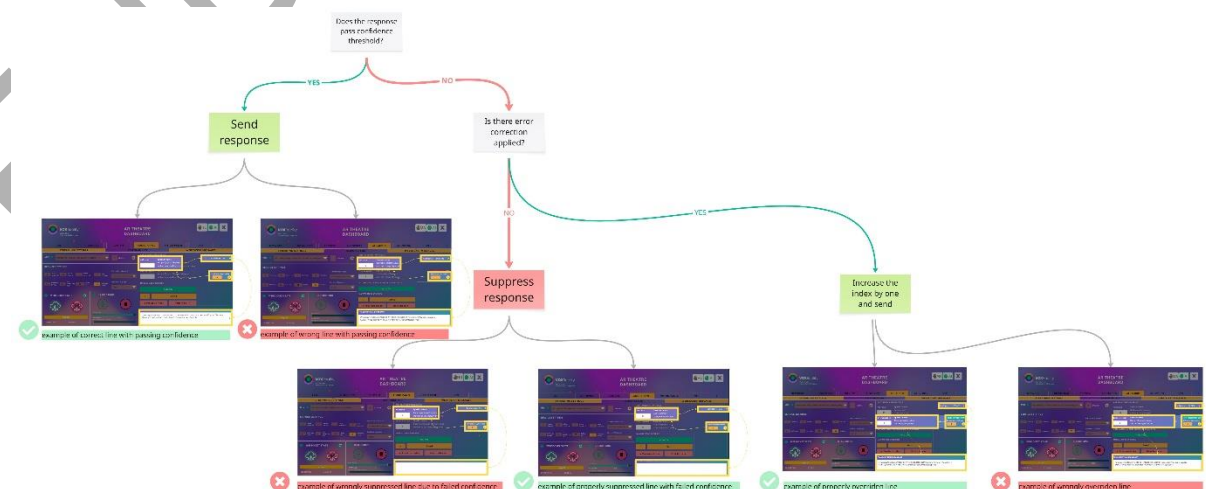


Figure 56. Quality assurance methods: confidence-based filtering and error correction

4.2.2.3.2 Vision language

The VL service used in the AR Theatre is the “VOXReality Image Visual Generic Question Answering service”. It is deployed as a Docker container and runs on device for minimal latency. The service supports providing an explicit question, which can be configured by the user through the user interface. For this use case, the question should be phrased in a “yes-no” format, such as “Is there a person in this picture?”. The user can redirect the camera’s field of view to target different areas of the stage and formulate questions with high levels of specificity, like “Is this space empty?”, “Is this door open?”, etc. The responses are received by a subroutine that detects changes in response content and raises relevant notifications. Specifically, based on the example relating to the detection of “people” in a photograph, a change from “no” to “yes” is interpreted as an “actor_entrance”, and vice-versa as an “actor_exit” stage event. These stage events are forwarded to the VFX Triggering logic which scans a csv file for associated keywords to distribute to the AR clients, just like with the ASR responses. Unlike the ASR responses though which relate to VFX keywords in a 1:1 relationship, an “actor_entrance” stage event can be triggered multiple times in a play and can be associated each time with a different VFX keyword based on the progress of the play. Since the model cannot distinguish actors without explicit training, any actor entrance will be described with the same stage event name and needs to be disambiguated to determine which actor entered - and therefore which VFX keyword to use each time. To do this, the VFX Trigger logic takes the previously delivered caption into account as a way to determining more context about the stage event. “Figure 60. Sample of VFX plan in csv format” showcases a practical example: the “actor_entrance” stage event in its first occurrence relates to lyrics index 0, refers to the entrance of Aphrodite according to the director’s instructions, and should trigger the “ActivateFireflies” and “PlayCue1” events, while the “actor_entrance” stage event in its second occurrence relates to lyrics index 6, refers to the entrance of Hippolytus, and should trigger the “DeactivateFireflies” event. This way the VL service can remain agnostic to the play’s details, but still provide detailed information about the stage events.

4.2.2.3.3 Translation

The NMT translation was performed using the contextual translation service, available in a dedicated container for optimal size, hosted in VOXReality’s DockerHub profile (docker pull voxreality/draft_asr:translation_context). The service requires setting the source and target language, the target text, as well as the contextual text. A screenshot of the respective endpoint, as demonstrated with FASTAPI and included in the service, can be seen in Figure 57. For the AR theatre use case, the context was chosen to be the previous caption line of the lyrical text, as formatted in a csv file by a human editor. The implemented code, illustrated in Figure 58, can generate automatically the translations for a configurable amount of csv entries and VOXReality languages, allowing for quick and easy changes to the body of text. A sample of the end-result is sampled in Figure 59. This result is available to the AR clients as a local or downloaded resource for retrieving the desired translation from file in runtime. It follows that there is no exposure of the translation service in the control server, since the translation step should be performed early on in the production process of the play, and not during the actual performance delivery.

POST

/contextual_translate_text

Contextual Translate Text

Parameters

Try it out

Name	Description
source_language * required (query)	Available values : en, el, it, nl, de, es <div>en</div>
target_language * required (query)	Available values : en, el, it, nl, de, es <div>en</div>

Request body required

application/json

Example Value

Schema

```
{
  "text": "string",
  "context": "string"
}
```

Figure 57. Contextual translation endpoint

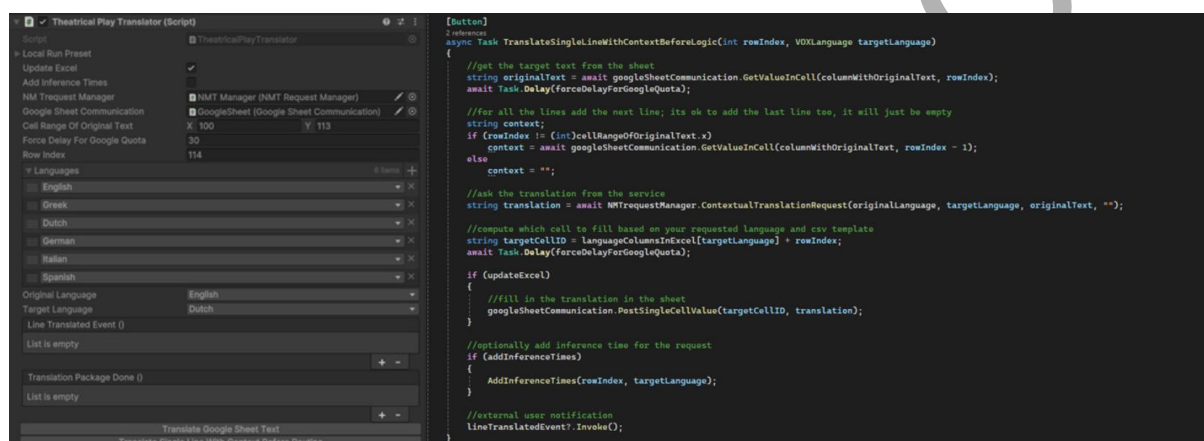


Figure 58. Contextual translation code block

	A	B	C	D	E	F	G	H
Lyrics	English	Spanish	Italian	German	Dutch			
1	Αφροδίτη	Αν είσαι άφροδίτη,	I'm not a stranger.	No soy un extraño.	Non sono uno straniero.	Ich bin kein Fremder.	Ik ben geen vreemdeling.	
2	Αφροδίτη	Είσαι η Αφροδίτη η θεά.	I am Venus, the goddess.	Soy Venus, la diosa.	Io sono Venere, la dea.	Ich bin Venus, die Göttin.	Ik ben Venus, de godin.	
3	Αφροδίτη	Και σ'ες έρωσάου άκου.	I am in charge of you all.	E yo me encargo de todos ustedes.	E io sono responsabile di tutti voi.	Und ich bin für euch alle zuständig.	Erk ben verantwoordelijk voor jullie allemaal.	
4	Αφροδίτη	Τυρώ εκεινους που με έφταναν.	I respect those who respect me.	Respeto a los que me respetan.	Rispetto chi mi rispetta.	Ich respektiere die, die mich respektieren.	Ik respecteer mensen die mij respecteren.	
5	Αφροδίτη	Μα εκεινους που θεωρουν τον εαυτο τους άνω των άλλων.	But those who consider themselves superior...	Pero aquellos que se consideran superiores...	Ma quelli che si considerano superiori...	Aber die, die sich selbst als überlegen halten...	Maar degene die zichzelf als superior beschouwen...	
6	Αφροδίτη	Τους ταξιδι.	I'm treating them apart.	Los voy a romper.	Li strappo a pezzi.	Ich zerreisse sie.	Ik scheur ze uit elkaar.	
7	Αφροδίτη	Ο γάμος του Θεού και της Αφροδίτης.	He's the Thesus and the Amazon.	El hijo de Tesao y el Amazonas,	Il figlio di Tesao e dell'Amazona,	Der Sohn von Theus und dem Amazonas,	De zoon van Theus en de Amazone,	
8	Αφροδίτη	Α ηρωίδα.	Α ηρωίδα.	Ηρωίδα.	Ηρωίδα.	Ηρωίδα.	Ηρωίδα.	
9	Αφροδίτη	Είμαι ο πρώτος που έχω αυτή την ιδέα.	She is the only in this country who considers me a lesser god	Es el único en este país que me considera un di	È l'unico in questo paese che mi considera un d	Er ist der Einzige in diesem Land, der mich für ei	Hijs is de enige in dit land die me als een minder	
10	Αφροδίτη	Αν έχω χρόνο για έρωτα.	Love has no time for love.	No tiene tiempo para el amor.	No han tempo per l'amore.	Er hat keine Zeit für Liebe.	Hijs heeft geen tijd voor liefde.	
11	Αφροδίτη	αγάπης, σέξ, και γάμου.	Love, sex, and marriage	(Amor, sesso y matrimonio)	(Amor, sesso e matrimonio)	Liebe, Sex und Ehe!	Liefde, seks en huwelijk!	
12	Αφροδίτη	Τη Άρτεμις την κυνηγό, που είναι η	Artemis the hunter, who is a virgin, she alone horns.	Artemis la cazadora, que es una virgen, ella sola	Artemis la cacciatrice, che è una vergine, è la sola	Die Jägerin, die eine Jungfrau ist, sie alle Ar	De jager die, die een maagd is, zij alleen een	
13	Αφροδίτη	Διο ψάει για τη κυνηγόλα που της έφα	She's always with her hunting dog, and she's running around an	Siempre está con su perro de caza, e corre por e	Siempre con il suo cane da caccia, e corre in g	Si sie immer mit ihrem Jagdhund und läuft durc	Ze altijd met haar jager, die een maagd is, en ze loopt	
14	Αφροδίτη	Αν έγω πορεύομαι άνω.	But he's gone for his death.	Pero ha ido de donde los lleva.	Ma er is al weg gegaan.	Aber er ist ja weg gegangen.	Maar hij is al weg gegaan.	
15	Αφροδίτη	Θνήσκει με το έρωτά απαραιτο (ζωή)	Mortal with a great god in a married couple.	Mortal con una diosa en una pareja casada.	Mortal con una dea in una coppia sposata.	Sterblich mit einer Göttin in einem verheiratete	Sterfelijk met een godin in een getrouwd stel.	
16	Αφροδίτη	Μεγάλη προσβολή, και να να εκδοί	Great insult, and here's my vengeance.	Groen insulto, y aquí está mi venganza.	Grande insulto, e questa è la mia vendetta.	Große Beleidigung, und hier ist meine Rache!	Groete belediging, en hier is mijn wraak.	
17	Αφροδίτη	Τη Φαίδρα, τη μητέρα του.	Faidra, his stepdaughter.	Faidra, su hijastra.	Faidra, sua figliastra.	Faidra, seine Stieftochter.	Faidra, zijn stiefmoeder.	
18	Αφροδίτη	την τρελάνα από έρωτα που κελουν	I'm crazy about him.	Estoy loco por él.	Sono pazzo di lui.	Ich bin verrückt nach ihm.	Ik ben gek op hem.	
19	Χορω	Τι έχει πάθει.	What's the matter?	¿Que pasa?	Que succede?	Was ist los?	Wat is er?	
20	Χορω	Αν λέει Τη πύρρα.	She doesn't say.	No dice.	Non dice niente.	Sie sagt nichts.	Ze zegt niets.	
21	Χορω	Ανέχεις, λέει, παραδίσεις.	You said he'd kicked, he'd kicked.	Dijste dat había leido, que había leído.	Hei detto che aveva leccato, ha leccato.	Da sagtest, er hätte geküsst.	Je zei dat hij had gekust, hij had gekust.	
22	Χορω	Τους μέρες φέρεται στο ανδρες.	They told me of mouth-crushing.	Telos días de apretamiento.	Tre giorni di abbracciamento.	Die Tage Mundschmelze.	Die dagen mond bezemen.	
23	Χορω	Την τρελά ο θείος τους.	You had been case of her?	¿Dijste que ocupó de ella?	Hei si è occupato di lei?	Uit die kende u haar bezem?	Het Gdijf voor haar bezem?	

Figure 59. Sample of translated lyrics across the VOXReality languages

4.2.2.3.4 VFX triggering logic

The VFX triggering logic is simple, but critical to synchronizing the play's content and realizing the artistic vision of the director, therefore is described shortly below. The VFX plan is structured as a single csv file which contains both verbally and visually (silent) triggered VFX keywords, as illustrated in [Figure 60](#). The lines that are not associated with a triggering event are included in this csv only for human understanding of the flow of the play (e.g. to assess the chronological order and density of triggered events). In addition, the same triggering event can be associated with multiple VFX keywords, with each keyword as a separate csv entry. The csv is parsed at the initialization of the control server by the VFX trigger manager, who



subscribes to ASR responses and VL-based stage event notifications. The manager, upon successfully detecting an associated VFX trigger keyword, distributes this as a WebSocket message to all connected clients. Clients have custom logic implemented for each of these keywords, according to detailed instructions by the director. For example, the AR clients trigger the rendering of specific predetermined visual effects, while the audio player client triggers the playback of specific predetermined audio effects. Figure 61 provides an example of the triggering method in the AR client and the VFX result for the VFX keyword “ActivateFireflies”, which is the first entry in the sample VFX plan provided below.

Lyrics index	VFXKeyword	Detection Method	Trigger condition (caption or stage event)
0	ActivateFireflies	visual	actor_entrance
0	PlayCue1	visual	actor_entrance
0			Δεν είμαι άγνωστη.
1			Είμαι η Αφροδίτη· η θεά.
2			Και σας εξουσιάζω όλους.
3			Τιμώ εκείνους που με σέβονται.
4			Μα εκείνους που θεωρούν τον εαυτό τους ανώτερο...
4			Μα εκείνους που θεωρούν τον εαυτό τους ανώτερο...
5	PlayCue2a	verbal	Τους τσακίζω.
5	ActivateGreenTreeAndFallingLeaves	verbal	Τους τσακίζω.
6	DeactivateFireflies	visual	actor_entrance
7			Ο γιος του Θησέα και της Αμαζόννας,
8	PlayCue2b	verbal	ο μοναδικός σε αυτή τη χώρα που με θεωρεί κατώτερη θεά, ανάξια.
9			Δεν έχει χρόνο για έρωτες,
10	ActivateFirstDeer	verbal	αγάπες, σεξ, και γάμο!
10	PlayCue2c	verbal	αγάπες, σεξ, και γάμο!
11			Την Άρτεμη την κυνηγό, που είναι παρθένα, αυτή μόνο τιμά.
12			Όλο μαζί της με κυνηγόσκυλα γυρνά και λιγοστεύει τα αγρίμια μεσ' στα δάση.
13	DeactivateFirstDeer	verbal	Το έχει παρακάνει όμως.
14	PlayCue5	verbal	Θνητός με μια θεά αταίριαστο ζευγάρι.
15			Μεγάλη προσβολή. Και να η εκδίκησή μου...
15	PlayCue6	verbal	Μεγάλη προσβολή. Και να η εκδίκησή μου...
16	DeactivateGreenTree	verbal	Τη Φαίδρα, τη μητριά του,
16	DeactivateFallingLeaves	verbal	Τη Φαίδρα, τη μητριά του,

Figure 60. Sample of VFX plan in csv format

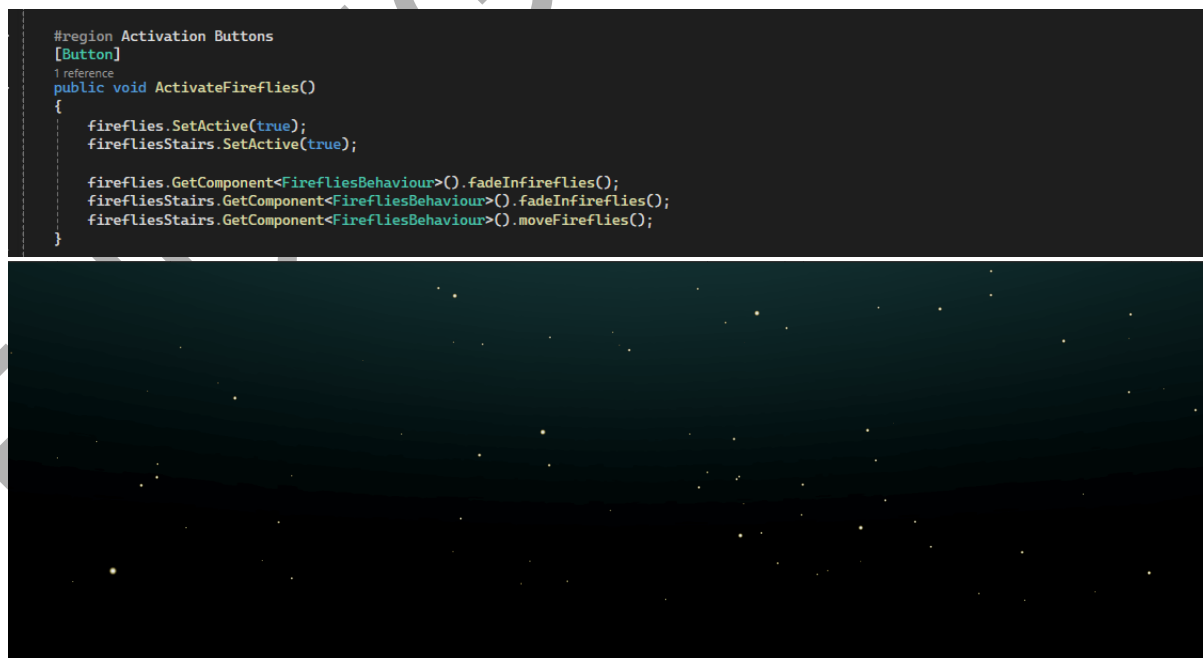


Figure 61. Example of a VFX trigger method and its result

4.2.2.4 User Interface Implementation

The user interface implementation section is structured in two subsections: subsection 4.2.2.4.1 is dedicated to the user interface of the AR Theatre server, which will be operated by the theatre's technical crew, and subsection 4.2.2.4.2 is dedicated to the AR Theatre client, which will be distributed to the audience.

4.2.2.4.1 AR Theater control server

The AR theater server has seven panels, positioned roughly in sequence of execution: "Network", "Download", "Preview", "AI-Services", "AI-Audio", "AI-Vision" and "VFX". Panels "Network" up to "AI-Services" are used for system setup and preparation before a performance or rehearsal starts. Panels "AI-audio" up to "VFX" are used during the performance. Each panel is described in detail below.

The tab "Network" (Figure 62) manages the WebSocket connection with the clients. Users can edit the IP and port for the connection, manage the connection state (start/stop) and preview the list of currently connected clients alongside relevant info (like client device battery levels). This feature is critical to detect any client disconnects in the audience. A future extension would be to report if a client disconnected gracefully (e.g. due to user command) or due to a application crash, system crash or network failure. In addition, the user can issue remote control commands to the clients. Remote control commands can be issued to all connected clients or only to selected ones using the "Connected Clients" panel. Remote control commands include setting up the user language for the AR client application and switching scenes (e.g. tutorial scene or main scene), as well as additional commands for remote supporting users. This is useful to minimize audience member interaction in case of inexperienced visitors (e.g. for localization), to quickly recover from any user handling errors, as well as to synchronize the state of the application across all members of the audience (e.g. for scene management).

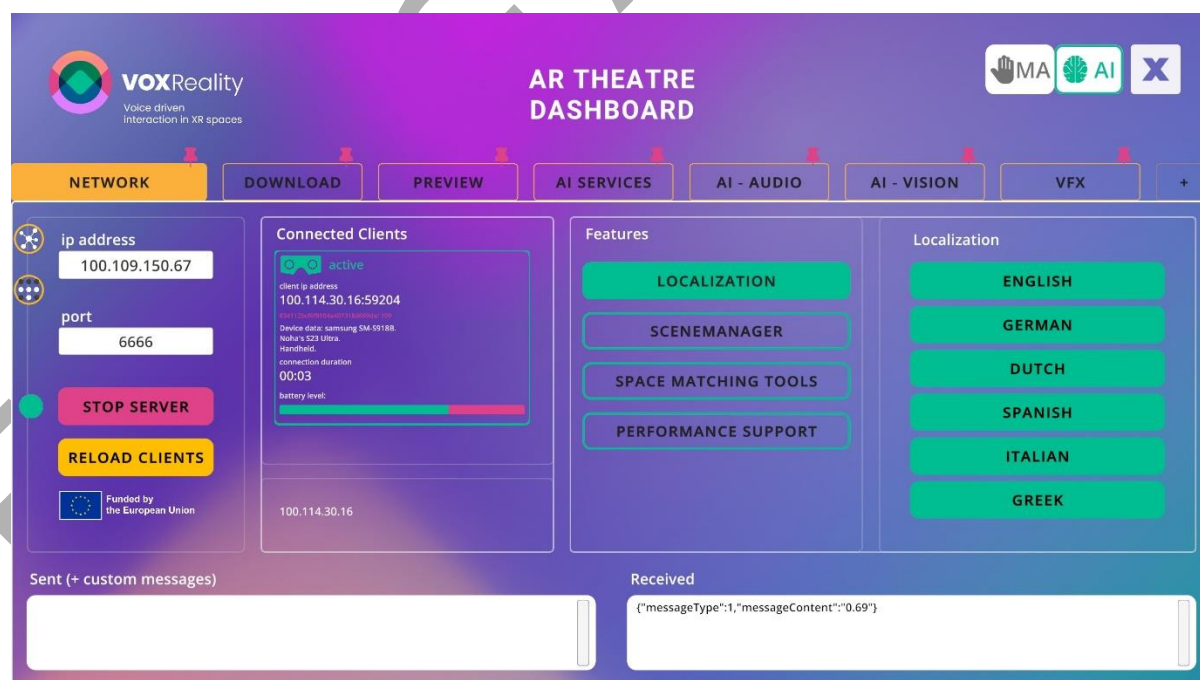


Figure 62. AR Theatre Server - Network tab

The tab "Download" (Figure 63) allows users to download the content of the current theatrical play from a remote location. This includes three assets: 1) the original transcript, 2) 0the caption translations, and 3) the VFX trigger plan. All three assets need to be in csv format and based on a given template. A sample is shown in Figure 64. The "download" feature allows

the AR Theatrical server to be content agnostic, avoids local file management for the convenience of the user, and supports reflecting on-the-fly edits to the content. Supporting on-the-fly edits has been proven especially useful during rehearsals, where quick iteration and practical experimentation are key to progressing the artistic vision. Downloaded files are stored on the persistent data path of the application on the device (e.g. for Windows “C:\Users\<username>\AppData\LocalLow\GruppoMaggioli\VoxReality AR Theatre Server”).

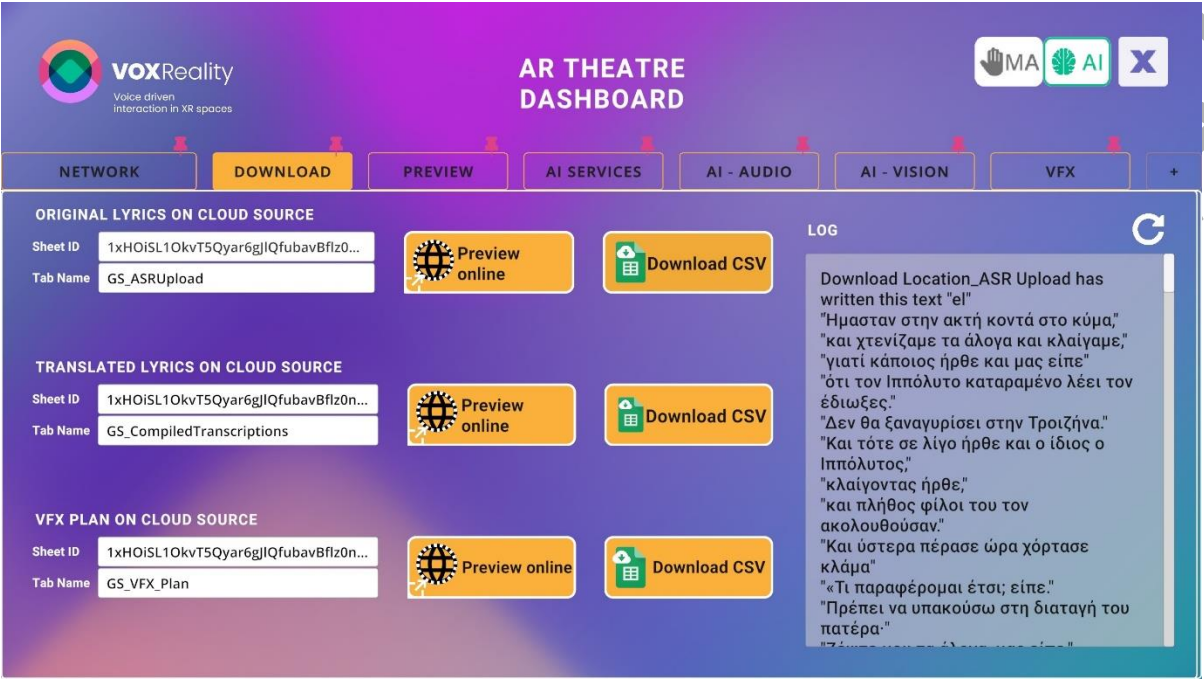


Figure 63. AR Theatre Server - Download tab

A	B	C	D	E	F	G	H
Lyrics index - e	SpeakerEnum	Greek	English	Spanish	Italian	German	Dutch
114	Chorus	τους τυλίγει με τα πολλά σου θέλητ'ρα.	wraps them in your many charms.	los envuelve con tus muchos encantos.	li avvolge nei tuoi mille incanti.	umhüllt sie mit deinen zahllosen Reizen.	En de wilde beesten van de b...
115	Chorus	Πετά στη γη	He flies to earth	Vuela sobre la tierra	Vola sulla terra	Er fliegt zur Erde	Hij vliegt naar de aarde
116	Chorus	και στην ασίγηστη αρμυρή θάλασσα	and in the unquenchable salty sea	y en el inextinguible mar salado	e nell'insonnabile mare salato	und ins unauslöschliche salzige Meer	En in de onblusbare zilt zee
117	Chorus	και την καρδιά, όπου θα μπει,	and the heart, wherever it enters,	y en el corazón, dondequiera que e	e nel cuore, ovunque entri,	und ins Herz, wo immer er eintritt,	en in het hart, waar hij ook bin
118	Chorus	την τρελαίνει ο χρυσόφειγος	is driven mad by the gold light	lo enloquece con la luz dorada	lo fa impazzire con la luce dora	er es wahnsinnig mit golden macht	hij het krankzinnig door
119	Chorus	και τα μαγεύει όλα	and enchants them all.	y los encanta a todos.	e li incanta tutti.	und verzaubert sie alle.	en betovt hij iedereen.
120	Chorus	και των βουνών τ' αγρία και της θάλασσας	And the wild beasts of the mountains and of	Y las bestias salvajes de las monta	E le bestie selvagge dei monti	Und die wilden Tiere der Berge und	En de wilde beesten van de b...
121	Chorus	και όλα τρέφει η γη	and all that the earth feeds	y todo lo que la tierra alimenta	e tutto ciò che la terra nutre	und alles, was die Erde nährt,	en alles wat de aarde voedt.
122	Chorus	και όσα ο αναμμένος Ήλιος βλέπει	and all that the burning sun sees	y todo lo que el ardiente sol ve	e tutto ciò che il sole ardente vi	und alles, was die brennende Sonn	en alles wat de brandende zor
123	Chorus	και τους ανθρώπους	and the mortals.	y los mortales.	e i mortali.	und die Sterblichen.	en de stervelingen.
124	Chorus	Τη μεγαλύτερη τιμή	The greatest honor	El mayor honor	Il più grande onore	Die größte Ehre	De grootste eer
125	Chorus	σε σένα ακουμπώ Αφροδίτη	I lay upon you, Aphrodite.	lo deposito en ti, Afrodite.	lo pongo su di te, Afrodite.	lege ich in deine Hände, Aphrodite	leg ik aan jouw voeten, Aphro
126	Chorus	Μόνη σου σε όλα εξουσιάζεις, Αφροδίτη.	You rule in all things alone, Aphrodite.	Tú reinas sobre todas las cosas en	Tu domini ogni cosa da sola, A	Du herrschst über alles allein, Aph	Jij regeert over alles alleen, A
127	Artemis	- Είμαι η Άρτεμη,	- I am Artemis,	- Yo soy Artemisa,	- Io sono Artemide,	- Ich bin Artemis,	- Ik ben Artemis,
128	Artemis	Θησεία!	Theseus!	¡Theseo!	Theseo!	Theseus!	Theseus!
129	Artemis	Η κόρη της Λητώ:	The daughter of Leto,	La hija de Leto, la diosa Artemisa.	La figlia di Leto, la dea Artemid	Die Tochter der Leto, die Göttin Ar	De dochter van Leto, godin Ar
130	Artemis	η Άρτεμη η θεά.	goddess Artemis.	Diosa Artemisa.	Dea Artemide.	Göttin Artemis.	Godin Artemis.
131	Artemis	Σου μιλώ και σε προστάζω:	I speak to you and command you;	Te hablo y te ordeno; escucha.	Ti parlo e ti comando; ascolta.	Ich spreche zu dir und befehle dir;	Ik spreek tot je en beveel je; ik
132	Artemis	άκουσε.	listen.	Escucha.	Ascolta.	Höre zu.	Luister.
133	Artemis	Θησεία, γιατί χαίρεσαι δύστυχ;	Theseus, why are you glad, poor fellow?	Teseo, ¿por qué estás feliz, pobre?	Teseo, perché sei felice, povero!	Theseus, warum freust du dich, ari	Theseus, waarom ben je blij, i
134	Artemis	Το γιο σου σκότωσες	You have killed your son.	Has matado a tu hijo.	Hai ucciso tuo figlio.	Du hast deinen Sohn getötet.	Je hebt je zoon gedood.
135	Artemis	Πίστεψες τα ψέματά της γυναίκας σου,	You believed your wife's lies,	Creíste las mentiras de tu esposa,	Hai creduto alle bugie di tua m	Du hast den Lügen deiner Frau ge	Je geloofde de leugens van je
136	Artemis	δεν άκουσες κανέναν	you wouldn't listen to anyone,	no quisiste escuchar a nadie, lo ser	non hai voluto ascoltare nessun d	wolltest niemanden anhören, di; je	wilde naar niemand luistere
137	Artemis	τον καταδικάσες σε θάνατο.	you sentenced him to death.	Lo sentenciaste a muerte.	Lo hai condannato a morte.	Du hast ihn zum Tode verurteilt.	Je veroordeelde hem ter dood
138	Artemis	Τρομερό το έγκλημά σου.	Your crime is terrible.	Tu crimen es terrible. Eres culpable	Il tuo crimine è terribile. Sei col	Dein Verbrechen ist schrecklich. D	Je misdaad is verschrikkelijk.
139	Artemis	Είσαι ένοχος.	You are guilty.	Eres culpable.	Sei colpevole.	Du bist schuldig.	Je bent schuldig.
140	Artemis	Τρέξε στα Τάρταρα και χύσου, κρύψου!	Run to Tartarus and hide!	¡Corre al Tártaro y escóndete!	Corri nel Tartaro e nasconditi!	Fieh in den Tartaros und verstecke	Vlucht naar Tartarus en verber
141	Artemis	Έγινε πουλί,	Be a bird, fly high to escape the plague.	Se un pájaro, vuela alto para escap	Sii un uccello, vola in alto per s	Werde ein Vogel, flieg hoch, um d	Word een vogel, vlieg hoog or
142	Artemis	πίτα ψιλά να ξεφύγεις το μίσος.	fly high to escape the plague.	Vuela alto para escapar de la peste	Vola in alto per sfuggire alla pe	Flieh hoch, um der Plage zu entk	Vlieg hoog om de plaag te ont

Figure 64. AR Theatre Server - Content sample

The tab “Preview” (Figure 65) supports previewing the downloaded content. This helps validate that the download process was successful and that the content was parsed correctly. It also allows examining the content in a more engaging format than the csv with the aim to detect potential improvements in lyrical content or formatting. Finally, this feature also allows fast cross-checking of the content between the AR client and the server-side reference.



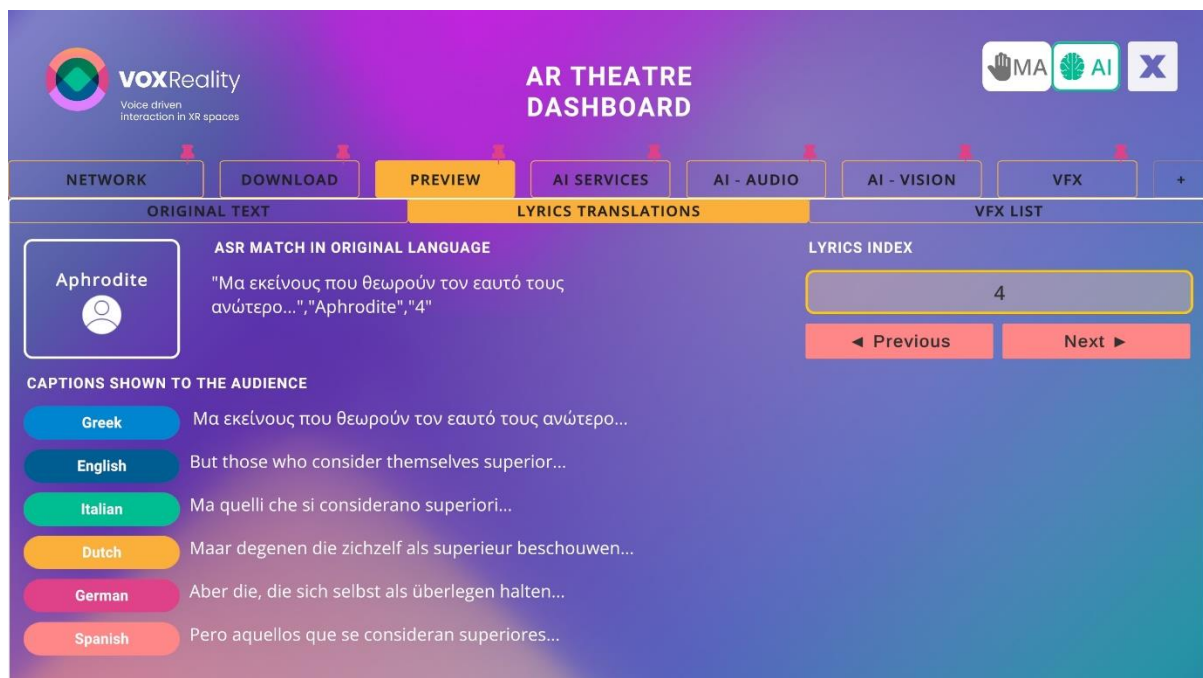


Figure 65. AR Theatre Server - Preview tab

The tab “AI services” (Figure 66) allows users to configure the location of the VOXReality AI services based on their current deployment option. Users can choose between cloud or edge deployment, and edit the respective address independently for the ASR and VL services. A text panel debugs the http request responses from the services validating their communication. In this step, the original lyrics (which were downloaded in the previous step), need to be uploaded to the ASR service to enable the matching process. This is performed automatically at initialization of the application but is also available as a user interface feature in case of live edits to the csv source file.

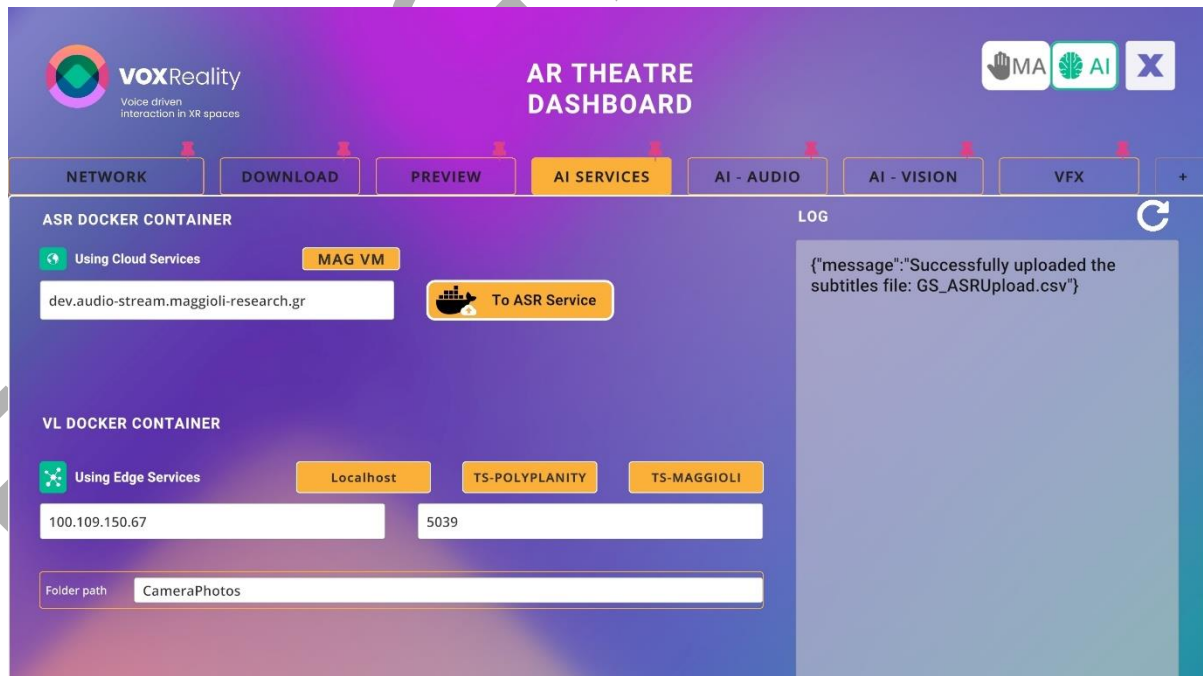


Figure 66. AR Theatre Server - AI Services tab

The “AI - Audio” tab is one of the most important interfaces of the application – it manages the connection to the ASR (audio streaming and matching) service. This tab has three subpanels: “Streaming settings”, “Response log” and “WebSocket messages”. The “Streaming settings”

panel (Figure 67) is used to set up all relevant recording and ASR service configuration parameters, such as recording duration, silence duration, allowed accumulation, etc. It also manages (start/stop) the connection to the ASR service over a WebSocket connection and the microphone streaming (start/stop). In addition, it allows users to export a recording of the audio stream for validation purposes. The "Response log" tab (Figure 68) debugs the unformatted responses and is only used for communication validation purposes. The "WebSocket messages" tab (Figure 68) shows the parsed responses as properly formatted WebSocket messages to the connected clients. As a side-note, the WebSocket connection between the ASR service and the AR Theatre server should not be confounded with the WebSocket connection between the AR Theatre server and the AR clients in the audience. ASR responses are also parsed by a separate logic which detects if the performed lyric is associated with a VFX trigger, as determined by the downloaded VFX plan. If so, the associated VFX trigger command is also streamed as a WebSocket message to the AR clients, as will be presented in the VFX tab.

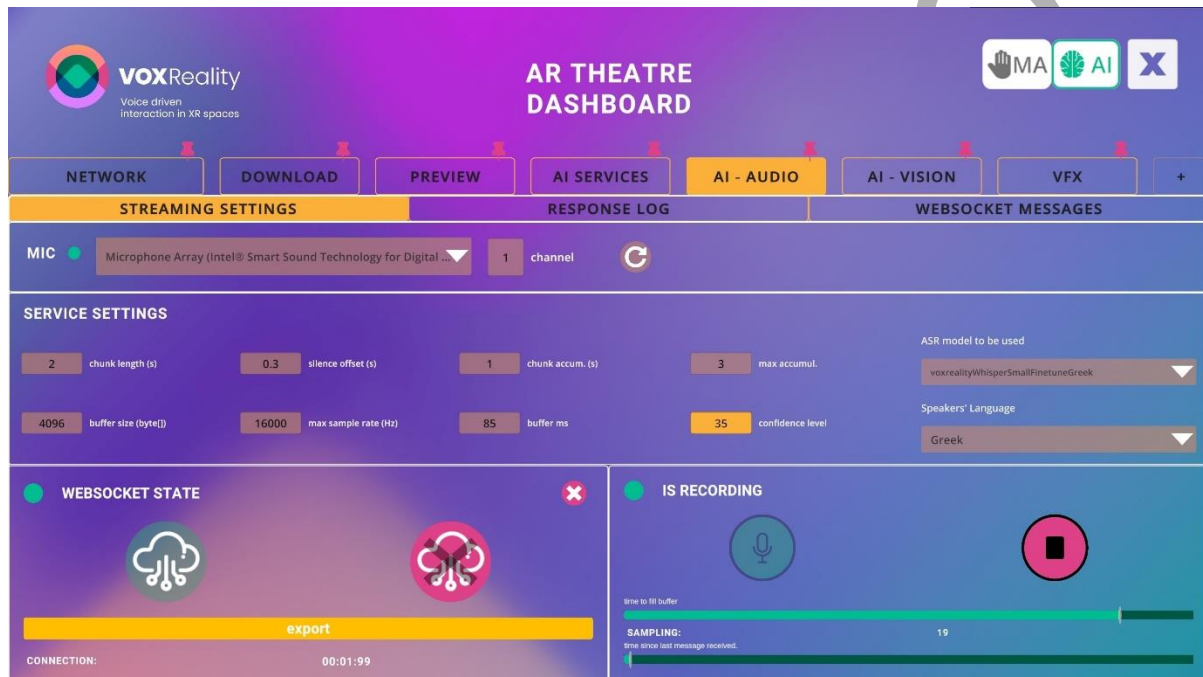


Figure 67. AR Theatre Server - AI-Audio tab: Streaming settings

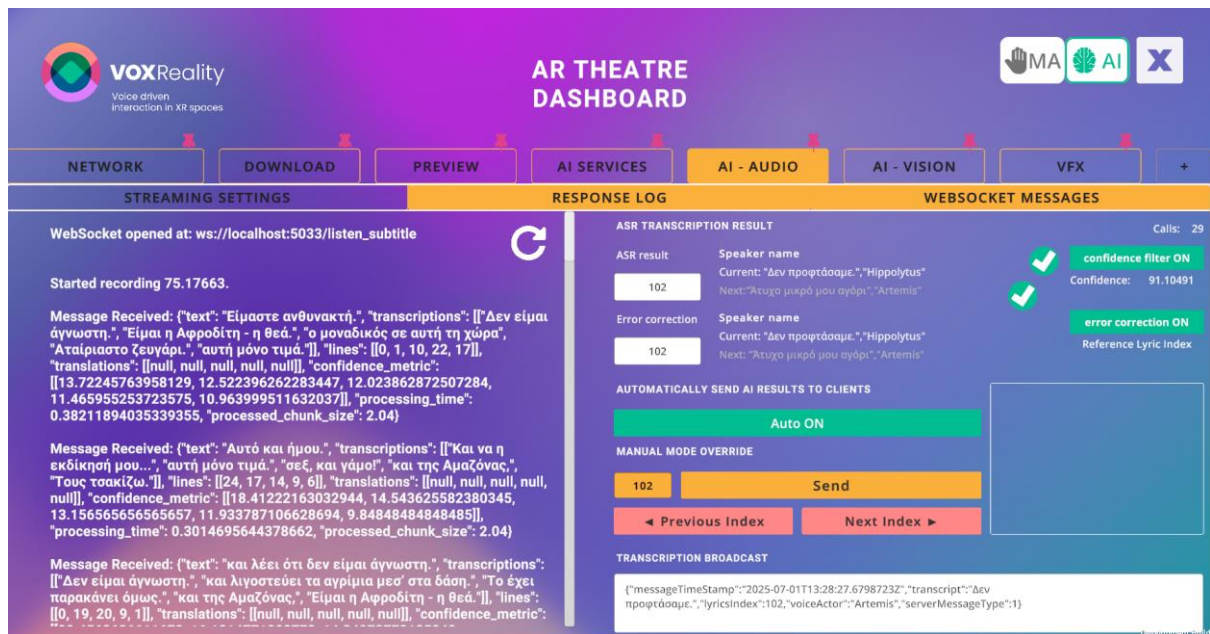


Figure 68. AR Theatre Server - AI-Audio tab: ASR response log & WebSocket messages

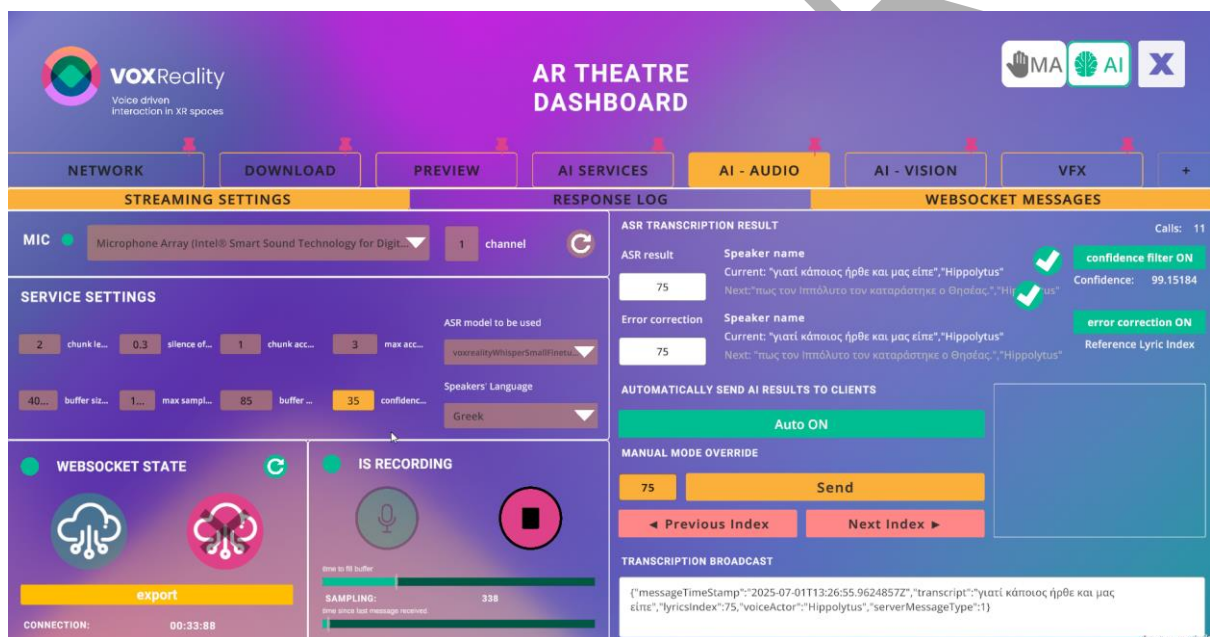


Figure 69. AR Theatre Server - AI-Audio tab: summary of configurable parameters

The “AI - Vision” tab manages the connection to the VL service. It supports capturing images from any of the connected WebCamera interface of the device and automatically forwards it to the configured VL service. After testing multiple services, the VOXReality Image Visual Generic Question Answering service was preferred as the most flexible yet concrete approach, as described in Section 4.2.2.3.2. The photocapturing can be automated with a configurable frequency exposed on the user interface. Alternatively, the application can instead load the latest image from a configurable folder on disk. This is useful in case a more sophisticated third-party photo capturing software (such as DSRL camera software) is used to capture and store images on disk instead. The responses can be previewed on the debug text panel. Responses are parsed by a separate post-processing logic, which checks for changes in the VL responses to detect events, like the presence or absence of the requested element (like people or items on stage). For the chosen play, it was decided that the logic should detect the entrance and the exit of actors on stage. The “actor_entrance” event and the “actor_exit” events are then used to trigger VFX, as presented in the next and final tab.

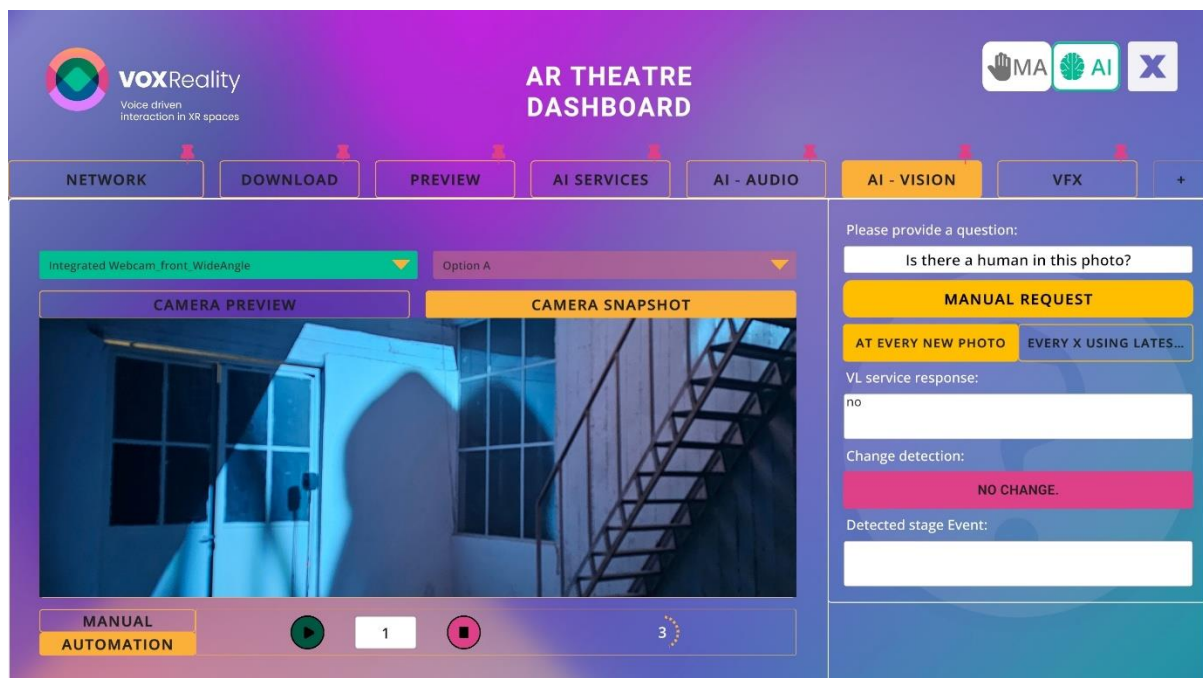


Figure 70. AR Theatre Server - AI-Vision tab – VQA with no stage event detected

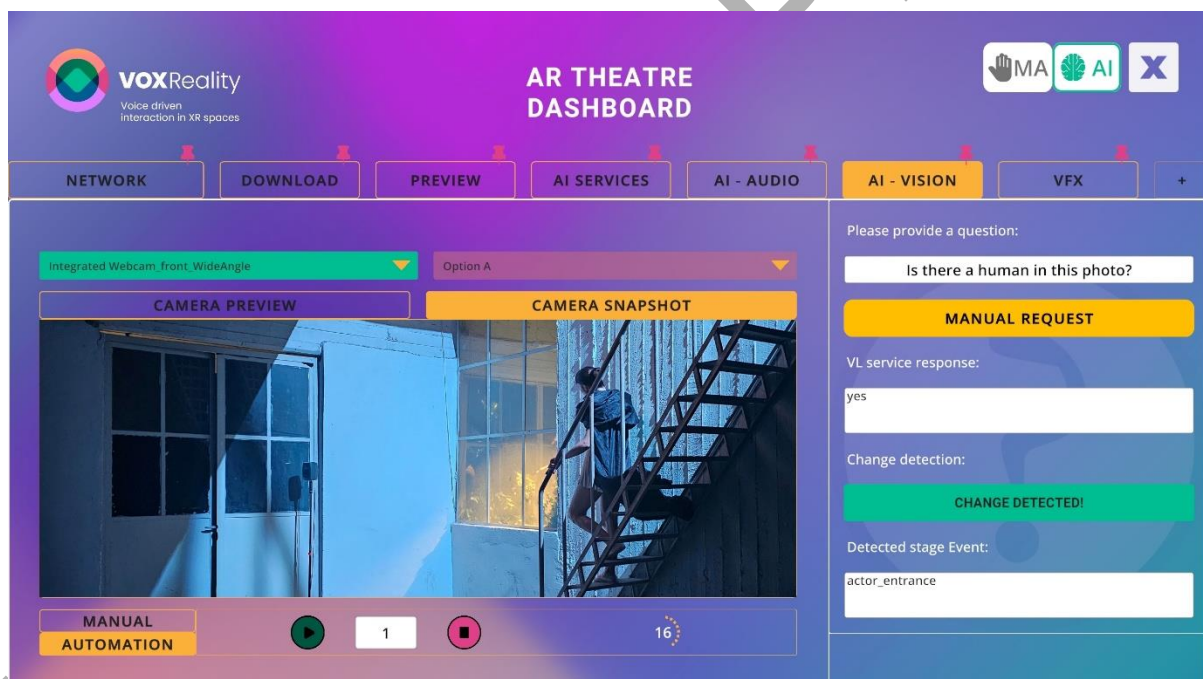


Figure 71. AR Theatre Server - AI-Vision tab – VQA with stage event detected

The “VFX” tab parses the downloaded VFX plan (as downloaded using the “Download” tab and previewed using the “Preview” tab) and generates a list of buttons for each detected VFX trigger word. This allows a user to manually trigger the VFX on the connected clients, which is especially useful for rehearsals. The tab also allows previewing which VFX have already been triggered at least once in this session with a checkmark.

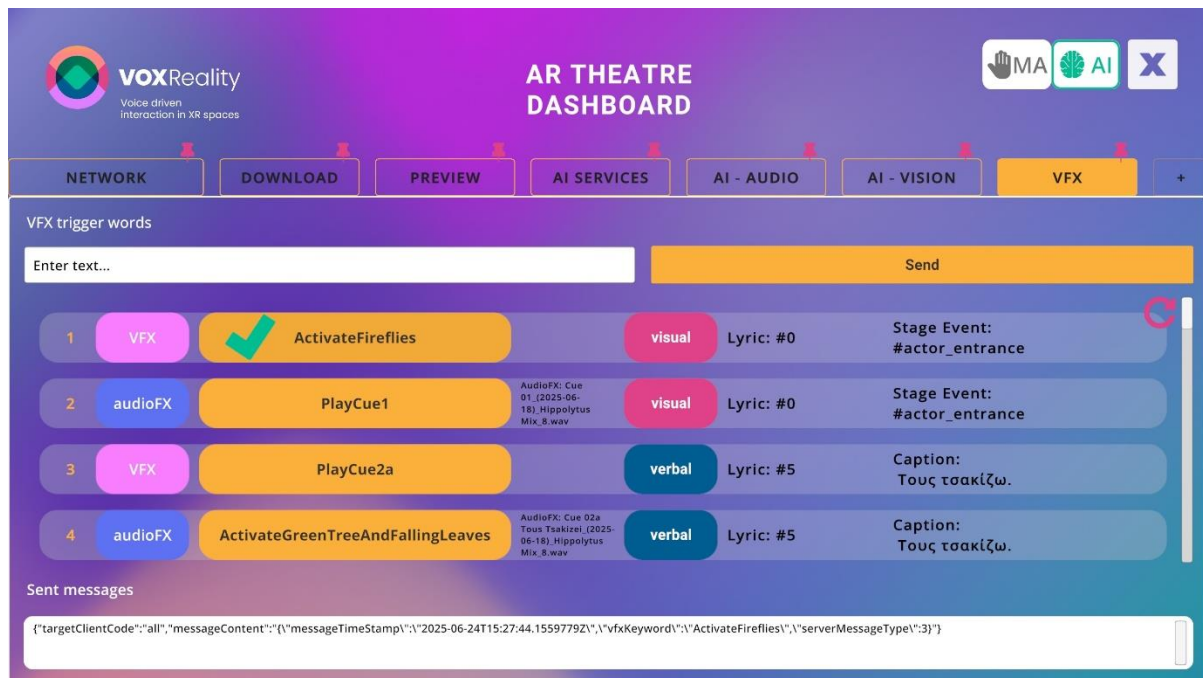


Figure 72. AR Theatre Server - VFX tab

Lastly, the AR Theatre server has been enriched with two user interface modes to support the specific needs of Pilot 2. As a reminder, Pilot 2 will compare between two conditions: the baseline condition, which reflects currently mainstream available technology, and the test condition which uses the VOXReality services. In the base condition, the operator will need to use the AR Theatre server interface to manually send WebSocket messages (captions and VFX trigger keywords) to the audience. To be able to time the messages, the operator will need to monitor the stage using their senses, i.e. listen to the actors' speech and watch the on-stage events. In addition, the operator will need to have basic knowledge of the play to know exactly when to send the appropriate messages. In the VOXReality condition, the AR Theatre server should send the appropriate messages automatically using the AI services and the feed from the microphones and cameras. The operator will need to monitor the AR Theatre server application's runtime instead of the stage events, and will need to know how to setup the application, instead of knowing about each specific play. The difference between the conditions is illustrated in Figure 73.

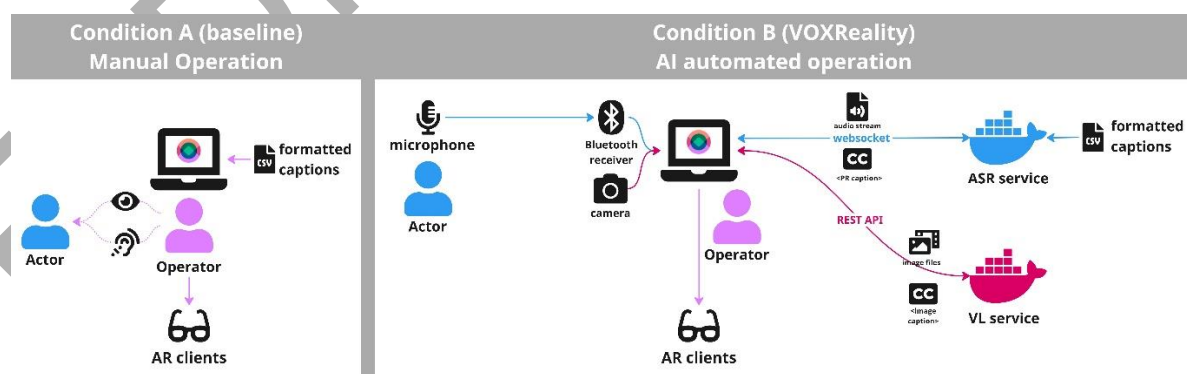


Figure 73. AR Theatre - Systems for Pilot 2 conditions

Given the above requirements and in order to facilitate and safeguard the piloting, the AR Theatre server has two modes restricting access only to the pertinent features based on the current condition. Switching between modes can be enabled from the top right corner with a toggle. The mode for the baseline condition is termed "MA" for "manual" and is marked with a hand icon to denote manual operation, while the mode for the VOXReality test condition is

termed “AI” and is marked with a brain icon to denote the processing of the audiovisual content by the AI services. The “MA” interface does not have access to the “AI services”, “AI-audio” and “AI-vision” tabs, as shown in Figure 74. Instead of the “AI-Audio caption”, the manual mode has access to a new tab called “Captions”, which can be used to manually send lyrics captions to the audience. To be practically useful, the operator is advised to also pin a preview of the play in their interface, as shown in Figure 75.

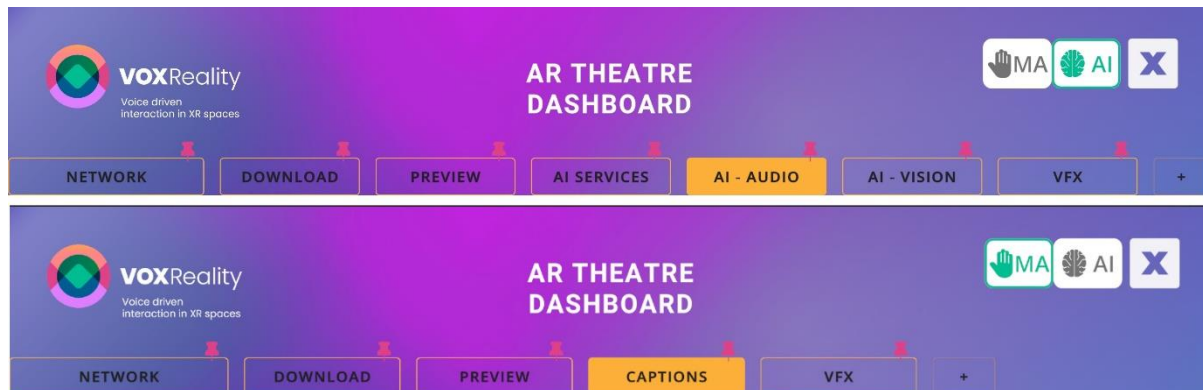


Figure 74. AR Theatre Server - Interfaces for manual and AI mode

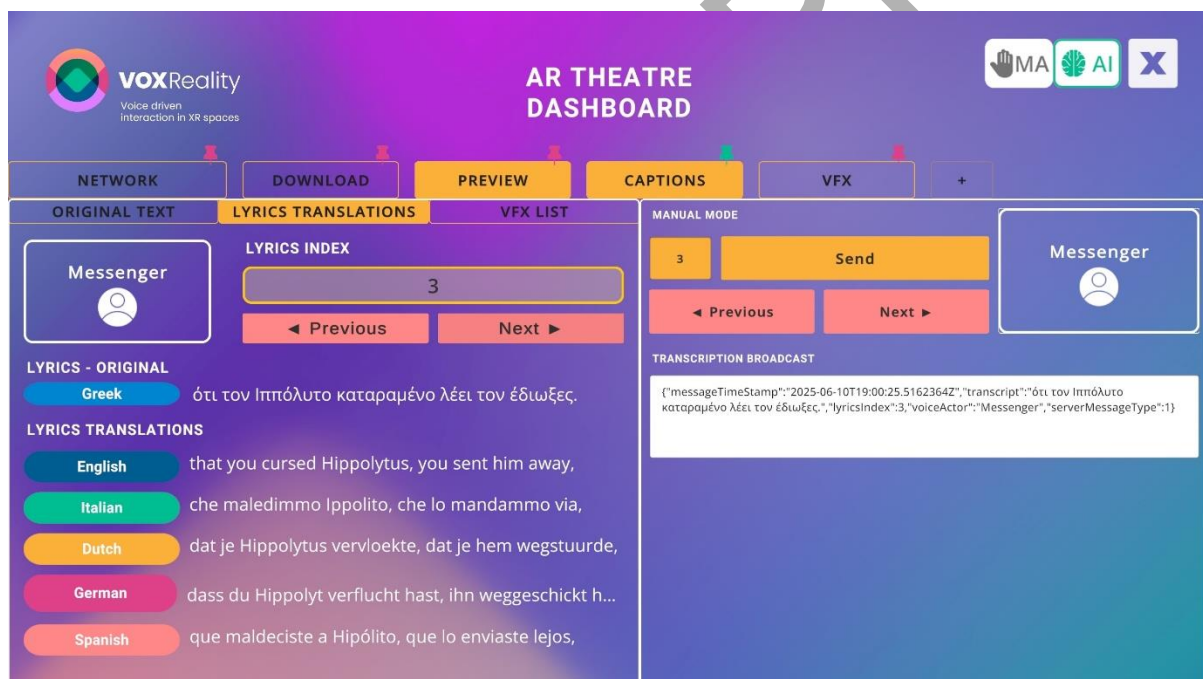


Figure 75. AR Theatre Server - manual mode interface setup

4.2.2.4.2 AR Theater AR client

The AR theatre client starts with a prompt for language selection by the audience (Figure 76). If the user is experienced, they can use their controller to select the desired choice. As another alternative, the venue operator can remote-control each individual client's application and set the language remotely based on the user's wishes to further facilitate the process.

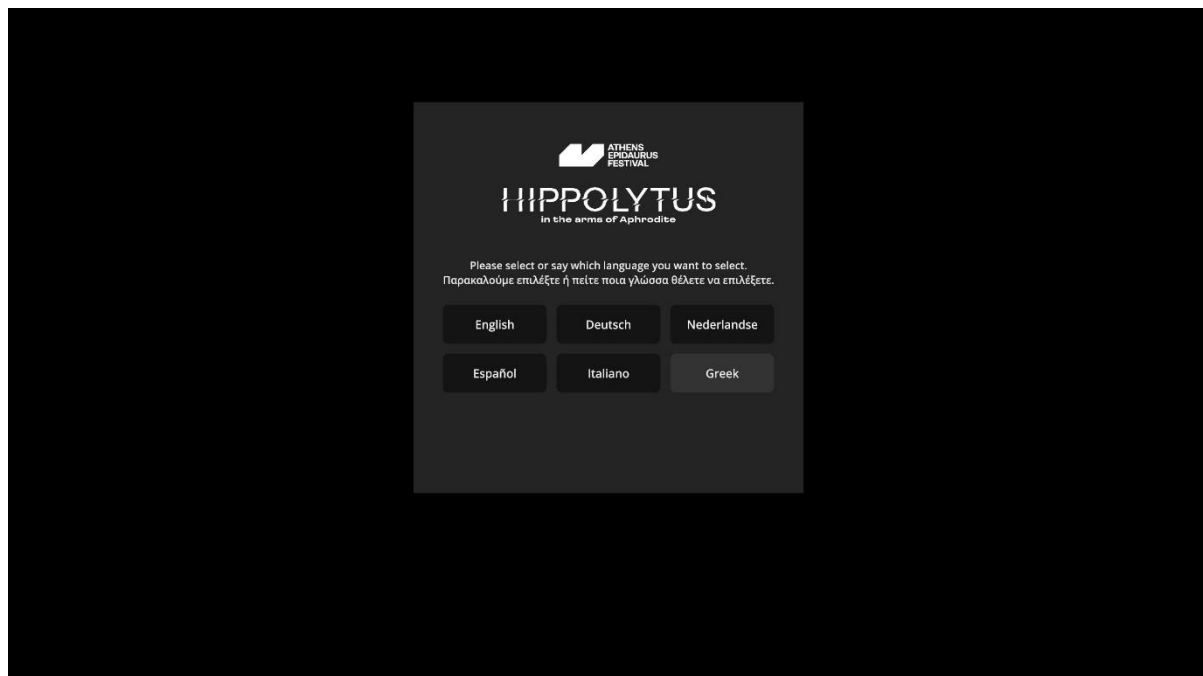


Figure 76. AR Client - Language selection menu

After the language selection, there are three preparatory steps that the user needs to complete in sequence before the start of the performance: the “introduction to device”, the “tutorial to application” and the “extras” for the play.

The “introduction to device” starts automatically for inexperienced users (with a remote-control command from the operator) and includes a hands-on step by step presentation to the device (Figure 77). It is mandatory to complete this step to be able to proceed to the next. As an exception for experienced users, this tutorial can be skipped. Alternatively, the venue operator can use a remote control command to skip this tutorial for users that explicitly request to do so.

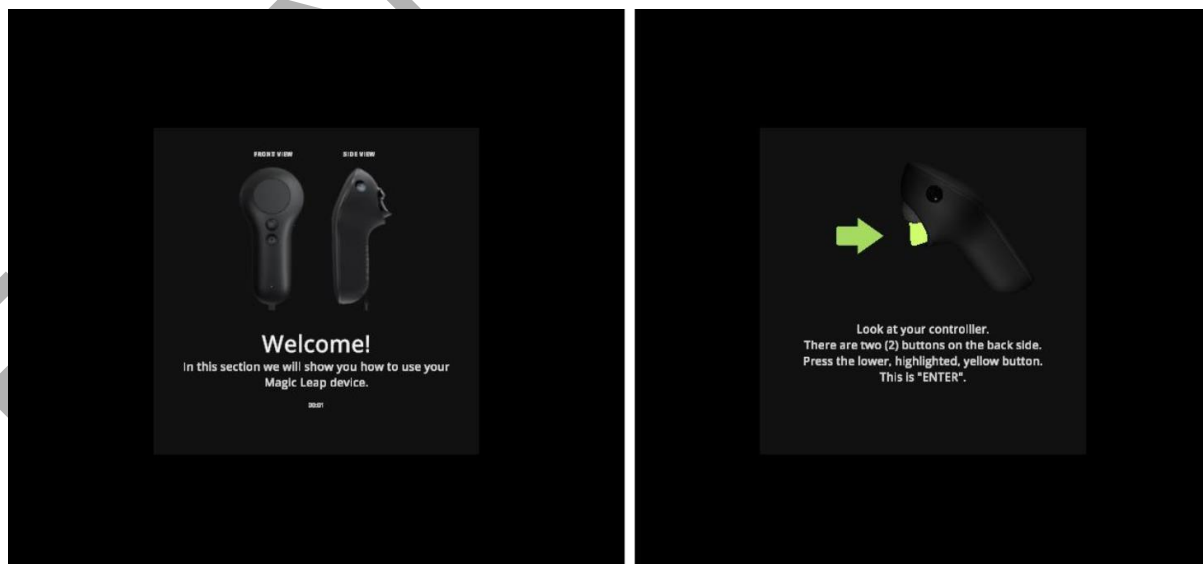


Figure 77. AR Client - Introduction to AR device

After the introduction to the device tutorial is completed, the user is provided with a menu panel (Figure 78) from which they can proceed to the next step, which is to complete the “tutorial to the application”. The “introduction to the device” button is available in case they

want to repeat the content. The buttons for the upcoming phases, i.e. “extras” and “performance”, are visible but locked, and unlock only when the user has successfully completed the previous steps.

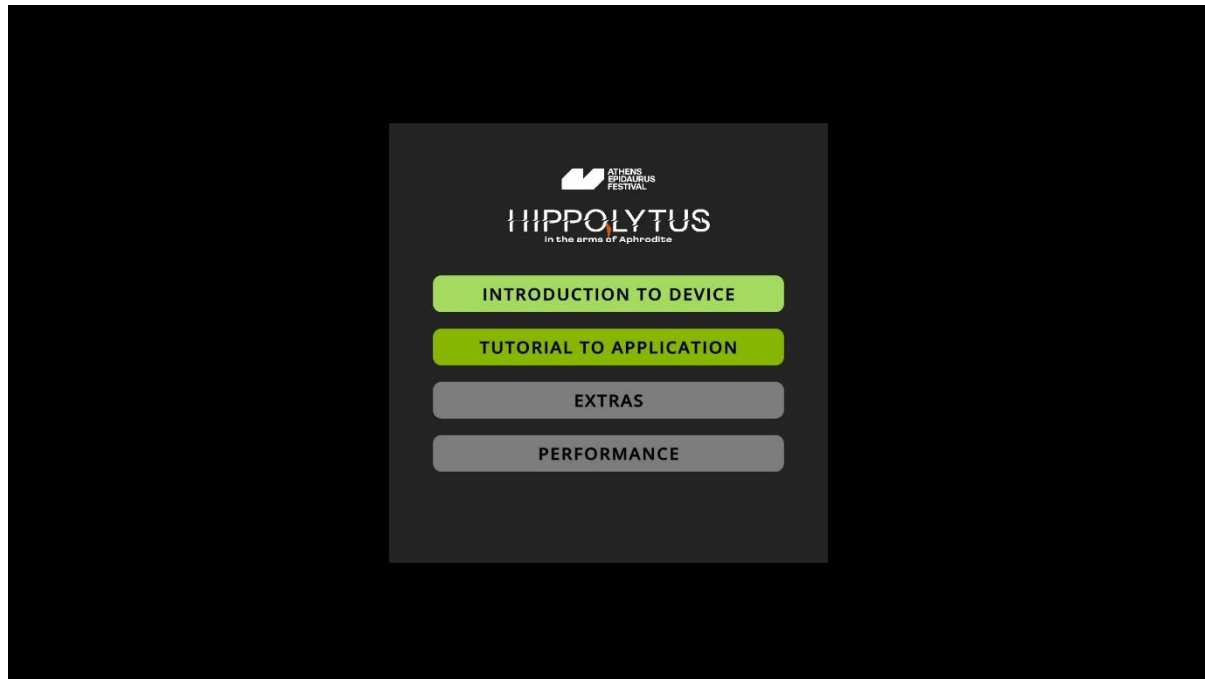


Figure 78. AR client – Menu panel with scene selection

The tutorial to the application contains a step-by-step explanation and hands-on experimentation for all provided features of the application. The features relate to customizing the display of the captions for readability and user comfort. The customization menu can be toggle on/off with a press-and-hold button input method and provides the following customization options: 1) (re)configuring the user language, 2) toggling subtitles on or off, 3) setting the position and movement behavior of the captions, 4) setting the font size and weight of the captions, 5) setting the background opacity for higher contrast and inverting the colour foreground/background scheme (from black to white, and vice versa) (Figure 79).

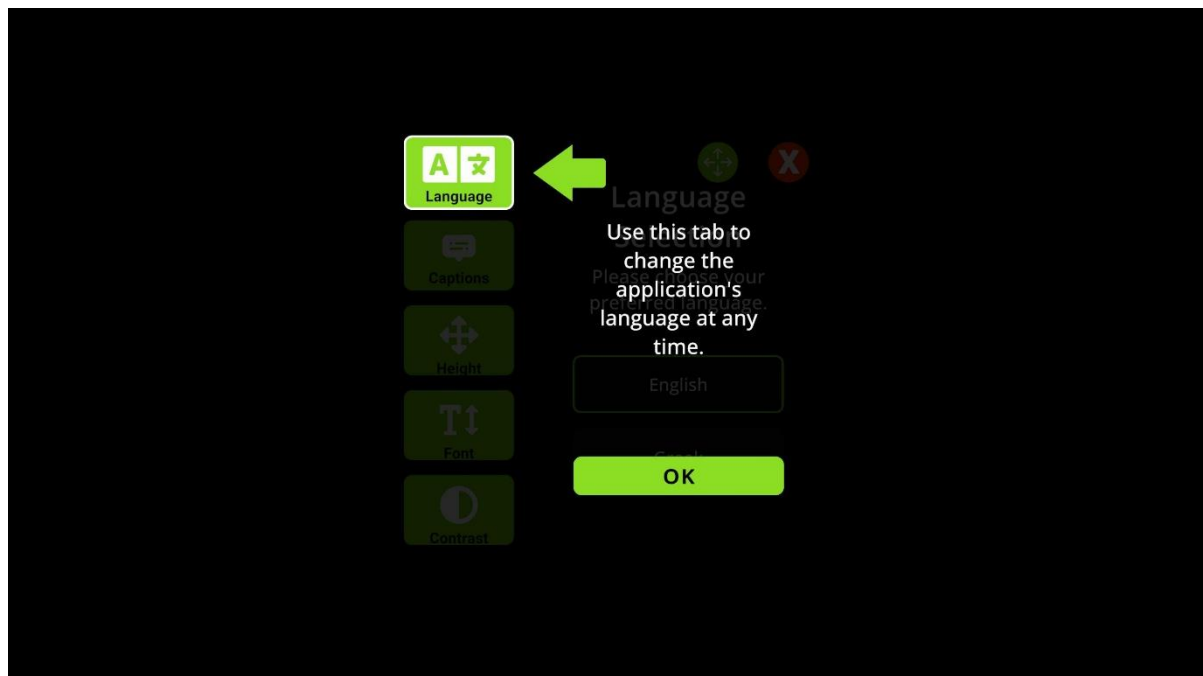


Figure 79. AR Client - Tutorial to application

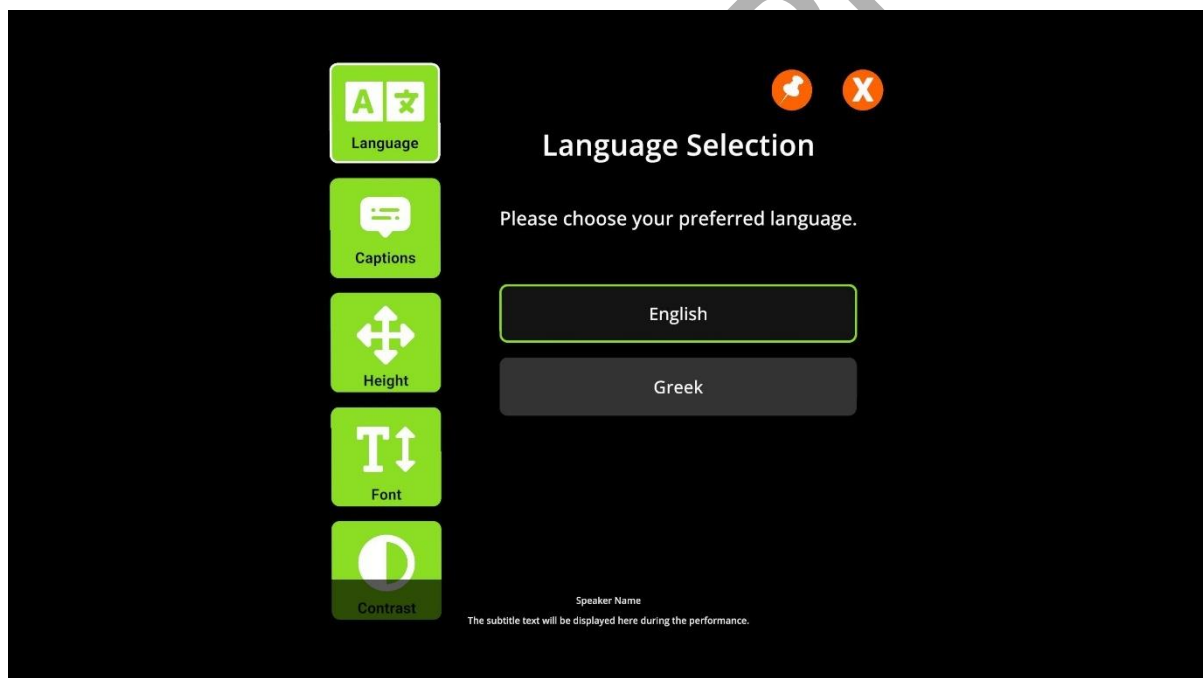


Figure 80. AR Client - Subtitles customization menu

Lastly, when the user has completed the tutorial to the application, they are provided again with the home menu panel, but this time the extras button is now available for interaction. The extras scene contains information about the theatrical play, the AR performance and the production alongside chosen graphical elements.

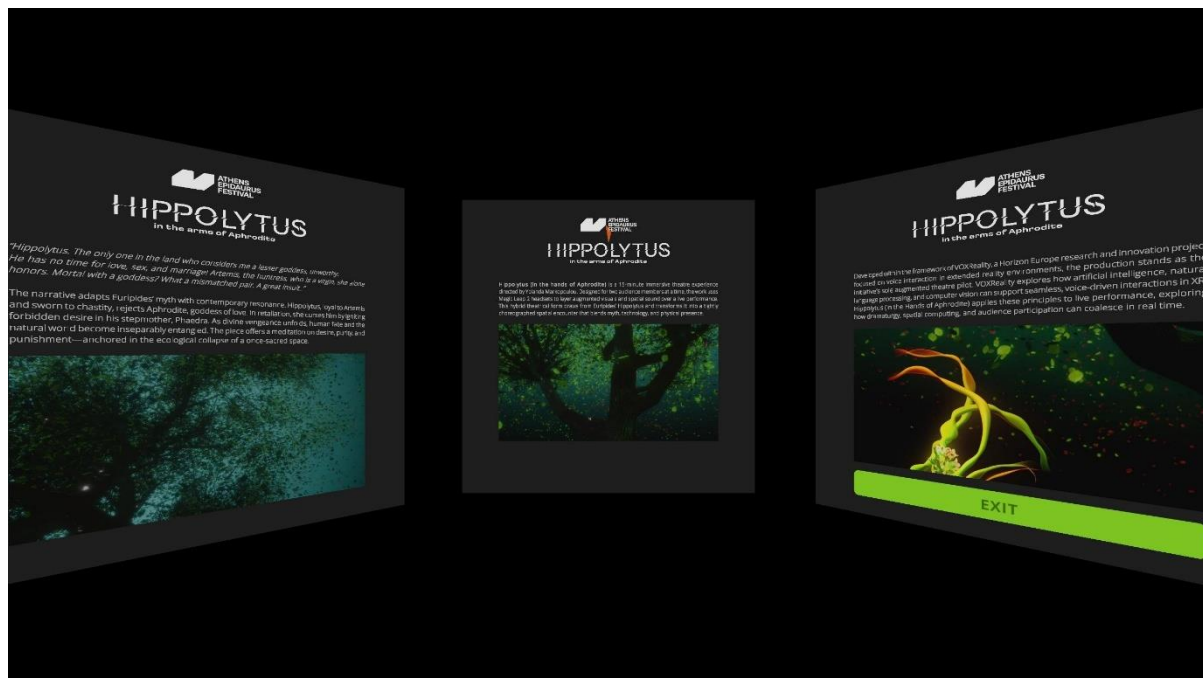


Figure 81. AR client - Extras scene content

After the extras scene has also been visited, the performance button is now available for interaction. Alternatively, the venue operator can remote-control all clients to trigger the transition to the performance scene. After a short loading intro screen, the user will experience the captions, as configured using the customization settings during the “tutorial to application” (their configuration is stored across scenes), and the VFX, as triggered by the received keywords (Figure 83).

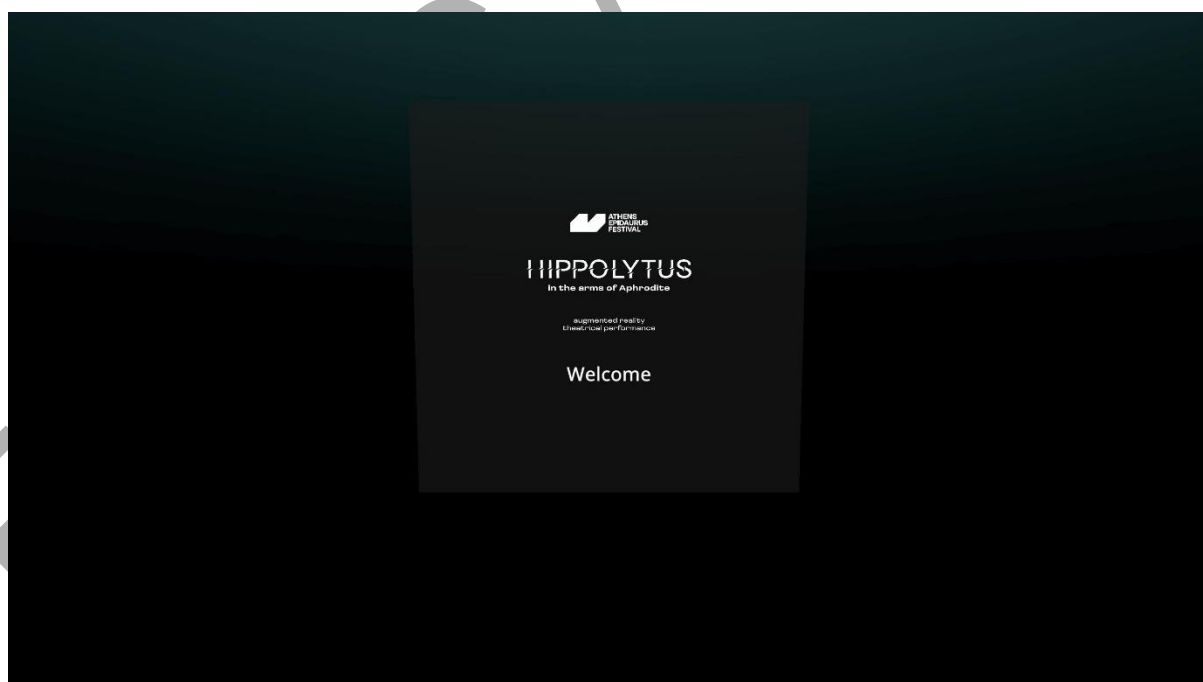


Figure 82. AR Client - Start of performance

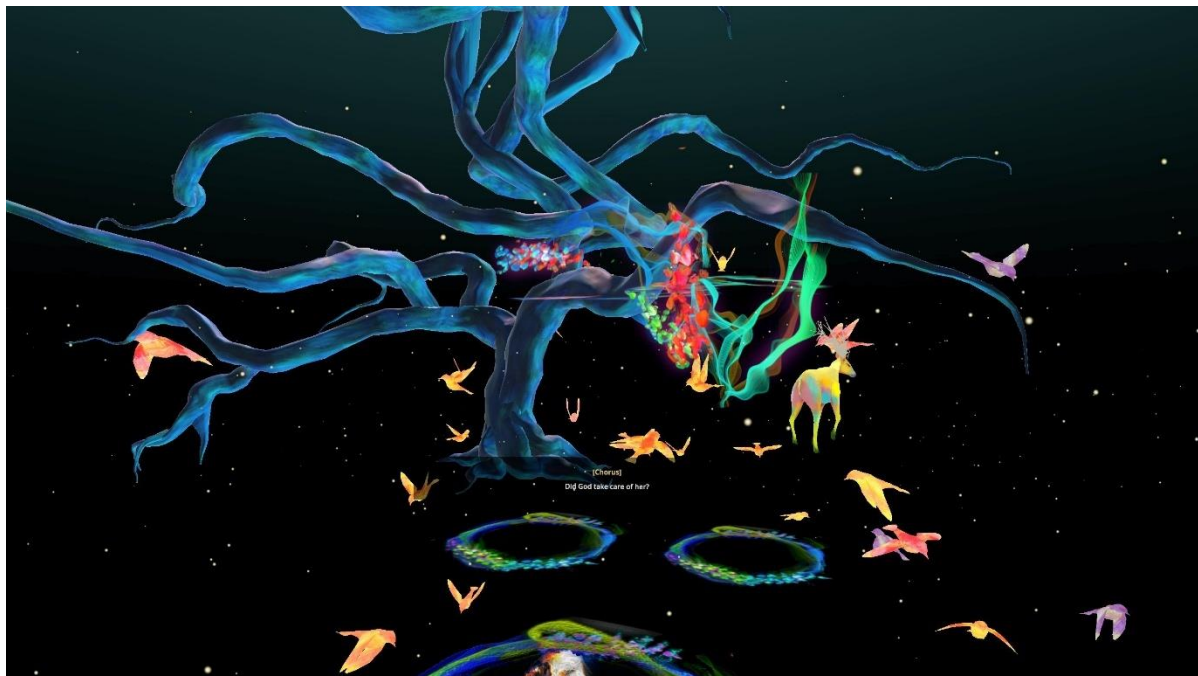


Figure 83. AR Client - Sample scene of performance featuring captions and triggered VFX

4.2.2.4.3 AR Theatre audio player client

The optional component of the AR Theatre audio player was created to deliver audio effects in sync with the visual effects. Although the visual effects are delivered as AR content through each individual AR device, the audio effects were not adequately serviced through the limited capabilities of the AR device speakers. Instead, a 5.1 audio surround system was targeted to provide an immersive audio feeling and a collective experience across the audience. Manual synchronization of the digital audio layer with the AR visual layer was very difficult for an operator without AR glasses though. To cover the synchronization need in an automated way, the audio player client was designed. Like the AR clients, the application has all creative content (music files) included in the executable and has a WebSocket connection during the performance. Similarly to AR clients, it responds to the received VFX trigger keywords, but with audio playback instead of visual rendering. The user interface can be seen in Figure 84.

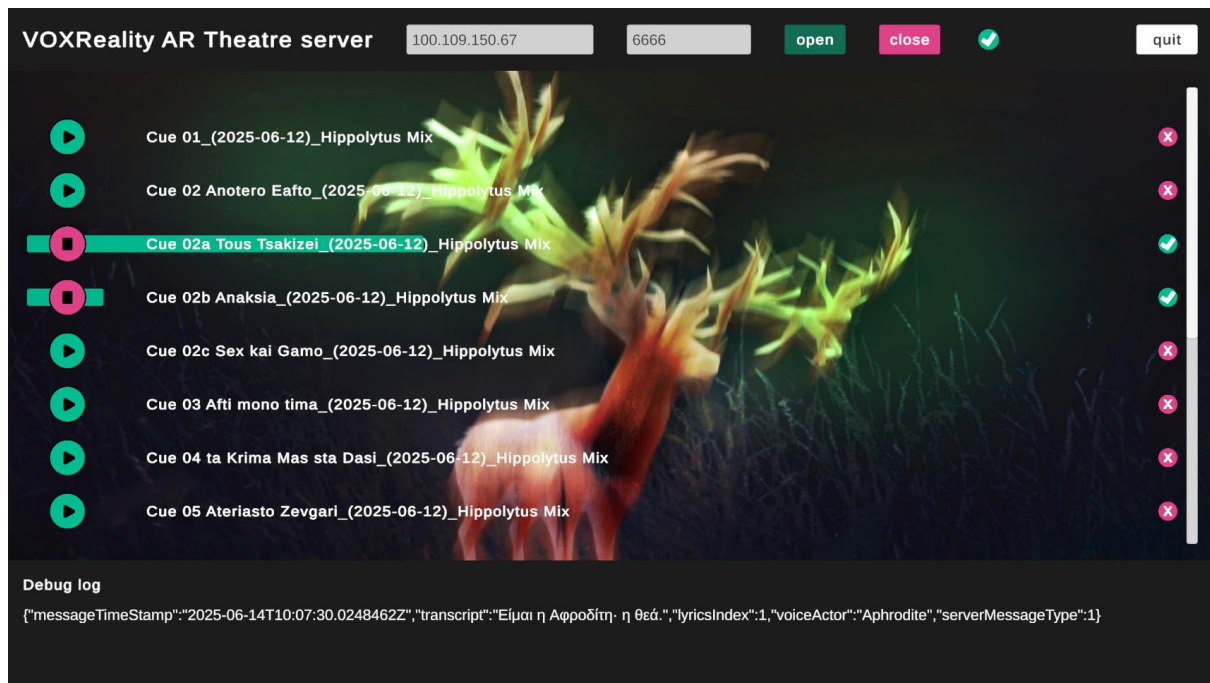


Figure 84. AR Theatre audio player user interface

4.2.2.5 Summary of Achieved User Requirements

The majority of the user requirements for the AR Theatre use case have been achieved for Pilot 2 (37 achieved out of 48). Table 7 presents the full list of requirements with their revised priority, achieved status and a short justification for the user requirements marked as “not achieved” in the “Reasoning” column. Not achieved user requirements are 9 in total, and are listed as follows: 2 user requirements relate to participant recruitment, therefore are not technical and not pertinent to WP4 (#14 and #15), 1 user requirement is marked as not technically feasible by AR technology (#24), 1 user requirement is marked as not implemented due to dependency on body tracking technology (#33), 3 user requirements are marked as not implemented due to dependency on text to speech technology (#35, #36, #37) and 2 user requirements are marked explicitly as not desirable by the theatrical director (#39, #47). For further context, text to speech technology (#35, #36, #37) was not preferred for operational reasons: using one’s earphones to listen to the synthetic generated speech would be at the cost of losing out on the original voice of the actors and the spatialized audio surround of the stage, which are core artistic elements. Contrary to this, the body tracking technology (#33) was not implemented due to technological challenges that are out of the scope of this research project. Finally, the theatrical director opted to allocate requirement #39 (“The director specifies the capability of the users to change AR Glass settings and subtitles”) to HCI expertise, and considered requirement #47 (“Audience can read about the play and characters or fun facts during the session”) as likely to trigger “fear of losing out” in the audience and negative impact attention and focus during watching the play. Instead, the requirement #48 (“Audiences can watch “behind the scene” (BTS) of the play before and after watching the play) was considered more important and less conflicting with the live performance.

Table 7. AR Theatre - Achieved user requirements

#	Type	Requirements	Revised Priority	Achieved in Pilot 2	Reasoning
1	Objective	The objective is to introduce AR technologies to theatre audiences.	High	yes	

2	Objective	Language translation and VFX experience with AR glasses at the theatre.	High	yes	
3	General	The experience is in AR.	High	yes	
4	General	The gestures and setup to operate AR glass should not intervene other audience's theater experience.	High	yes	
5	General	The possibility of the audience getting distracted by the additional information from AR glasses has to be considered.	Medium	yes	
6	Setup	The experiment takes place at the avant-garde theatre.	High	yes	
7	Setup	The AR glass setup should be easy and clear to audiences who are not familiar with the AR technology.	High	yes	
8	Setup	AR glass should give the audience enough controls but also limit their ability to change default/preset settings.	Medium	yes	
9	Scenario	A scene from the play "Hippolytus" by Euripides will be specially produced and played.	High	yes	
10	Scenario	The length of the performance will not exceed 15 minutes.	Medium	yes	
11	Assessment	Two users can watch the play at the same time with two AR devices in a single time.	High	yes	
12	Assessment	The target user is a theatergoer with diverse background whose native language is any of the VOXReality languages.	High	yes	
13	Assessment	The target user includes who don't know the spoken language of the play.	High	yes	
14	Assessment	The target user includes who cannot hear well.	Low	No	not a technical requirement
15	Assessment	The target user base includes people with vision correction glasses.	Low	No	not a technical requirement
16	Assessment	In Greek play, non-Greek audiences should participate in the experiment.	Medium	yes	
17	Assessment	Up to 50 people will participate in the complete experiment.	Medium	TBC	
18	Assessment	The translation of the play will be available in all VOXReality languages.	High	yes	
19	Assessment	The audiences may experience either one or both of the automatic translation and the visual effects technology.	High	yes	
20	Assessment	The user evaluation may include combined demonstration of translation and VFX components	Medium	yes	

21	Assessment	The creators of the performance may be asked to evaluate the experience through semi-structured interviews.	High	yes	
22	Interface	The AR glass could have menu on the screen to change practical settings.	High	yes	
23	Interface	Audience can learn about the play or the scene.	Medium	yes	
24	Interface	Audience can zoom in or out the scenes.	Low	No	not feasible
25	Interface	The elements in the AR glass screen should be feasible respective to the lighting, brightness, contrast and related settings of the stage (theater).	Low	yes	
26	Interface	The interface should not include extensive head movements.	Medium	yes	
27	Interface	The UI of the AR screen should consider audience accessibility.	Low	yes	
28	Subtitles	The default subtitle starts with the local language.	Low	yes	
29	Subtitles	The subtitles should be real-time and available in any of the VOXReality languages	High	yes	
30	Subtitles	The subtitles should provide different caption sizes and toggle options wherever feasible.	Medium	yes	
31	Subtitles	The audience should be able to fine-tune the placement of the subtitles.	Low	yes	
32	Subtitles	The subtitles should not overlap with the stage setup.	High	yes	
33	Subtitles	The subtitles could follow each actor depending on the context.	Low	No	no body tracking implemented
34	Subtitles	There should be various caption styles (standard, speech bubble etc).	High	yes	
35	Subtitles	The default audio starts with the original spoken language of the play.	Low	No	no text to speech (TTS) implemented
36	Subtitles	The audience should have an option to change the language audio.	Low	No	not pertinent if no TTS available
37	Subtitles	The audience should have option to listen to the original language.	Low	No	not pertinent if no TTS available
38	VFX	The VFX implementation should be discussed earlier with script and the plot.	High	yes	
39	VFX	The director specifies the capability of the users to change AR Glass settings and subtitles.	Low	No	not desirable
40	VFX	VFX can reflect or accompanies the narration of the scene.	High	yes	
41	VFX	Certain words or phrases will trigger VFX.	High	yes	

42	VFX	A narrator could be the person who triggers the VFX.	Low	yes	
43	VFX	Location of the visual effects should not exceed the range of the stage.	Low	yes	
44	VFX	The implemented VFX should not affect the actor's actions on the stage.	Medium	yes	
45	VFX	The VFX should help the audience be immersed in the performance.	Low	yes	
46	VFX	Style of the visual effects should be artistically relevant to the scene.	Medium	yes	
47	Extra	Audience can read about the play and characters or fun facts during the session.	Medium	No	
48	Extra	Audiences can watch "behind the scene" (BTS) of the play before and after watching the play.	Low	yes	

4.3 Training Assistant

The AR Training Assistant application brings intelligent support into industrial training environments by leveraging augmented reality and voice interaction. At its core is a Virtual Assistant, capable of understanding natural language and responding to users through speech, enabling hands-free and intuitive guidance during complex assembly tasks. Designed to improve the overall training experience, the Assistant provides real-time instructions, contextual task support, and on-demand clarification throughout the session. This AI-driven solution is intended to be a seamless and supportive presence in the training process through voice interaction. An ASR along with a DA from VOXReality project enable trainees get visual and auditory support while they can interact with the agent through natural speech. The agent can also trigger actions on behalf of the user on demand. These functionalities of the VOXReality models enable the user to freely interact with the virtual 3D CAD models and train in an AR space.

4.3.1 System Architecture and Design

4.3.1.1 3D Models and Scenes Design/Creation

The scene in the Training assistant application contains a 3D CAD model of a Raptor engine, a table on top of which all the grabbable objects and the tools are placed. On the table, there are grabbable parts of the Raptor engine, pins, nuts and bolts, screws, screw driver, QR code scanner are present. When the user starts the training the VOXY avatar launches itself to introduce and guide. The VOXY avatar is associated with a display panel where the texts, videos are shown. The avatar and the panel will move to a dedicated spot on the table after introduction. The scene has visual elements which have audible inputs during user interaction. The images below (Figure 85, Figure 86, Figure 87, Figure 88, Figure 89) show some examples of the scene in its preliminary stage. The scene was redesigned eventually to fit an open-ended training mode (free mode). The images are taken as screenshots from the windows PC without connecting to the glasses which is why the see-through effect of the glasses is absent and the background is black. The user however, would see the surrounding through the glasses along with the augmented training scene. This can be very useful when the user wants to compare or fit the 3D object in the scene with the objects in the real world.

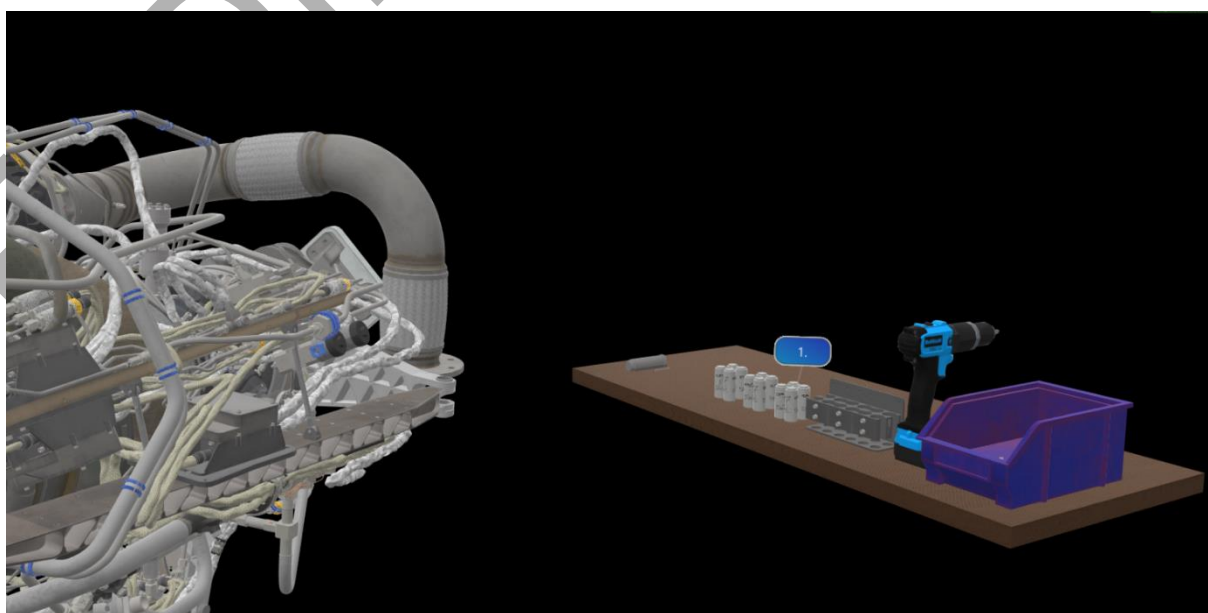


Figure 85. 3D scene with Raptor engine on the left and the table on the right

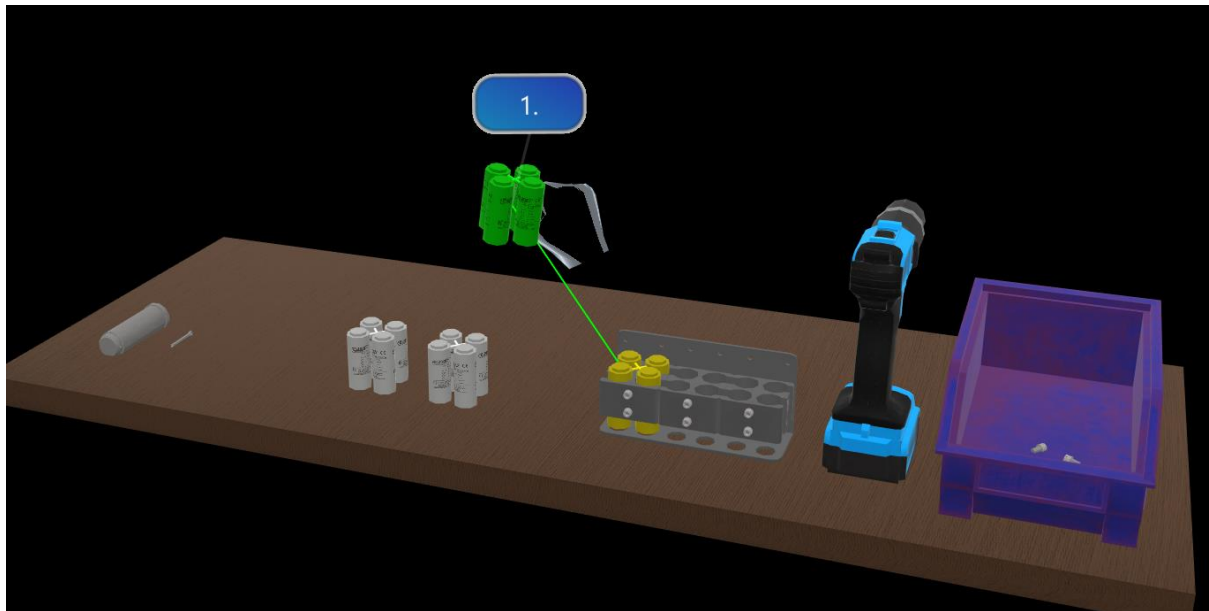


Figure 86. 3D scene with table and interactive objects – grabbed object in green and destination in yellow with a green guiding line. Object has a tool tip.

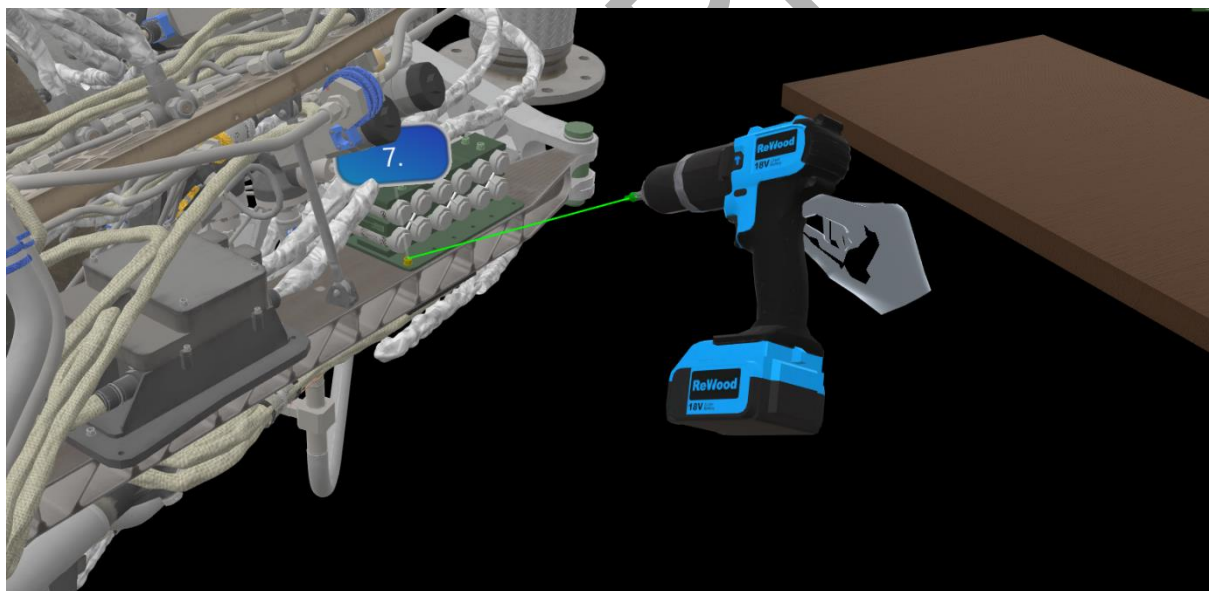


Figure 87. The screwing logic with audio feedback

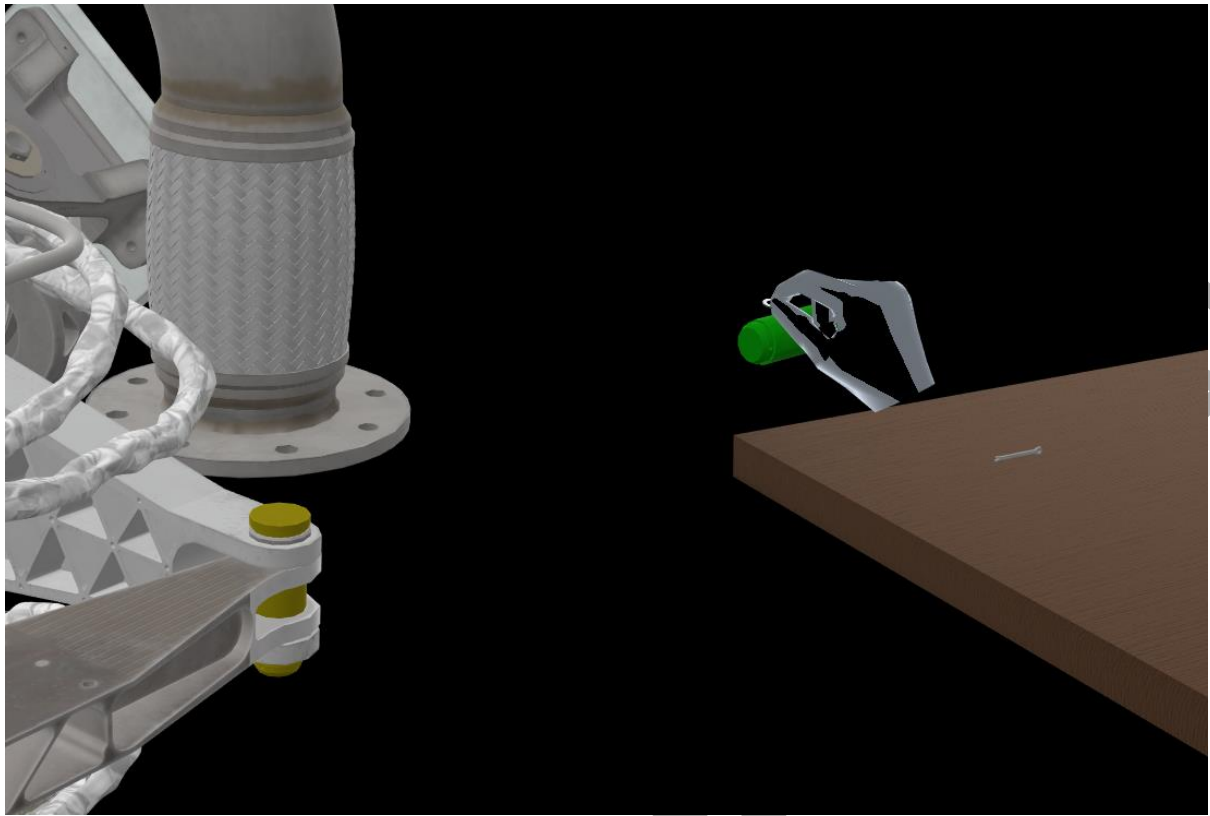


Figure 88. 3D scene with table, Raptor engine and assembly

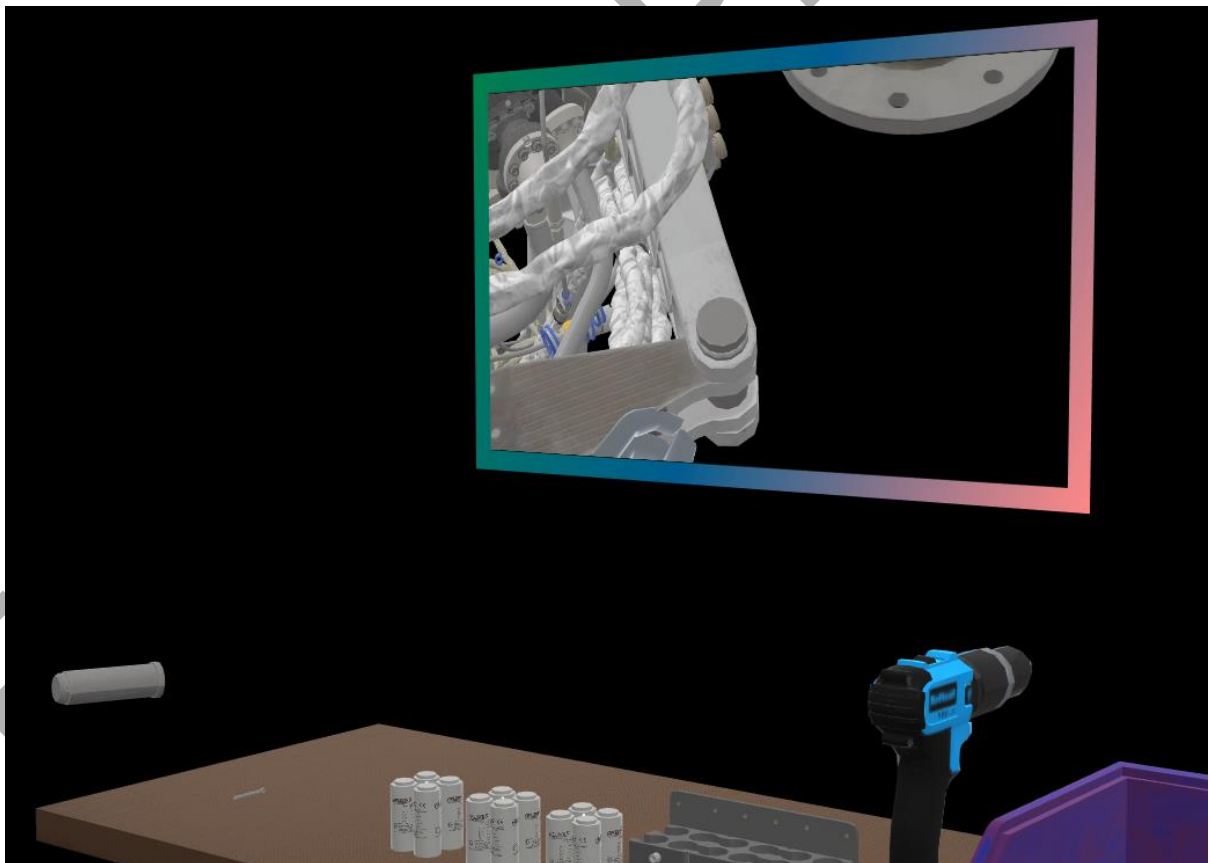


Figure 89. 3D scene with table and display panel with a video being played.

4.3.1.2 Application Workflow Diagram

The architecture of the application (Figure 90) integrates both internal and VOXReality software components to deliver a robust and intelligent AR training experience. Central to the VOXReality are the AI-driven modules that enable advanced user interaction through natural

language: the VOXReality ASR model and the VOXReality Dialogue Agent (DA), also referred to as ARTA. These services act as the cognitive layer of the system. The ASR module captures speech input from the user via HoloLens's built-in microphones. It processes the audio in real time and converts it into structured textual input, which is then forwarded to the DA.

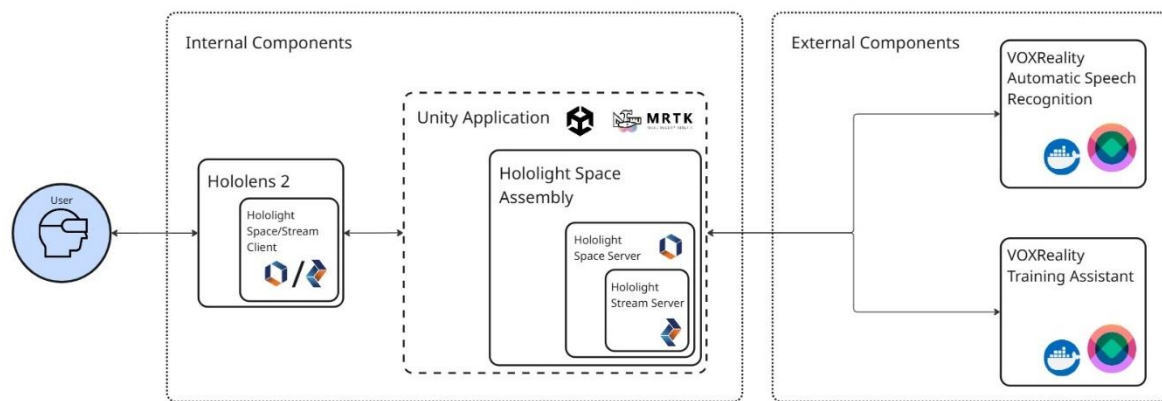


Figure 90. Training Assistant application workflow

The DA interprets this transcribed speech to understand user intent and context within the training scenario. It relies on a combination of natural language understanding, contextual awareness, and interaction history to determine the most relevant support action. This could involve answering a user query, guiding the user through an assembly step, providing feedback about the current state of the task, or triggering an action on the Unity application side. The actions involve showing a video of a particular step, giving a hint about the step, skipping a step at hand, undo a step, and resetting the training scenario on demand.

In the architectural flow, these AI services are tightly coupled with the internal logic of the application, particularly the open-ended Free Mode training system. As the user engages with objects in the MR environment, contextual data—such as the currently selected or manipulated object and the training step in progress—is transmitted to the Training Assistant. This ensures that responses from the AI are relevant and tailored to the user's current task and progress.

Together, these components form a closed feedback loop: user actions and speech inform the AI, and the AI responds with targeted guidance, creating a seamless and intelligent mixed reality training workflow. This architecture not only enables a high degree of interaction flexibility but also supports adaptive training experiences that respond to individual user behavior and learning needs.

The technological backbone of the application is built upon a suite of Hololight components—Hololight Stream, Hololight Space, and Hololight Space Assembly—which collectively enable real-time, collaborative mixed reality experiences tailored for industrial training. Hololight Stream is the foundation for high-performance remote rendering, allowing computationally demanding visual content such as high-resolution CAD models and complex 3D objects to be streamed directly to mobile devices. This approach overcomes the hardware limitations of standalone headsets by offloading rendering to a remote high-performance server, ensuring both high visual fidelity and low latency through secure web real-time communication protocols which enhance security.

Hololight Space extends this infrastructure by providing an industrial grade augmented and virtual reality platform for visualizing and interacting with complex 3D models in a shared virtual environment. It operates using a client-server architecture, where the headset acts as a client

connected to a powerful rendering server. This design enables multi-user collaboration, real-time interaction, and dynamic content updates within the same spatial context.

Central to the training functionality of the application is Hololight Space Assembly—an experimental module developed on top of Hololight Space. Originally designed for closed-ended or linear training workflows, Assembly facilitates industrial assembly line training through immersive interaction with high-resolution CAD models. It supports the import of structured training files, which define both the sequence of assembly steps and the initial configuration of virtual components. Training tasks include physical actions such as grabbing, positioning, and using virtual tools—for example, simulating the use of an electric screwdriver. The training is guided by intuitive visual cues like numbered tooltips, trajectory lines, and color-coded targets, with difficulty levels ranging from “Easy” to “Hard.”

This application customizes and extends the Assembly module significantly, transforming it into an open-ended, adaptive training scenario. The current work emphasizes experimental user interface and user experience (UI/UX) approaches to explore flexible training flows, enabling a more natural and user-driven learning process that maintains the immersive and interactive strengths of the underlying Hololight technology stack.

Open-Ended Training System (Free Mode):

Originally, the Assembly experimental module was designed for step-by-step training scenarios. These scenarios did not align with the concept of an open-ended training environment, as they required tasks to be completed in a predefined order. In “Easy” mode, for example, users could not attach any object other than the one specified in the current step; if attempted, the attachment animation would not occur. In “Medium” and “Hard” modes, although attaching the incorrect object was technically possible, the system would mark it as an error and the training could not be successfully completed.

To support the integration of AI models and enhanced learning, a new difficulty setting called “Free” mode is implemented (Figure 91). In this mode, training steps can be performed in any order, except for those steps that are logically dependent on others. The system does not guide the user toward a specific next step; instead, users are free to begin assembling objects based on their own decisions.

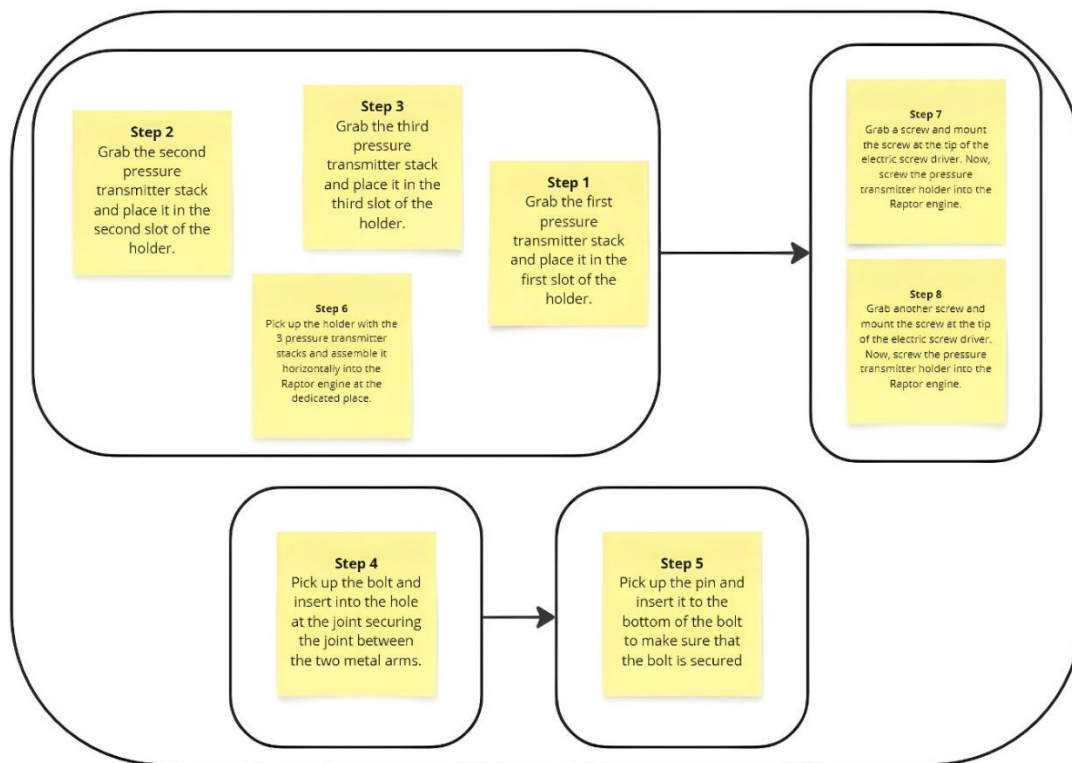


Figure 91. The logical scheme of the training steps in Free Mode¹⁴.

In the Free Mode, the step number no longer indicates execution order but serves only as an identifier. Since the system cannot determine the current step based on a predefined sequence, a new mechanism was created to infer which step the user intends to perform at any given moment. Each step corresponds to a training object that needs to be assembled. The system assumes that the most recently grabbed object indicates the user's intended step. It retrieves the step number associated with the object and sets that as the current step for a limited duration. This remains the current step until either the user completes it, picks up another object (thus changing the intended step), or the set time elapses. If none of these conditions are met, the current step resets to a default value of -1, indicating that no specific step is currently active. This default value is also used to inform the Training Assistant when the user has not committed to any particular step.

Another crucial feature of Free Mode is the dependency system, which ensures that users cannot complete steps out of logical order when dependencies exist. Each step may have a list of prerequisite steps that must be completed beforehand. When a user grabs an object, the system checks whether all dependencies for that step are satisfied. If not, the object cannot be attached, and the step cannot be completed. However, to maintain the exploratory nature of the experience, the system does not immediately inform the user that a step is invalid unless assistance is explicitly requested.

Table 8. The association between step number, training object, and the related manual instructions

¹⁴ The steps inside a box can be done in any order; the arrows between two boxes indicate a dependency, meaning that all the steps inside a box need to be done before doing any step inside the box pointed by the arrow.

Step #	Training Object	Manual Instruction
1	Pressure Transmitter #1	Grab the first pressure transmitter stack and place it in the first slot of the holder.
2	Pressure Transmitter #2	Grab the second pressure transmitter stack and place it in the second slot of the holder.
3	Pressure Transmitter #3	Grab the third pressure transmitter stack and place it in the third slot of the holder.
4	Metal Arm Bolt	Pick up the bolt and insert it into the hole at the joint, securing the joint between the two metal arms.
5	Metal Arm Pin	Pick up the pin and insert it into the bottom of the bolt to make sure that the bolt is secured
6	Pressure Transmitters Holder	Pick up the holder with the 3 pressure transmitter stacks and assemble it horizontally into the Raptor engine at the dedicated place.
7	Screw #1	Grab a screw and mount the screw at the tip of the electric screw driver. Now, screw the pressure transmitter holder into the Raptor engine.
8	Screw #2	Grab another screw and mount the screw at the tip of the electric screw driver. Now, screw the pressure transmitter holder into the Raptor engine.

4.3.2 Implementation Details

4.3.2.1 Development Environment Setup

The development environment is described in Table 9 for Hardware and Table 10 for Software:

Table 9. Development Environment Specifications – Hardware

Component	Specifications
Operating System	Windows 11 Pro
CPU	Intel Core i7-9750H @ 2.60GHz
Memory	16GB
Dedicated GPU	NVIDIA GeForce RTX 2060
Dedicated GPU Memory	6GB

Table 10. Development Environment Specifications – Software

Component	Specifications
Unity version	2021.3.30f1
Unity components	Universal Platform Build Support Windows Build Support IL2CPP
Mixed Reality Toolkit version	2.8.3
Hololight Stream version	2023.1.0
Hololight Space version	2024.0.0

The application is specifically developed for the Microsoft HoloLens 2 mixed reality device, selected for its advanced spatial computing capabilities and suitability for delivering immersive experiences. The internal software architecture consists of several critical components. At its core is the Unity game engine, a widely adopted 3D development platform known for its robust support for mixed reality applications. To further streamline development, the Mixed Reality Toolkit 2 (MRTK2) is employed. MRTK2 is a comprehensive software development kit

provided by Microsoft that facilitates cross-platform MR application development within Unity, offering built-in support for the HoloLens 2 and other compatible devices. These tools collectively provide the foundation for implementing the core application logic and interaction mechanisms.

4.3.2.2 Core Algorithms and Techniques

This section outlines the VOXReality software components integrated into the system.

Automatic Speech Recognition:

Within the VOXReality project, an ASR model has been developed to support natural language transcription. It provides essential functionality to convert speech recordings into textual output and serves as a core input method for enabling voice interaction within the training environment. The ASR model is easily deployable through Docker by pulling it from the VOXReality Docker Hub. Communicating with the deployed model happens via REST API. The basic functionality needed for this use case is speech-to-text, and this is done by sending a POST request to the endpoint `<api-url>/transcribe_audio_files?source_language=en`, with inside the request body the .wav file containing the speech to be transcribed. The response to that request will be in the response body in JSON format as shown below:

```
{
  "translations": null,
  "transcriptions": [
    "Can you help me with this step?"
  ]
}
```

The application leverages Hologlight Stream for remote rendering. Consequently, speech input must be captured by the HoloLens microphone and transmitted to the server, which hosts the running Unity application, i.e., Space Assembly. While Stream inherently manages microphone access and data transmission to the server, a minor customization to its code was implemented. This modification enables the capture of audio segments (AudioClips) with a predefined, fixed duration. This fixed duration is necessary because the ASR API accepts .wav files as audio files, and the AudioClips to be converted into .wav files had to be of a predefined length. Additionally, this predefined duration directly corresponds to the recording window activated immediately after the user says the wake phrase "Hey Voxy", ensuring the relevant query is captured.

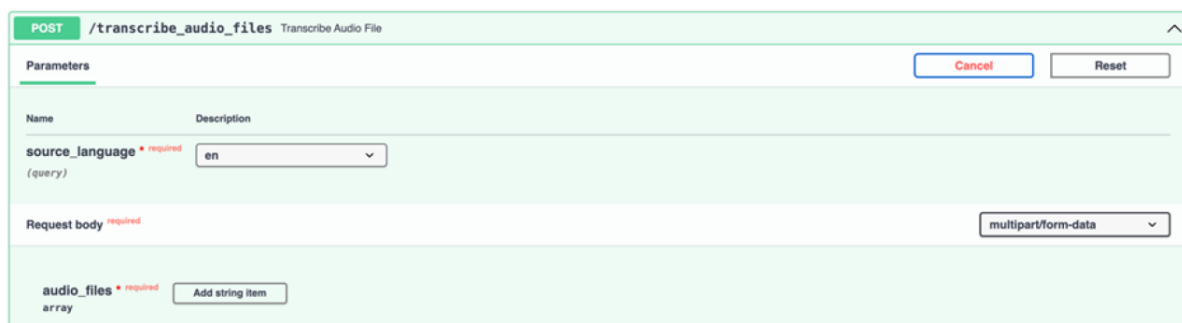
The image shows a REST client interface for a POST request to the endpoint `/transcribe_audio_files`. The request is titled "Transcribe Audio File". Under the "Parameters" tab, there is a query parameter `source_language` with a value of `en`. The "Request body" is set to `multipart/form-data`. There is a section for `audio_files` (array) with an "Add string item" button. "Cancel" and "Reset" buttons are visible in the top right.

Figure 92. The ASR transcription endpoint

Dialogue Agent:

This model leverages AI to guide users through virtual training scenarios. It supports users by interpreting their requests, providing context-aware responses, referencing the training manual, and executing relevant actions to assist during the simulation.

The ARTA model has evolved through several iterations, initially focused on closed-ended tasks. In the current form, it is capable of supporting open-ended interactions, allowing for more natural dialogue with the user. This capability enhances its effectiveness in training environments by delivering tailored and dynamic support based on user needs. Similarly with the ASR model, the VOXReality Training Assistant model is easily deployable through Docker by pulling it from the official VOXReality Docker Hub. Like the ASR, the DA is accessed via POST requests sent to its API endpoint: <api-url>/ask. The following is an example of a request-response pair. An example of the user request is as follows:

```
{  
  "question": "Who are you?",  
  "current_step": -1,  
  "wrong_object_flag": false  
}
```

The response from the DA is as follows:

```
{  
  "answer": "I am VOXY, a Dialogue Agent designed to assist you with the Assembly  
    ↪ process for the Raptor Engine. I can provide step-by-step instructions,  
    ↪ hints, and video demonstrations to help you complete the assembly  
    ↪ process.",  
  "current_step": -1,  
  "question": "Who are you?"  
}
```

Although the core communication is theoretically just sending a text prompt and receiving a text response, the primary challenge for this use case lies in providing the assistant with the current context during each request. This is because the model needs to know the current status of the training to generate accurate responses or activate functions at the right time. This means the Unity application needs to send extra information to the Assistant, precisely the current step number, which gets updated according to the Free Mode logic, and if the current step is a dependent step that cannot be done right now. Therefore, the POST request has the following fields: question, a string containing the user prompt, current_step, an integer indicating the current step number, and wrong_object_flag, a boolean which tells if the current step is doable or not based on the Free Mode's dependency system. The combinations of these parameters define the Training Assistant behavior in an open-ended training environment.

From the scheme, we can see all the different cases the agent is designed to cover. First, there is the classification of requests into Questions and Commands. Then, the current_step parameter is evaluated. A value of -1 indicates the user did not grab any object recently; the Assistant can then trigger the Start Training function or answer contextual questions, but cannot assist with specific steps (e.g., trigger "Show Video") and will prompt the user to grab an object. Finally, if current_step is a valid instruction number, the wrong_object_flag parameter becomes relevant. If this flag is true, the assistant avoids providing help for that step, instead advising the user to tackle a different step first. The figure below shows a scheme representing ARTA behavior based on users' requests and those combinations of parameters.

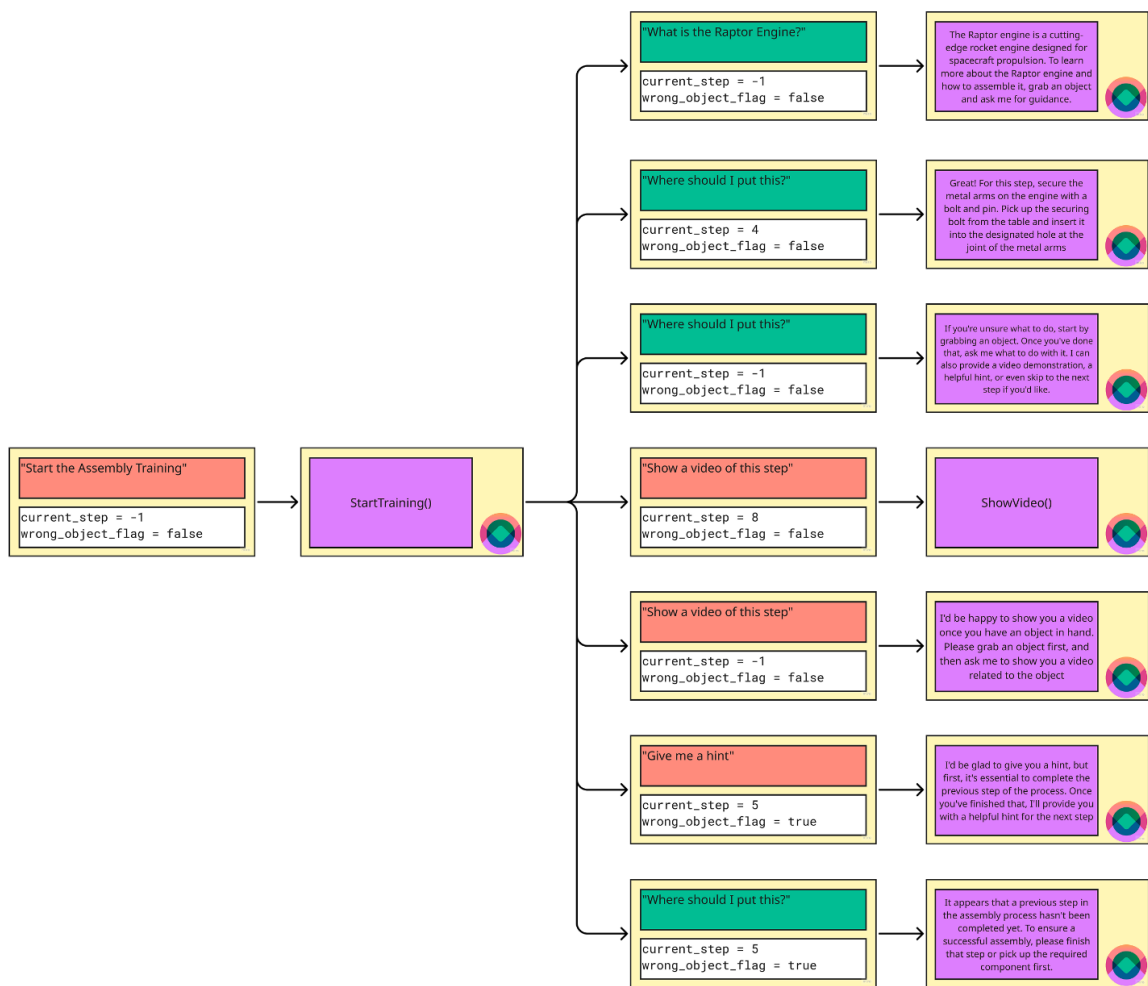


Figure 93. Scheme showing ARTA behaviour based on the user requests (Questions in green and Commands in red) and the other parameters

Training Assistant Avatar- Voxy:

The Training Assistant has a virtual presence in the scene through a 3D avatar called Voxy. A name plus an avatar makes the virtual assistant recognizable, enhancing perceived social presence, even if the avatar is not in a human form, but it is just a textured sphere which bounces to give the impression of being "alive". Studies suggest not giving human-like appearance can avoid the risk of falling into the uncanny valley, that instead can happen for embodied agents. Because of the afore mentioned aspects, users start interacting with the agent often, and they feel secure when they ask for assistance. Below are a bunch of images showing examples of the Training assistant supporting the user with its functionalities.

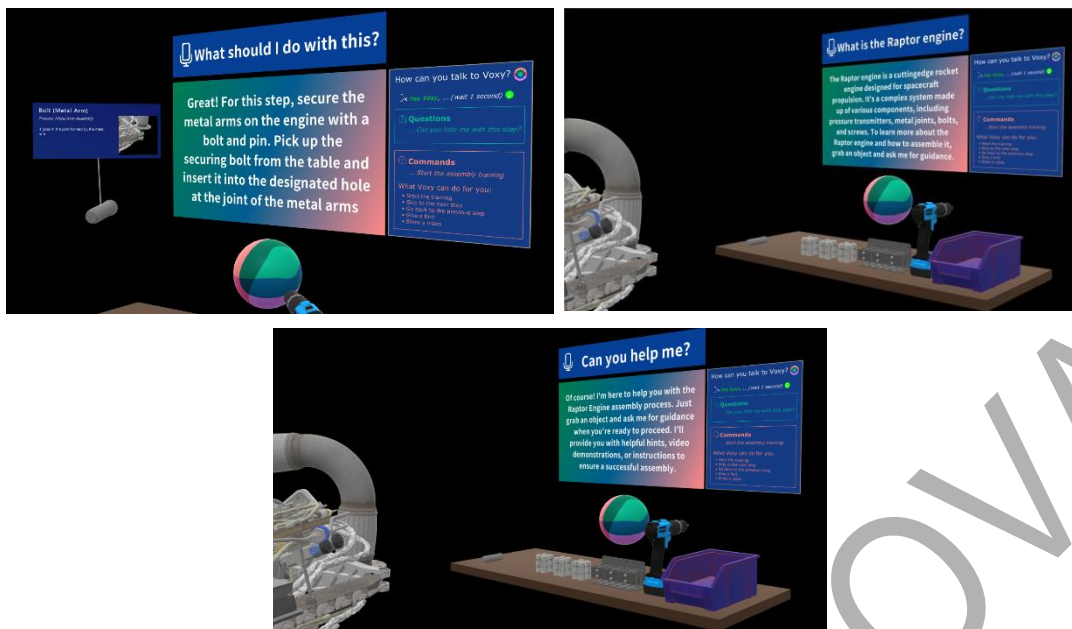


Figure 94. Examples of responses to user's questions



Figure 95. DA's response to user's request for help through video and hint

4.3.2.3 User Interface Implementation

This section describes the user interface (UI) solutions implemented in the training application.

Hand Interactions: The system uses MRTK2's input system to allow grabbing and moving virtual objects. Objects are highlighted in yellow at their target location and the grabbed objects are shown in green. When released, objects either attach (snap) to the destination or return to their original position after the timer runs out if dropped incorrectly.

Object Tooltips: Every training object has an associated tooltip displayed on a panel. When an object is grabbed, its tooltip appears on the panel, showing the associated instruction from the manual. This helps the user understand what the object is and the required action for the step.

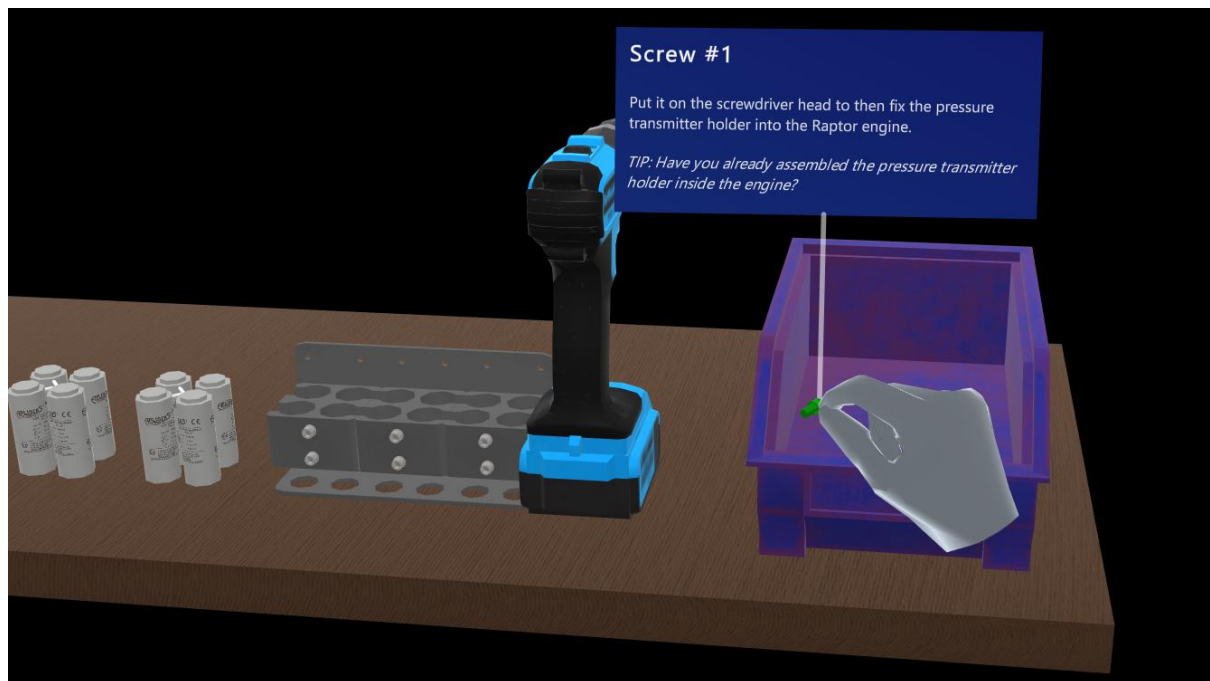


Figure 96. Example of object tooltip

The training assistant use case is characterized by the use of voice interactions with an AI assistant named Voxy. For the visual feedback regarding the voice interaction, a display panel is present on the table showing the text for all the voice inputs and outputs. It displays system messages (e.g., starting training), transcribed user requests, answers from the assistant, tutorials and alerts (e.g., inactivity or misplaced objects). It acts as a central visual communication tool.

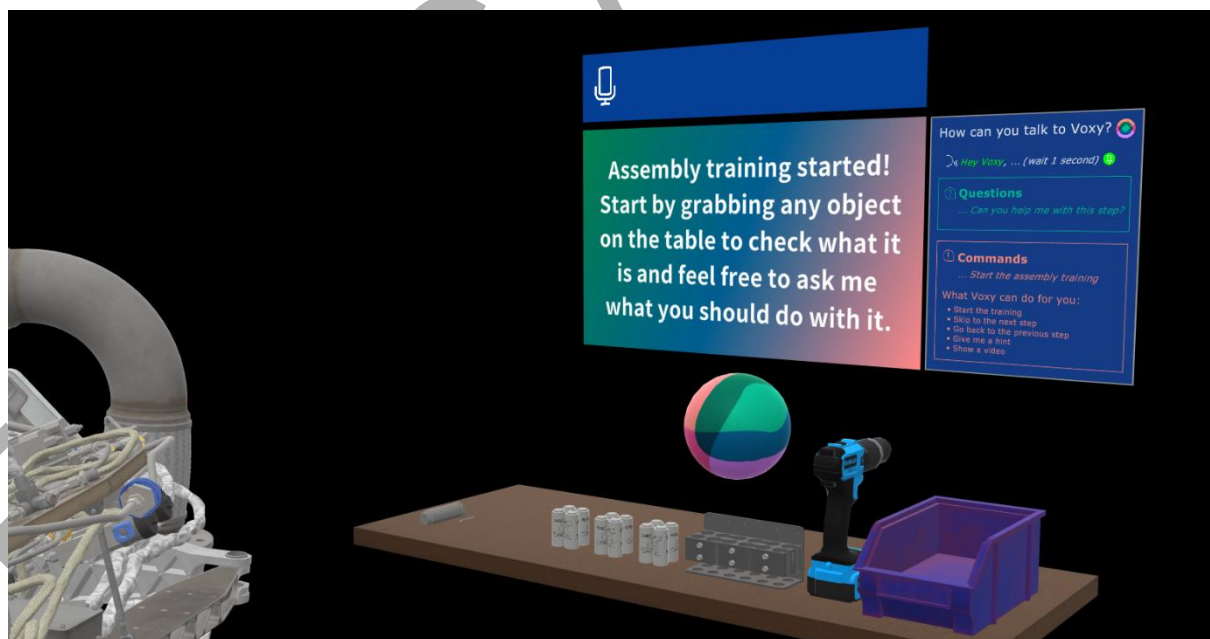


Figure 97. The display panel with visual feedback

Speech Input: A wake word, "Hey Voxy", is used to activate the assistant. After saying the wake phrase, the system records a fixed-duration audio clip which is transcribed and sent to the assistant. Speech recognition only works within this controlled timeframe. The recorded speech is then interpreted either as a question or command. When the user calls "Hey Voxy", a voice user interface icon pops up on the screen. The icon is a mic and when activated with "Hey Voxy" command, it turns green in order to give the user the feedback that the application

is listening. The Voxy avatar also turns green as a positive feedback. When the input is being processed, a loading symbol is shown. There are two types of voice inputs. The first being fixed MRTK commands where VOXReality models are not involved. These are used for UI interactions. For example, before the training is launched, the MRTK command “setup training” can be used. During the tutorial, to flip the pages, “Next” or “Back” can be used. However, once the wake word “Hey Voxy” is used, the voice input goes to the DA through ASR. This interaction will happen in natural language without keywords.

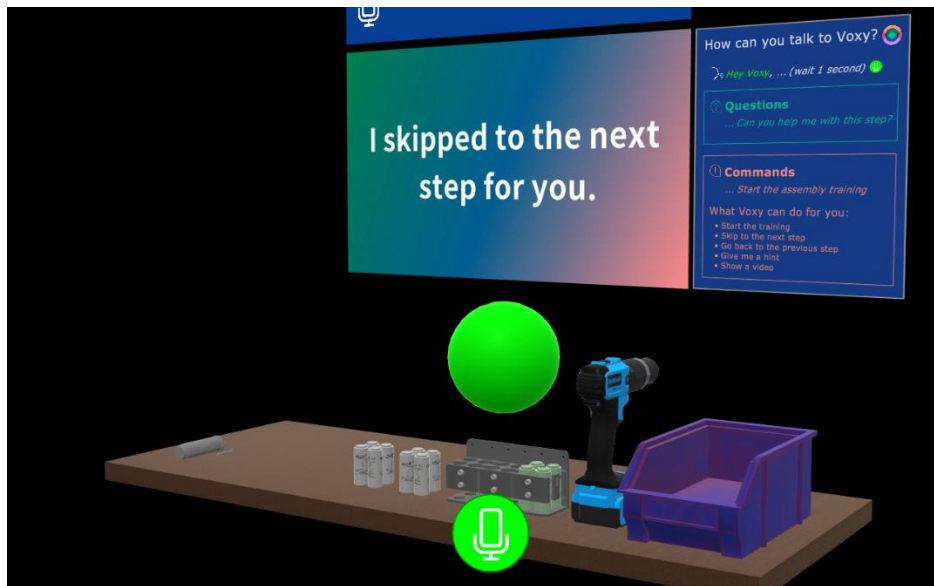


Figure 98. The display panel with feedback on action performed by Voxy and the listening feedback with green mic and green Voxy avatar

Speech Output (Text-to-Speech): Responses from the assistant are not only shown on the panel but also spoken aloud using Text-to-Speech (TTS). This gives a conversational, human-like quality to the interaction and enhances social presence.

Tutorial: The application features a voice-guided tutorial to teach users how to interact with the system. It includes instructions on using the wake word (Hey Voxy), formulating different types of queries (questions or commands), getting assistance and navigating the system. The tutorial is designed to ensure users understand how to effectively interact with the voice assistant from the beginning of the training session. The tutorial involves both speech and textual inputs. Here are some examples:

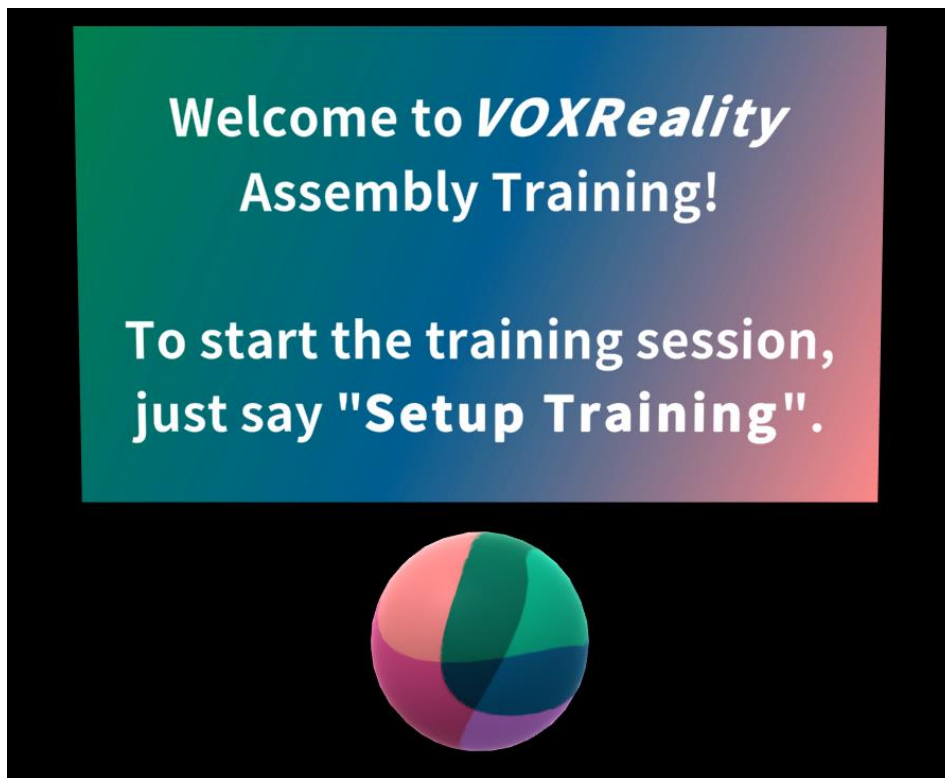


Figure 99. The tutorial with guidance for the user regarding MRTK commands



Figure 100. The tutorial with guidance regarding the cheat sheet

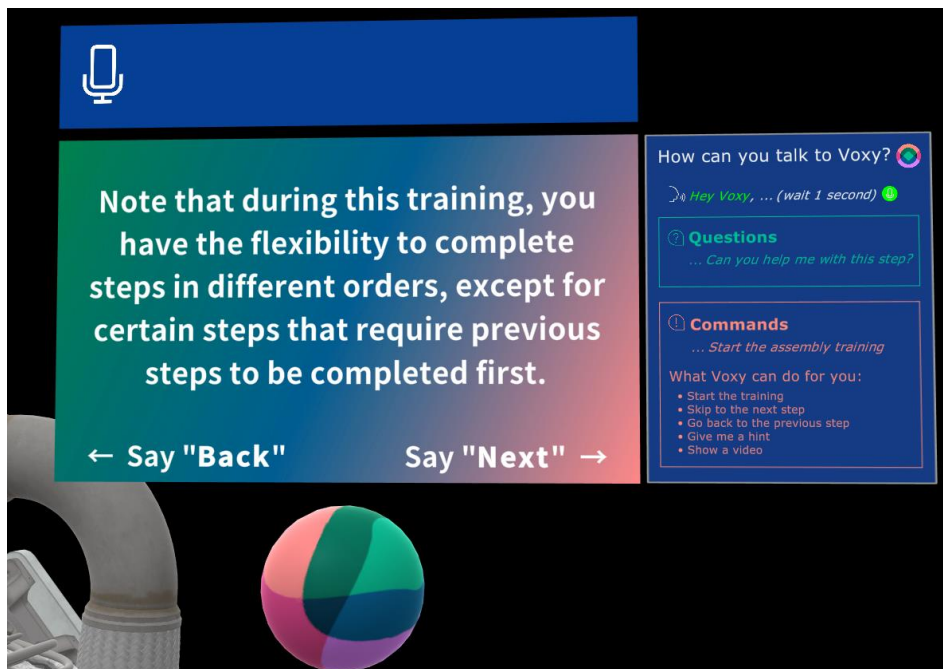


Figure 101. The tutorial with guidance regarding the application logic

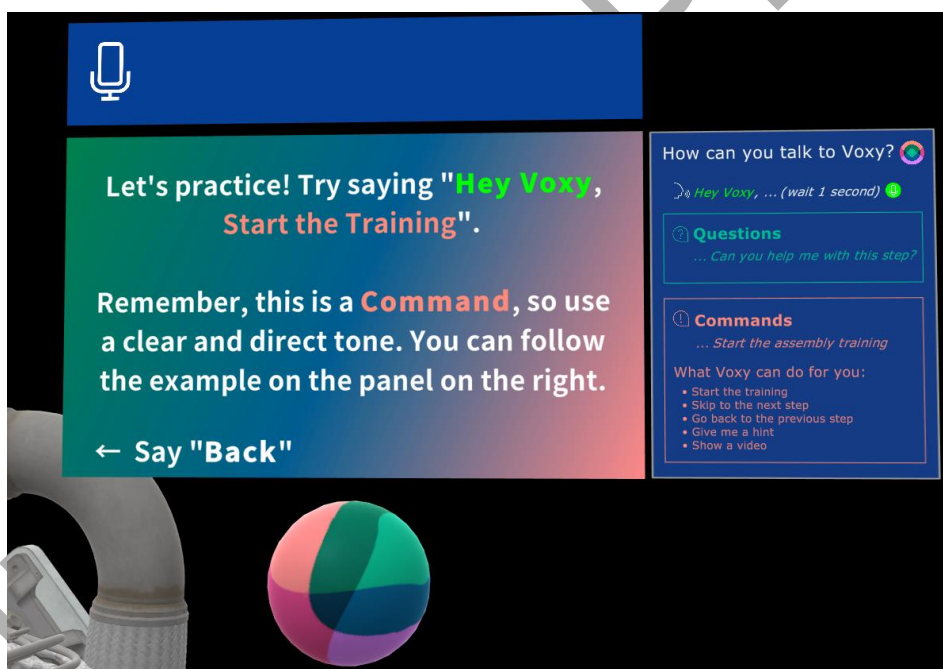


Figure 102. The tutorial with practice sessions on voice interaction

Feedback: The application performs additional actions without the user's demand. Those tasks are generally guided by some logics guided by a timer associated with user action. If an object is grabbed but misplaced or lost, the object will be replaced at its original location. The user will get a feedback regarding this action with a blue popup panel. The system will also provide feedback through the popup panel when the user demands an action that cannot be logically performed by the application.

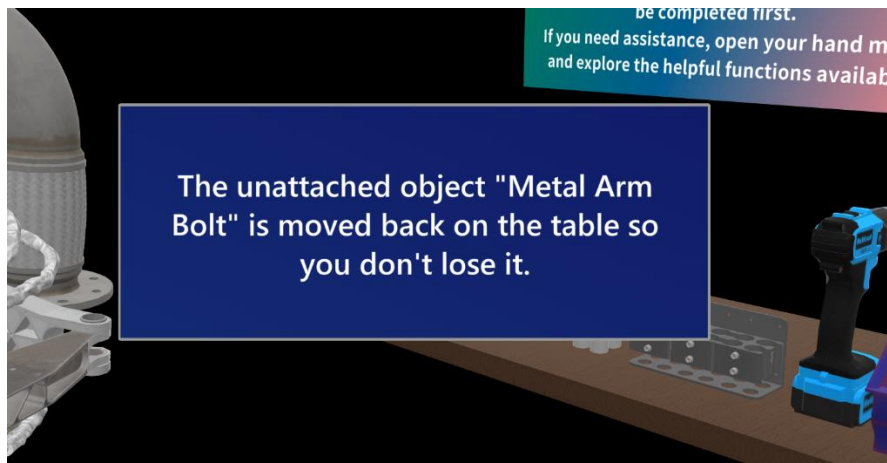


Figure 103. The blue popup panel giving user feedback regarding the replacement of the misplaced object

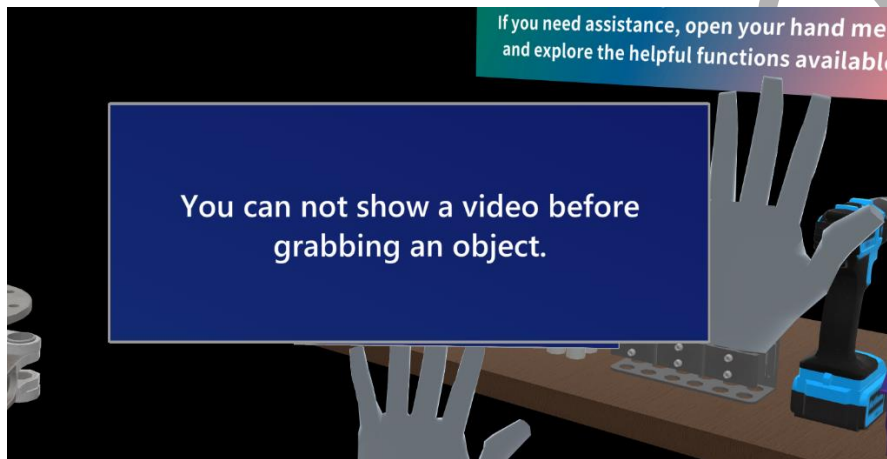


Figure 104. The blue popup panel giving user feedback regarding an action that cannot be logically performed

4.3.2.4 Summary of Achieved User Requirements

Table 11. Achievement of User Requirements

	Type	Requirements	Priority	Final Status	Reasoning
1	Objective	The goal is to provide a guided augmented reality (AR) industrial training scenario.	High	YES	
2	Setup	The external environment having a defined location is not essential.	High	YES	
3	Setup	Three levels - easy, medium, and difficult - will be offered.	High	YES	
4	Setup	A location for demo would be a standard industrial shop environment or similar lab setup.	High	NO	The AR training can be done in any location which is the highlight of the usecase. A specific location is not required.
5	Setup	An ideal location for test demo is an industrial shop or similar which approximate a canonical (standard) environment.	Medium	NO	The AR training can be done in any location which is the highlight of the usecase. A specific location is not required.
6	Assessment	The operational language is English.	High	YES	
7	Assessment	One user (trainee) at a time will be in the AR training environment.	High	YES	
8	Assessment	At least 3-5 users will be assessed in total.	High	YES	
9	Assessment	The assessment metrics may include time spent doing a virtual task, number of incorrect steps, number of times the virtual agent interceded to help - etc.	High	YES	
10	Assessment	The target user is an experienced worker with some assembly experience and knowledge of assembly training.	High	YES	
11	Assessment	The target user has no prior knowledge of the specific industrial assembly task performed in the use case.	High	YES	
12	Assessment	The target user may or may not have much experience with augmented reality.	Medium	YES	
13	Assessment	The target user will cover all possible end-user demographics, including equal numbers of genders.	Low	YES	

14	Assessment	A round of training and experiment for the use case scenarios pilot could take approximately 1-3 hours.	Medium	YES	
15	Assessment	The user's assembly training experience will be evaluated with an emphasis on the interaction with the virtual agent	High	YES	
16	Assessment	The user feedback will be collected via questionnaires, survey and/or direct verbal feedback through structured OR semi-structured interviews.	High	YES	
17	Interaction	The AR experience involves the virtual manipulation of object components in the training environment by the user.	High	YES	
18	Interaction	User interaction is with hands, without controllers and may also verbally engage with the virtual agent.	High	YES	
19	Interaction	Users should be able to choose between textual instructions OR visual cues/highlights for the training depending on the training sequence.	High	YES	
20	Interaction	User can ask (virtual agent / training assistant) for instruction to assemble a machine/object.	High	YES	
21	Interaction	Users should have options to turn on/off, speed-up or slow-down the narration.	Low	NO	It is possible to turn off the narration. The speed of the narration has been fixed and tested through user studies. Adjusting narration speed would simply be an additional UI. Additionally, the usecase is trying to provide a conversation assistant with natural speech like behaviour.
22	User Interface	The interface should provide color schemes and visual cues appropriate for the action.	High	YES	
23	User Interface	There should be a dashboard for most common functions, which users can customize and personalize for their needs.	Low	NO	A 2D dashboard would make a less immersive experience in a 3D environment.
24	User Interface	Access to help and FAQs should be available on screen at all times.	Low	YES	

25	Virtual Agent	The training session should be in both text and/or audio/video format.	Medium	YES	
26	Virtual agent	Virtual agents can provide verbal / textual explanation and display corresponding visual contents.	High	YES	
27	Virtual agent	Virtual agents will be animated in 3D avatar format.	Medium	YES	
28	Virtual agent	If/when assistance is required, virtual agents may direct the user towards objects that are necessary for the application or assembly task.	High	YES	
29	Virtual agent	There should be on-demand help during the interaction.	High	YES	
30	Virtual agent	The help should include hints and quick tips to guide the users	High	YES	
31	Virtual agent	The virtual agent should monitor the user action and aid help demonstrate the user on how to assemble a part correctly.	High	YES	
32	Virtual agent	In case of errors, the virtual agent should intuitively guide users towards the solution.	Low	NO	As the training is open ended, there are no errors that can be done by the user. When the user is stuck, the agent would assist with hints or videos.
33	Virtual agent	Support from a virtual agent may come in the form of documentation, such as images or PDFs, videos.	Medium	YES	
34	Virtual Agent	The virtual agent should intervene and offer automatic help after a certain number of incorrect steps by users.	High	YES	
35	Feedback	Users should have both visual / audible notifications regarding the correctness of the tasks at hand.	High	YES	
36	Feedback	The feedback should appear automatically in case of incorrect assembly.	High	YES	
37	Feedback	Feedback should be as close to immediate as possible and self-explanatory when requested or needed.	High	YES	
38	Feedback	The training and assembly should end with results and feedback.	High	YES	

	Type	Extended Requirements	Priority	Final Status	Priority
39	Performance	Enable microphone input transmission from HL2 client to laptop application	High	YES	

40	Performance	Extend the available interactions between the XR application and the dialogue agent	High	YES	
41	Performance	Perform internal technical testing to gather metrics for comparison in laboratory conditions to compare with the metrics gathered in pilot 2	High	YES	
42	User Interface	Improve the UI to avoid overlapping information with the virtual training scene	High	YES	
43	User Interface	User's should be provided with either visual or audible feedback before agent initiates a task	High	YES	
44	User Interface	Provide visual/audible indication when the virtual agent is listening	Medium	YES	
45	Virtual Agent	Place virtual agent at a stable, defined location in the AR environment	Medium	YES	
46	Virtual Agent	Virtual Agent initiates help that is beneficial to the user	Medium	YES	

Several requirements outlined for the system were successfully implemented; however, a few were intentionally not addressed, as shown in the tables above. The setup-related requirements (IDs 4 and 5) were deprioritized because the training is designed to be flexible and can be effectively conducted in various environments, making a specific industrial setup unnecessary. Regarding interaction (ID 21), narration speed control was excluded as the interaction design was optimized with user. Nevertheless, users retain the ability to disable the dialogue agent through a voice command. Additionally, the proposed customizable dashboard (ID 23) was omitted, as its 2D interface would detract from the immersive experience essential to the system. Finally, the virtual agent's intuitive guidance in case of errors (ID 32) is not implemented as it is to avoid spoon-feeding. However, the user gets hints and video suggestions.

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 38rd 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 Virtual Reality Conference application has successfully implemented 40 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 Reality 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.

Following that, the consortium focuses on planning and execution of Pilot 2, 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 is included in the D5.2 “Pilot planning and validation V2” (due on M33 as well), while the execution details and results analysis will be presented in D5.4 “Pilot analysis & feedback V2” (due on M37). 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. That 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 are 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 2.



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] 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.
- [9] 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.
- [10] 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.
- [11] J. Larson, M. Menickely and S. M. Wild, "Derivative-free optimization methods.," *Acta Numerica*, vol. 28, pp. 287-404, 2019.
- [12] Z. Yuan, Y. Shang, Y. Song, Wu., Y. Yan and G. Sun, "Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv*," vol. 2313.05821, 2023.
- [13] X. Wang, Y. Zheng, Z. Wan and M. Zhang, "Svd-llm: Truncation-aware singular value decomposition for large language model compression," p. 2403.07378, 2024.
- [14] X. Wang, S. Alam, Z. Wan, H. Shen and M. Zhang, "SVD-LLM V2: Optimizing Singular Value Truncation for Large Language Model Compression.," *arXiv preprint arXiv*., p. 2503.12340..
- [15] Y. C. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen and H. Jin, "Language model compression with weighted low-rank factorization.," vol. *arXiv preprint arXiv*, p. 2207.00112, 2022.
- [16] VOXReality, "D3.1 - Advanced AI multi-modal for XR analysis V1," 2023.
- [17] VOXReality, "D2.3 - Development infrastructure and integration guidelines," 2023.
- [18] 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.