International Journal
Information Technology Applications (ITA)

Volume 15, Number 1, July 2026

PROPOSAL AND IMPLEMENTATION OF CONVERSATION AI SYSTEM FOR AVATARS IN VIRTUAL REALITY

Roman Zemaník

Abstract:

This article presents the design and implementation of a conversational AI system for avatars in virtual reality. The system enables spoken interaction with MetaHuman avatars in Slovak and English and is implemented in Unreal Engine 5. The work connects speech recognition, large language model response generation, speech synthesis, audio playback, avatar state feedback, and lip-sync animation. Existing local and cloud-based models are integrated into a modular STT-LLM-TTS pipeline so that different processing configurations can be compared. The evaluation focuses on response latency, hardware load, and the selection of a suitable configuration for VR use. Technical testing showed that the lowest average latency was achieved by the configuration using cloud STT, local LLM, and local TTS with an average response time of 1.94 s. For user testing, cloud STT, cloud LLM, and local TTS were selected as a better compromise between response quality and latency. The results show that LLM-based conversational AI can be integrated with VR avatars in Unreal Engine while preserving flexibility for further model and performance evaluation.

Keywords:

Virtual reality, conversational avatar, speech recognition, text-to-speech, large language models, Unreal Engine.

Introduction

Virtual reality allows the user to immerse themselves in a digital environment and perceive it as a space around them. Therefore, forms of interaction that resemble communication in the real world feel more natural in VR. Traditional text input through a virtual keyboard or selection from predefined dialogue options can interrupt the intended experience. One way to bring VR interaction closer to natural communication is through conversational avatars capable of responding to the user's free spoken input.

The implementation of such conversational avatars can be based on large language models, which can process freely formulated user input and generate responses in natural language. However, the language model alone is not sufficient for creating spoken interaction in virtual reality. It must be connected with speech recognition, speech synthesis, and the visual representation of the character so that the user can conduct a conversation directly in the virtual environment. Important factors are therefore not only the correctness of the answers, but also the response speed of the system and the visual presentation.

The aim of this work is to design and implement a prototype of a conversational avatar in virtual reality. The system is built in Unreal Engine 5, uses MetaHuman to create a realistic avatar, and supports voice communication in Slovak and English. The conversational pipeline is divided into separate parts for speech recognition, text response generation, and speech synthesis. The implementation also allows local and cloud-based solutions to be combined so that different combinations of these components can be tested.

The work first focuses on an overview of existing approaches to creating conversational avatars in virtual reality. Then the system design, its requirements, architecture, and the selection of used technologies are described. The implementation chapter describes how the system captures the user's voice input, processes it in the conversational pipeline, and presents the response through the avatar's voice and visual behavior. The final chapters focus on technical and user testing of the implemented system and possibilities for further extension.

1 Overview

Conversational agents developed from rule-based text systems to embodied agents in virtual reality. ELIZA used keyword matching and predefined response patterns without semantic understanding [1]. Later systems introduced more advanced rule-based methods and early natural language processing. STEVE was one of the early embodied pedagogical agents used in a virtual environment [2]. Current conversational-agent research describes a shift toward generative systems based on large language models (LLM), which can produce responses from conversation context [3].

Simulating human interaction with avatars requires several connected mechanisms. The system must interpret user input, generate responses, process speech in real time through STT (Speech-To-Text), LLM, and TTS (Text-To-Speech), and preserve at least short-term conversational context [4], [5]. Embodiment, personality modeling, facial animation, lip-sync, and state feedback influence how natural and readable the avatar feels [6]. Spatial context is also important, because questions in VR can depend on objects, gaze, pointing, and timing [7].

Applications with LLM avatars usually combine a VR engine, a language model, and a speech pipeline. The engine provides the virtual environment, avatar representation, and interaction logic. The language model generates the avatar's answer, while the speech pipeline converts the user's voice to text and the generated answer back to speech. In several related systems, the speech output is also connected with lip-sync or facial animation so that the answer appears to come from the virtual character [4], [8], [9].

LLM avatars can be used in several types of VR applications. In games, they can extend scripted NPC dialogue and make virtual characters react to the current scene [7]. In education and professional training, they can guide the user through tasks or act as simulated conversation partners [9]. Other examples include productivity assistants that use workspace context, social VR agents that maintain longer conversations, and virtual characters used in public events or self-talk scenarios. These use cases differ in purpose, but they rely on the same basic idea: spoken interaction with an embodied character.

Although LLM avatars enable more natural and flexible interaction than traditional systems, their use in VR introduces several limitations. Slow responses reduce the fluency of conversation and can frustrate users [10]. Cloud services can improve quality and simplify integration, but they introduce internet dependence, API costs, and external processing. Local models reduce these dependencies, but they increase hardware requirements. Other limitations include restricted context and memory, hallucinated or factually incorrect answers, weaker control over the avatar's role-specific behavior, and the need to connect LLM, STT, TTS, avatar animation, and VR interaction manually [4], [5].

The quality of an LLM avatar in VR cannot be evaluated only by whether the model generates linguistically correct answers. In related work, evaluation is usually divided into technical parameters of the system and the user experience of interaction. Technical evaluation focuses mainly on response time, stability, consistency, and the quality of generated answers. User-oriented evaluation focuses on naturalness of interaction, social presence, immersion and overall user satisfaction [6], [10].

2 Design of the Conversational Avatar

The system requirements follow the goal of voice-based interaction with an avatar in virtual reality. The user should communicate through a microphone without text input or predefined dialogue options. The system must support Slovak and English, produce answers in the corresponding language, keep response latency low enough for interactive use, and measure the duration of individual processing steps. The generated answer should be presented through voice, avatar visual feedback, and lip-sync, not only as text.

One interaction starts when the user asks the avatar a question by voice (Fig.1). The application records the input, sends it to the conversational system, and the avatar switches from idle to thinking while the request is processed. The conversational system contains three separated modules: STT converts speech to text, LLM generates the answer, and TTS converts the answer back to speech. After synthesis, the avatar plays the generated audio, switches to the talking state, applies lip-sync, and returns to idle after playback. This architecture separates conversational logic from avatar presentation and allows local and cloud STT, LLM, and TTS components to be compared without changing the interaction flow.

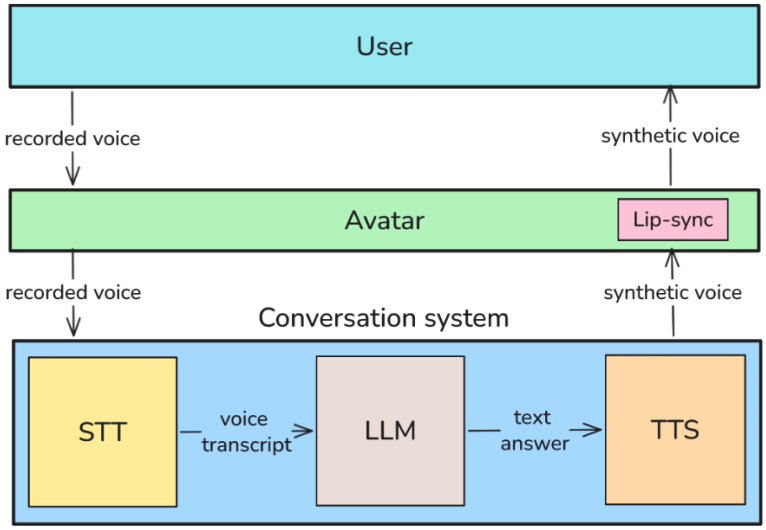

Fig.1. Conversation system.

After defining the expected behavior and architecture of the system, it was necessary to select the technologies used for implementation. The selection considered compatibility, latency, output quality, ease of integration, and practical usability for creating a functional prototype. Unreal Engine 5.7 was selected because it supports OpenXR for VR development and provides a prepared VR template. MetaHuman was selected for the visual representation of the avatar because it allows the creation of a realistic human character without modelling it from the beginning. Meta Quest 3 was selected as the VR headset because it provides a built-in microphone, stereoscopic display, headset and controller tracking, hand tracking, and compatibility with Unreal Engine 5 through OpenXR.

For the conversational pipeline, the selected cloud models were gpt-4o-mini-transcribe for speech recognition, gpt-4.1-nano for response generation, and gpt-4o-mini-tts for speech synthesis. The selected local speech recognition model was faster-whisper-small, because it provided the best compromise between latency and transcription accuracy.

Local speech synthesis was implemented with Piper using sk_SK-lili-medium for Slovak and en_US-norman-medium for English. For local response generation, qwen3-4b-2507 Q8_0 was used because it was compatible with the selected Unreal Engine plugin.

3 Implementation of the System

The first implementation step was creating the Unreal Engine project from the VR template. The template provided the basic VR player pawn, teleport movement, controller input, and settings needed to run the application in VR mode. This made it possible to focus on the conversational part of the system and avatar integration instead of implementing basic VR controls from the beginning. The project was developed and tested in the Unreal Engine editor using VR Preview. External plugins were added for cloud model integration, local language model inference, and lip-sync processing.

MetaHuman Creator was used to prepare the avatars that represent the conversational characters in the VR scene. Instead of creating characters manually from the beginning, prepared MetaHuman presets were used and adjusted for the needs of the prototype. The avatars were exported in the UE Optimized quality setting, because the system is intended for an interactive VR application where performance is important. After the characters were created, animation assets for the avatar states were prepared and retargeted to the MetaHuman skeleton. The avatars were then placed into the virtual scene, creating the visual basis for the conversational system.

Several Unreal Engine plugins and modules were added to support the conversational pipeline. The UnrealOpenAIPlugin [11] plugin was used for communication with cloud STT, LLM, and TTS models, Llama-Unreal [12] was used for local language model inference, and OVRLipSync [13] was used to generate visemes for lip-sync animation. Cloud processing was prepared through the OpenAI Platform, where an API key was created for access to the selected cloud model. Local speech recognition and local speech synthesis were implemented in a separate FastAPI Python server instead of directly inside Unreal Engine. The Unreal Engine application communicates with this server through HTTP requests.

The main implemented components are (Fig.2) BP_XRPawn, ConversationalAvatarComponent, ConversationSubsystem, STTService, LLMService, and TTSService. BP_XRPawn records the user's voice and creates an audio file. ConversationalAvatarComponent receives the file from the active avatar and forwards it to ConversationSubsystem. ConversationSubsystem then calls STTService, LLMService, and TTSService in sequence. The generated WAV response is returned to ConversationalAvatarComponent, which plays it through the avatar audio component and uses it for lip-sync and animation state changes.

Voice input is captured in BP_XRPawn, which represents the player in the VR scene. An Audio Capture component and recording submix are used to record the microphone signal. Recording can be started manually with the A button on the Quest 3 controller or automatically in Open Mic mode, where recording starts after the microphone volume exceeds a threshold and stops after a period of silence. After recording ends, BP_XRPawn exports the input as a WAV file and passes its path to the currently active ConversationalAvatarComponent. This component is attached to each conversational MetaHuman avatar and stores avatar-specific settings such as active state, name, language, gender, and local or cloud processing options. Before forwarding the recording, the component checks that the WAV file has already been fully written to disk. It then switches the avatar to the thinking state and sends the recording to ConversationSubsystem through GenerateAudioResponseFromAudio().

The conversation flow is controlled by ConversationSubsystem, implemented as a GameInstanceSubsystem available during the whole runtime of the application. During initialization, it creates STTService, LLMService, and TTSService and later coordinates their execution instead of performing the processing directly.

Before each request, it prepares the active configuration by combining global settings from ConversationSettings with avatar-specific values such as language, name, gender, and selected local or cloud processing options. The configuration is divided into STT, LLM, and TTS settings, so each service receives only the values needed for its part of the pipeline. Processing starts in GenerateAudioResponseFromAudio(), where the subsystem receives the recorded WAV file and the active avatar, prepares the configuration, and sends the recording to STTService as the first step of the pipeline.

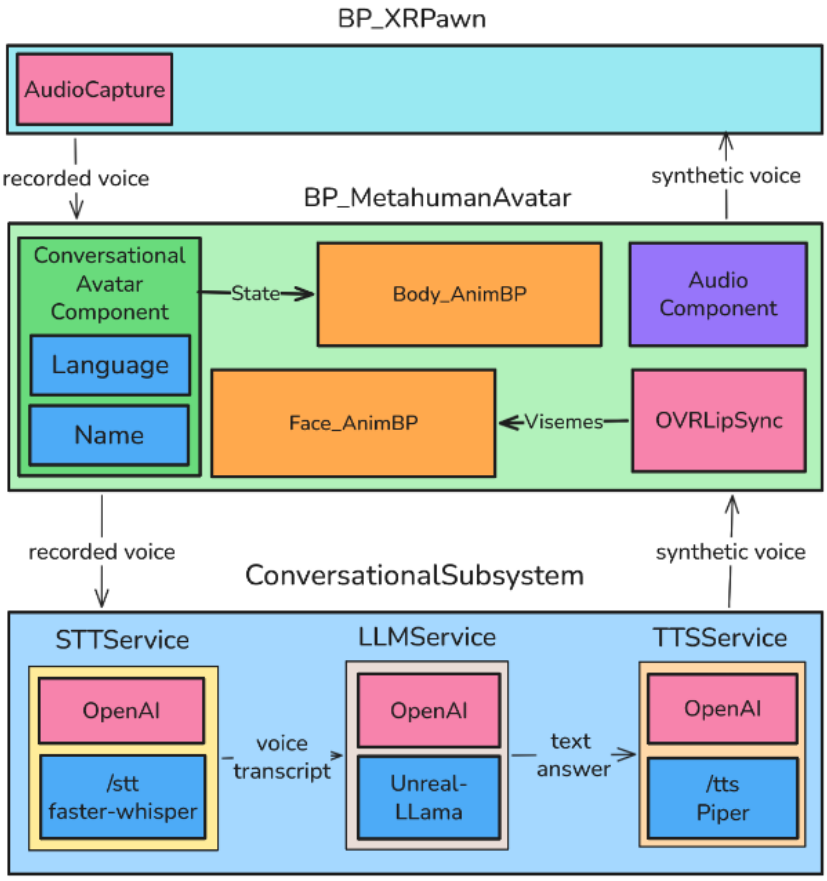

Fig.2. The main implemented components.

The three processing steps are implemented in STTService, LLMService, and TTSService. STTService receives the recorded WAV file, checks that it is available, and converts the user's speech to text using either the local FastAPI server with faster-whisper-small or the cloud model gpt-4o-mini-transcribe through UnrealOpenAIPlugin. The transcript is then passed to LLMService, which generates the avatar's text response either locally through Llama-Unreal with a Qwen model or in the cloud using gpt-4.1-nano. The active system prompt and avatar-specific values such as name, gender, and language are used during generation. The generated text is then passed to TTSService, which creates the spoken response either locally through the FastAPI server with Piper or in the cloud using gpt-4o-mini-tts. The result is saved as a WAV file and returned to ConversationSubsystem, which sends it back to the avatar component for playback.

After the WAV response is generated, ConversationalAvatarComponent presents it in the VR scene. The HandleGeneratedAudioResponse() method ends the thinking state, plays the WAV file through the avatar audio component, and switches the avatar to the talking state during playback. After the audio finishes, the avatar returns to idle. The same WAV file is also used to prepare lip-sync data. Body animation is controlled by Body_AnimBP, which switches between idle, thinking, and talking states and turns the active avatar's head toward the user (Fig.3). Face animation is controlled by Face_AnimBP. The audio is split into 10 ms segments and processed by OVRLipSync, which produces viseme values for each moment of speech. Because MetaHuman does not use OVR visemes directly, the values are mapped to MetaHuman face curves such as jaw opening and lip movement. These curves are applied during playback, while Face_AnimBP also adds simple randomized blinking so that the avatar does not appear static.

Fig.3. Avatar.

A simple VR menu was extended from the existing template menu to support avatar selection and voice input mode switching. The user can select the active avatar, which updates bIsActiveAvatar on the available ConversationalAvatarComponent instances and changes CurrentTargetAvatar in BP_XRPawn. The menu also allows switching between manual recording and Open Mic mode by changing bOpenMic and reinitializing voice recording. ConversationSubsystem also logs each request into a TSV file for later evaluation. The log contains the selected configuration, avatar, processing times for individual pipeline steps, total duration, transcript, generated response, status, and possible errors, which makes it possible to compare local and cloud processing without manual timing.

4 Evaluation

The technical evaluation focused on comparing the latency of different STT, LLM, and TTS combinations and selecting a configuration suitable for user testing. Testing was performed on a computer with an AMD Ryzen 7 7700X CPU, 32 GB RAM, AMD Radeon RX 9070 XT GPU with 16 GB VRAM and Windows 11 Pro. Hardware load during VR runtime was measured with MSI Afterburner in three scenarios: the base VR scene without the conversational pipeline, a cloud-based pipeline, and a fully local pipeline. The base scene and cloud pipeline kept stable FPS, while the fully local pipeline caused visible FPS drops when local models started processing the input. GPU load remained high in all scenarios, which indicates that the VR scene itself was already demanding, while CPU load showed clearer differences because the local pipeline produced short peaks correlated with FPS drops.

The latency evaluation (Fig.4) compared all eight combinations of local and cloud processing for the three pipeline components: STT, LLM, and TTS. The notation uses C for cloud and L for local processing, in the order STT-LLM-TTS. For each request, the system recorded STT time, LLM time, TTS time, and total pipeline time from sending the recorded audio to receiving the synthesized response. The lowest average latency was achieved by C-L-L with a total time of 1.94 s. A similar result was achieved by C-C-L with 2.12 s, while the highest latency was measured for L-L-C with 5.80 s. The results show that cloud TTS increased the total processing time the most, while the difference between local and cloud LLM processing was smaller than the differences caused by STT and TTS choices.

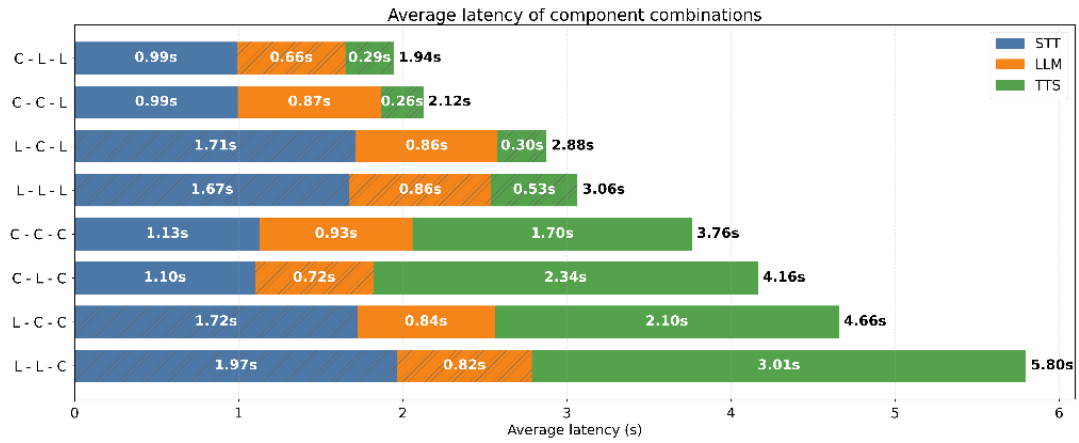

Fig.4. Average latency.

User testing was conducted after technical testing and after selecting the final system configuration. The selected configuration was C-C-L, which used cloud STT, a cloud language model, and local TTS. Each participant first received a short explanation of the test and basic VR controls. During the test scenario, the participant had to ask the avatar at least 10 questions, including at least one follow-up question related to a previous answer. This was used to test both separate responses and the ability to maintain context in a short conversation. After the interaction, each participant filled in a questionnaire based on a 5-point Likert scale, where 1 meant strongly "disagree" and 5 meant "strongly agree". The questionnaire contained 13 questions focused on ease of communication, quality and naturalness of answers, voice output, fluency of interaction, perceived latency, and the overall impression of the conversation in VR.

User testing showed a generally positive response to the system. Participants rated (Fig.5) the interaction as engaging, considered the response time acceptable, and indicated that a similar form of communication could be useful in another VR application. Lower ratings were connected mainly with the naturalness of the answers, the voice output, and lip-sync. Since users were mostly satisfied with the speed of the system but less satisfied with the voice presentation, one possible improvement is to replace the local TTS model with a higher-quality cloud TTS model. This would probably increase latency, but it could improve the naturalness and persuasiveness of the avatar's spoken output.

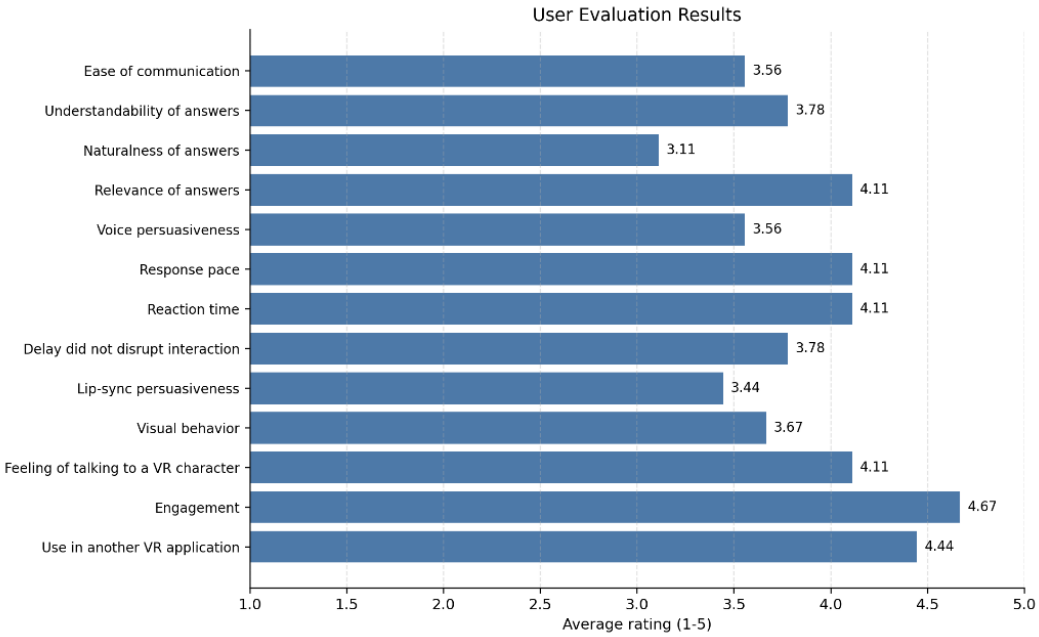

Fig.5. Average rating.

Conclusion

The aim of this work was to design and implement a prototype of a conversational avatar in virtual reality that supports spoken communication in Slovak and English. The system was implemented in Unreal Engine 5 with MetaHuman avatars and a modular conversational pipeline for speech recognition, response generation, and speech synthesis.

The implementation separates the VR player, avatar component, conversation subsystem, and individual STT, LLM, and TTS services. The system supports both local and cloud-based processing, which makes it possible to compare different configurations. Local STT and TTS are provided through a separate FastAPI server, while cloud processing is handled through OpenAI models. The avatar also provides visual feedback through idle, thinking, and talking states, head direction toward the user, blinking, and lip-sync animation.

Technical testing showed clear differences between local and cloud configurations. The lowest average latency was achieved by the configuration using cloud STT, local LLM, and local TTS, but this setup did not provide sufficient response quality in Slovak. Fully local processing also caused higher hardware load during VR use. Therefore, the configuration with cloud STT, cloud LLM, and local TTS was selected for user testing as a better compromise between response quality and latency. User testing showed that participants generally accepted the interaction positively, especially its

engagement and response speed, while the voice output quality and lip-sync remained the main areas for improvement. The resulting prototype fulfills the main goal of the work, because it allows the user to communicate with a VR avatar by voice in both supported languages. Further development could focus on improving the naturalness of the avatar's voice and lip-sync, extending avatar animations, adding spatial context from the virtual environment, or turning the solution into a reusable Unreal Engine plugin.

Acknowledgement

I would like to thank RNDr. Ján Lacko, PhD., for his professional assistance, valuable advice, and consultations during the preparation of this master thesis.

References

[1] Weizenbaum, J. (1966). ELIZA: A computer program for the study of natural language communication between man and machine. Communications of the ACM. vol. 9, no. 1, pp. 36-45, doi: 10.1145/365153.365168.

[2] Johnson, W. L., Rickel, J. (1997). STEVE: An animated pedagogical agent for procedural training in virtual environments. ACM SIGART Bulletin. vol. 8, no. 1-4, pp. 16-21.

[3] Schöbel, S., Schmitt, A., Benner, D., Saqr, M., Janson, A., Leimeister, J. M. (2024). Charting the Evolution and Future of Conversational Agents: A Research Agenda Along Five Waves and New Frontiers. Information Systems Frontiers.

[4] Buldu, K. B., Özdel, S., Lau, K. H. C., Wang, M., Saad, D., Schönborn, S., Boch, A., Kasneci, E., Bozkir, E. (2025). CUIfy the XR: An Open-Source Package to Embed LLM-Powered Conversational Agents in XR. IEEE AIxVR, pp. 192-197, doi: 10.1109/AIxVR63409.2025.00037.

[5] Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, doi: 10.1145/3586183.3606763.

[6] Sonlu, S., Bendiksen, B., Durupinar, F., Güdükbay, U. (2025). Effects of Embodiment and Personality in LLM-Based Conversational Agents. IEEE Conference Virtual Reality and 3D User Interfaces, pp. 718-728, doi: 10.1109/VR59515.2025.00094.

[7] Radež, G., Bohak, C. (2024). Integrating environmental awareness into NPCs: contextual conversational interaction in games. CEUR Workshop Proceedings. vol. 3866, pp. 11-29.

[8] Wan, H., Zhang, J., Suria, A. A., Yao, B., Wang, D., Coady, Y., Prpa, M. (2024). Building LLM-based AI Agents in Social Virtual Reality. Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, doi: 10.1145/3613905.3651026.

[9] El Hajji, M., Ait Baha, T., Berka, A., Ait Nacer, H., El Aouifi, H., Es-Saady, Y. (2025). An Architecture for Intelligent Tutoring in Virtual Reality: Integrating LLMs and Multimodal Interaction for Immersive Learning. Information. vol. 16, no. 7, article 556, doi: 10.3390/info16070556.

[10] Christiansen, F. R., Hollensberg, L. N., Jensen, N. B., Julsgaard, K., Jespersen, K. N., Nikolov, I. (2024). Exploring Presence in Interactions with LLM-Driven NPCs: A Comparative Study of Speech Recognition and Dialogue Options. Proceedings of the 30th ACM Symposium on Virtual Reality Software and Technology, doi: 10.1145/3641825.3687716.

[11] life-exe (2026). UnrealOpenAIPlugin: Complete Unreal Engine plugin for the OpenAI API. Retrieved online from https://github.com/life-exe/UnrealOpenAIPlugin.

[12] getnamo (2026). Llama-Unreal: Llama.cpp plugin for Unreal Engine 5. Retrieved online from https://github.com/getnamo/Llama-Unreal.

[13] Shiyatzu (2026). OculusLipsyncPlugin-UE5: Oculus LipSync Plugin compiled for Unreal Engine 5. Retrieved online from https://github.com/Shiyatzu/OculusLipsyncPlugin-UE5.

[14] Mozilla (2024). Common Voice Dataset. Retrieved online from https://mozilladatacollective.com/datasets/cmn2e8ojy01l2mm07giwrvaqf.

[15] westbrook (2024). English Accent DataSet. Retrieved online from https://huggingface.co/datasets/westbrook/English_Accent_DataSet.

Author

Bc. Roman Zemaník
Faculty of Informatics
xzemanik@paneurouni.com
Student at the Faculty of Informatics,
currently working as a Frontend Developer.