Politecnico di Torino

Corso di Laurea Magistrale in
Ingegneria del Cinema e dei Mezzi di Comunicazione

Politecnico
di Torino

Assistente Conversazionale in Ambienti XR

Analisi e implementazione di una pipeline vocale interattiva basata su tecnologie AI

Relatore Accademico:
Prof. Andrea Bottino

Tutor Aziendale:
Matteo Barosso

Candidato:
Matteo Giuseppe Filpi
Matr. 306269

Anno Accademico 2024/2025 — Sessione di Laurea Luglio 2025

Indice

Elenco delle figure **iii**

Elenco delle tabelle **vi**

Abstract **vii**

1 Introduzione **1**

1.1 Contesto 1
1.2 Obiettivi della tesi 3
1.3 Struttura del documento 4

2 Stato dell'arte **6**

2.1 Fondamenti di XR 6
2.2 Intelligenza Artificiale: origini e sviluppi 14
2.3 Modelli di linguaggio di grandi dimensioni (LLM) 19
2.4 Intelligenza artificiale generativa 23
2.5 Chatbot e intelligenza artificiale conversazionale 26
2.6 Soluzioni esistenti e confronto 31

3 Metodologia **34**

3.1 Struttura generale del sistema 34
3.2 Gestione dello stato globale 39
3.3 Spawn dinamico e gestione multi-avatar 43
3.4 Ingegneria del prompt dinamico 48
3.5 Whisper e controllo sulla trascrizione dell'input vocale 55
3.6 Interazione con il modello LLM 71
3.7 Sintesi vocale con Amazon Polly 75
3.8 Animazione e sincronizzazione del viso e del corpo 86
3.9 Sistema di logging interno 95
3.10 Interfaccia grafica e componenti interattivi 99

4 Valutazione dei risultati e iterazioni di sviluppo **103**
4.1 Strategia di test e metodologia di validazione 105
4.2 Sviluppo incrementale e approccio iterativo 108
4.3 Prestazioni e osservazioni tecniche 111
4.4 Feedback raccolti e valutazione soggettiva 114
4.5 Considerazioni per lo sviluppo futuro 117

5 Conclusioni **123**

Bibliografia **125**

Elenco delle figure

2.1 La *Sword of Damocles*, primo visore di realtà virtuale sviluppato da Ivan Sutherland nel 1968 (Liu et al., 2018). 8
2.2 *EyePhone* e *DataGlove* sviluppati da VPL Research (Pape, 1999). . . 9
2.3 Il visore Meta Quest 3, utilizzato come dispositivo principale per l'implementazione e il testing del progetto descritto in questo elaborato (Meta Platforms, Inc., 2024). 10
2.4 Rappresentazione grafica di un modello di riconoscimento automatico delle immagini (Aphex34, 2017). 15
2.5 Schema di una rete neurale profonda (IBM, 2021). 16
2.6 Schema di una rete neurale convoluzionale (CNN) che mostra i vari strati fondamentali e il flusso di dati tra essi (ResearchGate, 2019). . 17
2.7 Struttura base di una rete neurale ricorrente (RNN) con il loop ricorrente, illustrata nella sua forma srotolata nel tempo (Goodfellow et al., 2016; Wikipedia contributors, 2024). 17
2.8 Diagramma del processo di backpropagation in una rete neurale (GeeksforGeeks, 2023). 18
2.9 Diagramma generale di una rete generativa avversaria (GAN), in cui il generatore (G) produce immagini sintetiche a partire da un input casuale, mentre il discriminatore (D) cerca di distinguere tra immagini reali e generate (Mehmood et al., 2021). 24
2.10 Screenshot di una conversazione con il chatbot ELIZA, che simula un terapeuta rogersiano attraverso un sistema di regole basato su pattern matching (Weizenbaum, 1966). 26
2.11 Immagine storica del chatbot ALICE(Onlim, 2025). 27
2.12 Esempio dell'interfaccia utente del sistema sviluppato dalla Hong Kong Metropolitan University, con visualizzazione in realtà virtuale e interazione mediata da avatar (Lee et al., 2024). 32

3.1 Esempio di creazione di un avatar con la piattaforma RPM. 35
3.2 Struttura ad albero del prefab `Chatbot` all'interno della gerarchia Unity. 38

3.3 Eventi reattivi generati da ChatbotGlobalData.cs per notificare i cambiamenti di stato agli altri moduli del sistema. 39
3.4 Schema degli eventi generati da `ChatbotGlobalData.cs` e dei moduli coinvolti nella propagazione dei cambiamenti di stato. Ogni evento notifica selettivamente i componenti interessati per mantenere la coerenza del sistema. 41
3.5 Screenshot dell'editor di Unity con una *Spawn Zone* visibile (area `Reception`), delimitata da un `BoxCollider`. L'utente deve entrarvi per poter attivare il chatbot. 44
3.6 Screenshot dell'Inspector di Unity relativo al componente `SpawnManager`. Sono visibili il riferimento alla posizione del giocatore e i prefab degli avatar chatbot multilingua. 46
3.7 Interfaccia Inspector dello script `PromptInput.cs`, collegato a ciascuna `SpawnZone`. I campi configurabili permettono di definire restrizioni tematiche, informazioni contestuali e correzioni di trascrizione per il prompt dinamico. 49
3.8 Screenshot del metodo `GetPrompt()` nello script PromptEngineering.cs. La funzione costruisce il prompt dinamico in base al modello LLM attivo, selezionando automaticamente uno tra i template disponibili per adattarsi alle caratteristiche del modello. 51
3.9 Porzione del prompt costruito in `PromptEngineering.cs`, dedicata all'inserimento delle correzioni note (`wrongTranslations`) per errori di trascrizione vocale. 53
3.10 Implementazione del metodo `TrimSilence()` nello script `SaveWav.cs`. 60
3.11 Implementazione della funzione `IsSilence()`, utilizzata per il rilevamento in tempo reale dei segmenti di silenzio basata sul valore RMS. 61
3.12 Funzione `FilterPromptFromTranscription()` nel componente `Whisper DirectText.cs`. 63
3.13 Funzione `CleanTranscriptionFromEdgeEmptyPhrases()` nel componente `TranscriptEmptyPhrases.cs`. 65
3.14 Logica di fallback e gestione dell'inattività nel componente `Whisper DirectText.cs`. Se la trascrizione risulta vuota per più volte consecutive, viene pronunciato un messaggio di chiusura e l'interazione viene terminata automaticamente. 67
3.15 Gestione degli eventi interni nel componente `WhisperDirectText.cs`. 69

3.16 Esempio di aggiornamento dinamico del messaggio `system` nella cronologia. Se già presente, viene sovrascritto; altrimenti, viene inserito in testa alla lista. Segue l'aggiunta del messaggio dell'utente. 72
3.17 Strutture dati per associare animator e sorgenti audio a ciascuna lingua, con supporto all'inserimento tramite Inspector. 78
3.18 Assegnazione manuale degli animatori e delle sorgenti audio per ciascuna lingua nel componente `TextToSpeech`, tramite l'Inspector di Unity. 79
3.19 Esempio di corpo della richiesta inviata al servizio di sintesi vocale Amazon Polly, con evidenza dei principali parametri configurabili come testo, voce, formato audio e modalità di sintesi. 81
3.20 Codice per la gestione asincrona del flusso audio di Amazon Polly, dalla conversione in file MP3 al caricamento come AudioClip in Unity. 82
3.21 Uno degli avatar 3D generati con la piattaforma Ready Player Me, integrato nell'ambiente virtuale del museo aziendale. 86
3.22 Vista dell'inspector del prefab `ChatbotVisual-ITALIAN` con i relativi componenti. 88
3.23 Struttura del prefab `ChatbotVisual-ITALIAN` all'interno della *Hierarchy* di Unity. 89
3.24 Vista dei quattro avatar chatbot multilingua all'interno dell'ambiente virtuale del museo. Da sinistra verso destra sono rappresentate le versioni in italiano, francese, spagnolo e inglese, ciascuna associata a un proprio prefab e sistema vocale indipendente. 90
3.25 Inspector di Unity relativo allo *Skinned Mesh Renderer* dell'*head*, con la lista dei blend shape configurati e utilizzabili nello script `OVRLipSyncContextMorphTargetTest`. 92
3.26 Schema dell'Animator Controller con gli stati `Idle` e `Talking` e i trigger per le animazioni corporee. 93
3.27 Screenshot dell'inspector di Unity relativo allo script `EyeAnimation Handler`, con i parametri configurabili per la gestione del battito delle palpebre e micro-movimenti oculari. 94
3.28 Esempio di interazione registrata nel `ChatbotLog` della reception, con una domanda dell'utente e la risposta generata dal sistema. Il pannello mostra la formattazione testuale configurata dinamicamente, inclusi colore, stile e timestamp. 96
3.29 Lo script `ButtonsManager.cs` visualizzato nell'Inspector di Unity. . . 99

3.30 Schema della disposizione dei bottoni all'interno dell'interfaccia del museo virtuale. 100
3.31 Inspector del bottone associato alla lingua `Italian`, in cui è visibile il componente `Interactable Unity Event Wrapper`[14]. Questo consente di collegare l'evento all'invocazione del metodo `UpdateLanguage` nello script `ButtonsManager`, passando come parametro la stringa `"Italian"`. 101

4.1 Chatbot in azione durante una sessione di interazione vocale nell'applicazione standalone su dispositivo *Meta Quest*. 103
4.2 Esempio di interazione con il chatbot tramite pulsanti grafici. 106
4.3 Esempio di animazione del chatbot durante la conversazione, con sincronizzazione del movimento di bocca e braccia in risposta al parlato. 113

Elenco delle tabelle

3.1 Esempi di messaggi localizzati predefiniti in base alla lingua attiva . . 42
3.2 Esempi di errori fonetici comuni rilevati nella trascrizione di "Akkodis" e corretti tramite sostituzioni personalizzate nello script `PromptInput.cs`. 54
3.3 Parametri di configurazione e variabili interne relative alla registrazione e alla gestione del silenzio. 57
3.4 Valori soglia consigliati per la funzione `TrimSilence()` in base al livello di rumore ambientale. 59
3.5 Parametri di timeout per la trascrizione vocale. 62
3.6 Esempi di frasi irrilevanti mappate manualmente nello script `Transcript EmptyPhrases.cs`. 66
3.7 Modelli supportati e template di prompt associati 73
3.8 Mappatura tra lingua, voce Polly e nome visualizzato del chatbot . . 77
3.9 Mappatura tra visemi Oculus, blend shapes dei modelli Ready Player Me e loro funzione nel sistema di animazione. 91
3.10 Parametri configurabili tramite *Inspector* nello script `ChatbotLog.cs`. 97

Abstract

Il presente lavoro riguarda lo sviluppo di un assistente conversazionale per ambienti di realtà estesa (XR), progettato per gestire interrogazioni vocali in scenari immersivi. Il prototipo è stato sperimentato all'interno di un museo virtuale aziendale, ma la sua architettura modulare ne consente l'adattamento a diversi contesti XR con opportune personalizzazioni.

L'obiettivo è stato realizzare un chatbot in grado di rispondere a domande pertinenti a un dominio tematico ben definito, nel rispetto dei vincoli progettuali e aziendali, e coerente con il ruolo e la posizione assegnati all'interno dell'esperienza immersiva. Particolare attenzione è stata posta sulla riduzione della latenza tra domanda e risposta, al fine di mantenere un flusso comunicativo fluido e naturale.

Il sistema integra il modello di linguaggio di OpenAI (LLM) per l'elaborazione semantica, Whisper per la trascrizione speech-to-text e Amazon Polly per la sintesi vocale text-to-speech, orchestrati da una pipeline che gestisce l'intero ciclo di interazione: acquisizione della domanda, elaborazione del contenuto e riproduzione della risposta.

L'approccio metodologico ha previsto anche l'integrazione di un avatar 3D con animazioni di idle, di interazione vocale e sincronizzazione labiale, nonché la gestione della lingua tramite un selettore a pulsante separato, che permette di attivare un avatar e un accento specifici per ciascuna lingua. Inoltre, la pipeline è stata ottimizzata attraverso tecniche di prompt engineering per migliorare la pertinenza e la precisione delle risposte.

I risultati evidenziano un prototipo funzionante, capace di rispondere in tempo quasi reale: sebbene sia presente un breve intervallo tra input e output, il sistema mantiene una reattività adeguata a garantire l'immersione. La sfida principale ha riguardato l'ottimizzazione del bilanciamento tra velocità di risposta e accuratezza semantica.

In conclusione, l'architettura modulare del sistema supporta la scalabilità verso altri progetti XR. I futuri sviluppi prevedono un perfezionamento delle animazioni, una maggiore ottimizzazione del codice e l'esplorazione di tecnologie alternative per la trascrizione e la sintesi, nonché l'integrazione di modelli AI aggiuntivi per potenziare funzionalità e prestazioni.

1. Introduzione

1.1 Contesto

Negli ultimi anni, le tecnologie immersive hanno conosciuto una rapida evoluzione e una crescente diffusione anche in ambiti che vanno ben oltre l'intrattenimento, affermandosi come strumenti strategici in numerosi contesti professionali e industriali. Con il termine *realtà estesa* (*Extended Reality*, XR) si fa riferimento a un insieme di tecnologie che comprendono la realtà virtuale (VR), la realtà aumentata (AR) e la realtà mista (MR), ciascuna caratterizzata da differenti livelli di immersività e modalità di interazione con l'ambiente circostante. Tale classificazione funge da termine ombrello per descrivere tutte le forme di integrazione tra mondo reale e contenuti digitali tridimensionali.

Il tratto distintivo di queste esperienze interattive è la capacità di generare ambienti simulati o arricchiti, in cui l'utente può sperimentare una sensazione di presenza e agire con modalità naturali e multisensoriali. Questo livello di immersività consente di superare la semplice visualizzazione passiva di contenuti, favorendo un coinvolgimento attivo, cognitivo ed emozionale. L'esperienza utente risulta così ampliata e trasformata, rendendo le informazioni non solo visibili, ma tangibili e navigabili, spesso mediante un'interazione diretta basata sul movimento corporeo, il tracciamento dello sguardo o comandi vocali.

Se inizialmente l'impiego della realtà estesa era circoscritto principalmente al settore videoludico e dell'intrattenimento, oggi queste soluzioni immersive trovano ampio impiego in ambiti come l'educazione, la formazione professionale, l'industria manifatturiera, la medicina e la comunicazione culturale. In questi contesti, la capacità di ricreare scenari complessi, visualizzare processi non osservabili direttamente, o semplicemente catturare l'attenzione dell'utente in modo efficace, rende l'XR uno strumento estremamente versatile e potente. L'interattività e la personalizzazione dell'esperienza, infatti, aumentano la capacità di apprendimento, migliorano la memorizzazione dei contenuti e rafforzano l'engagement complessivo.

Fattori recenti come la crescente accessibilità delle tecnologie hardware (visori, controller, sistemi di tracking), l'avanzamento dei motori grafici real-time e l'inte-

resse verso il cosiddetto *metaverso*[1] hanno contribuito ad accelerare l'adozione delle soluzioni immersive anche da parte di aziende, istituzioni e pubbliche amministrazioni.

Parallelamente, l'evoluzione dell'intelligenza artificiale, in particolare dei modelli linguistici di grandi dimensioni (Large Language Models, LLM), ha profondamente trasformato le modalità di interazione tra esseri umani e tecnologie digitali. Questi modelli, addestrati su enormi quantità di dati testuali, sono in grado di interpretare il linguaggio naturale con crescente precisione, generando risposte pertinenti e contestualmente adeguate a partire da input non strutturati. Tale progresso ha aperto nuove possibilità per lo sviluppo di sistemi conversazionali avanzati, capaci di sostenere conversazioni complesse e adattabili, ben oltre la semplice esecuzione di comandi.

L'integrazione di queste componenti intelligenti in ambienti immersivi come la realtà estesa (XR) permette di creare interfacce vocali sofisticate, dove l'interazione con contenuti digitali avviene in modo fluido, naturale e adattivo. In questo contesto, gli assistenti conversazionali virtuali non solo arricchiscono l'esperienza immersiva, ma svolgono un ruolo attivo nel guidare l'utente attraverso ambienti tridimensionali, fornendo supporto immediato, rispondendo a domande e assistendo nell'esecuzione di attività complesse in tempo reale.

Il progetto alla base di questa tesi è stato inizialmente concepito per essere integrato all'interno del museo virtuale sviluppato da Akkodis, con l'obiettivo di arricchire l'esperienza immersiva mediante un assistente conversazionale vocale. Tuttavia, l'obiettivo non è limitato alla realizzazione di una soluzione ad hoc per il museo, bensì alla progettazione e sperimentazione di un sistema modulare e riutilizzabile, capace di adattarsi, con le giuste modifiche e ottimizzazioni, a differenti scenari XR futuri, anche al di fuori del dominio aziendale specifico.

[1]Il termine *metaverso* indica un insieme di ambienti virtuali tridimensionali e persistenti, in cui gli utenti possono interagire tra loro e con oggetti digitali in tempo reale, spesso rappresentati da avatar.

1.2 Obiettivi della tesi

Questo elaborato si propone di descrivere nel dettaglio la progettazione e lo sviluppo di un sistema conversazionale vocale pensato per ambienti XR, con particolare attenzione all'integrazione tra i vari componenti software e alla gestione della latenza tra input vocale e output dell'assistente.

L'obiettivo principale è stato realizzare un prototipo funzionante di assistente vocale in grado di:

- riconoscere domande vocali in diverse lingue attraverso un sistema di *speech-to-text*;
- interpretare semanticamente le richieste dell'utente tramite un LLM;
- fornire risposte vocali tramite un sistema di sintesi text-to-speech, con voce e accento adeguati alla lingua selezionata;
- animare un avatar 3D che esegua movimenti coordinati, come il *lip-sync*, espressioni facciali e gestualità di conversazione;
- garantire una comunicazione fluida e credibile attraverso un tempo di risposta ridotto;
- operare in modo modulare, per poter essere facilmente adattato a scenari futuri al di fuori del museo virtuale;
- sviluppare un sistema di prompt engineering per definire e ottimizzare le risposte dell'assistente in base a limitazioni aziendali, di progetto e del contesto applicativo in cui è stato inserito.

Inoltre, particolare attenzione è stata data alla coerenza comportamentale dell'assistente, in relazione al suo ruolo istituzionale e alle specifiche esigenze di esperienza utente in ambienti tridimensionali. Questi aspetti hanno rappresentato una sfida cruciale nell'integrazione tra AI e ambienti immersivi, dove la qualità dell'interazione e la percezione del "realismo" del sistema sono fondamentali per il successo dell'applicazione.

1.3 Struttura del documento

Il contenuto dell'elaborato, oltre all'introduzione, è articolato nei seguenti capitoli:

- **Capitolo 2 – Stato dell'arte**: fornisce un inquadramento teorico e tecnologico dei principali ambiti su cui si basa il progetto. Si apre con un'analisi delle tecnologie immersive (XR), comprendente definizioni, dispositivi e ambiti applicativi, seguita da una rassegna sui fondamenti dell'intelligenza artificiale, dal simbolismo al deep learning, fino ai modelli linguistici di grandi dimensioni (LLM). Il capitolo approfondisce il ruolo dell'intelligenza artificiale generativa e delle tecnologie vocali (riconoscimento vocale e sintesi vocale), cruciali per l'interazione naturale nei contesti XR. Ampio spazio è dedicato ai chatbot, alla loro evoluzione e classificazione, con particolare attenzione alle applicazioni aziendali. In chiusura, viene proposta una panoramica critica delle soluzioni esistenti in ambito accademico e industriale, evidenziando i limiti attuali e motivando la direzione progettuale adottata nel lavoro.

- **Capitolo 3 – Metodo**: descrive in modo dettagliato il processo di progettazione e sviluppo dell'assistente conversazionale in ambienti XR, evidenziando l'architettura modulare e basata su eventi che ne garantisce la scalabilità e la riusabilità. Viene illustrata l'integrazione dei principali moduli software, tra cui Whisper per il riconoscimento vocale multilingua, Amazon Polly per la sintesi vocale realistica e le API di OpenAI per l'elaborazione semantica. Particolare attenzione è dedicata al *prompt engineering*, che consente la generazione dinamica e contestualizzata delle istruzioni per il modello linguistico. Il capitolo analizza inoltre la gestione dello stato globale del sistema, la sincronizzazione tra avatar multilingua e ambienti XR, la logica di spawning e il controllo delle trascrizioni vocali, con meccanismi di filtro, correzione e fallback volti a garantire interazioni fluide e robuste.

- **Capitolo 4 – Valutazione dei risultati e iterazioni di sviluppo**: analizza in modo dettagliato le prestazioni del sistema conversazionale attraverso una serie di test in scenari d'uso realistici, sia controllati che non controllati. La valutazione comprende aspetti quantitativi (tempi di risposta, stabilità, accuratezza della trascrizione) e qualitativi (naturalezza dell'interazione, coerenza semantica, percezione dell'utente). Vengono descritte inoltre le principali iterazioni di sviluppo, organizzate in fasi successive, che hanno permesso di affinare progressivamente la pipeline e l'architettura del sistema. Viene infine offerta una riflessione critica sui limiti riscontrati e sulle potenzialità evolutive, con

proposte concrete per ottimizzazioni future in termini di modularità, latenza, scalabilità e compatibilità con tecnologie emergenti.

- **Capitolo 5 – Conclusioni**: riassume i risultati principali del lavoro, sottolineando come l'integrazione modulare dei componenti abbia permesso di realizzare un sistema di assistenza vocale immersivo e funzionale in ambienti XR. Il capitolo riflette sull'importanza delle scelte progettuali adottate, evidenziando sia i punti di forza che le limitazioni emerse durante lo sviluppo. Infine, si propongono possibili evoluzioni future e scenari applicativi ampliati, indicando come il progetto possa rappresentare una solida base per ulteriori approfondimenti e innovazioni nel campo degli assistenti conversazionali immersivi.

2. Stato dell'arte

Il presente capitolo ha l'obiettivo di offrire un inquadramento teorico e tecnologico delle conoscenze che costituiscono il contesto scientifico del lavoro svolto. Vengono presentati i concetti fondamentali legati alla realtà estesa (XR) e alle sue componenti (realtà virtuale, aumentata e mista), con un richiamo alla loro evoluzione storica e ai dispositivi di visualizzazione.

Segue una panoramica sull'intelligenza artificiale, dalle prime impostazioni simboliche ai più recenti sviluppi del machine learning e del deep learning, fino ai modelli linguistici di grandi dimensioni (LLM), di cui vengono analizzate le architetture e le modalità di interazione tramite prompt. Vengono quindi approfondite le tecnologie vocali, in particolare il riconoscimento vocale automatico e la sintesi vocale, e il loro impiego in sistemi conversazionali.

Una sezione specifica è dedicata ai chatbot: ne vengono descritte le origini, la classificazione in base al funzionamento, e soprattutto il ruolo sempre più centrale che ricoprono nel contesto aziendale, sia in ambito operativo sia formativo.

In chiusura, vengono analizzati alcuni progetti esistenti, sia accademici sia commerciali, per individuare i principali punti di contatto e differenziazione rispetto al lavoro proposto.

2.1 Fondamenti di XR

Le tecnologie immersive rappresentano una delle più significative innovazioni degli ultimi decenni nel campo della comunicazione digitale e dell'interazione uomo-macchina. Esse si basano sull'idea di estendere o alterare la percezione dell'ambiente circostante tramite l'uso di dispositivi digitali, creando esperienze più coinvolgenti, interattive e naturali. Tali sistemi si avvalgono di un'ampia gamma di dispositivi hardware, tra cui visori, controller, sistemi di tracking e interfacce aptiche, che lavorano congiuntamente a soluzioni software per gestire ambienti tridimensionali, spazialità e interazioni in tempo reale. Questo permette di coinvolgere l'utente in modo multisensoriale, promuovendo esperienze che combinano percezione visiva, uditiva e motoria.

Nel contesto di questo elaborato, è fondamentale chiarire le definizioni e i confini di ciascuna di queste componenti, per comprenderne sia le specificità tecniche sia il

potenziale in applicazioni immersive. In particolare, si procederà dapprima a una distinzione tra VR, AR, MR per poi ripercorrerne brevemente l'evoluzione storica e dei principali dispositivi utilizzati, concludendo con un'analisi dei loro ambiti d'impiego nei settori più rilevanti.

2.1.1 Definizione e differenze tra VR, AR e MR

La realtà virtuale (VR), la realtà aumentata (AR) e la realtà mista (MR) rappresentano tre forme distinte di esperienze immersive, che si differenziano principalmente per il livello di immersione e per il modo in cui combinano il mondo fisico con contenuti digitali.

- **Virtual Reality (VR)**: La VR immerge completamente l'utente in un ambiente digitale simulato, isolandolo visivamente e acusticamente dalla realtà fisica. L'interazione avviene all'interno di un mondo interamente generato da computer, tipicamente attraverso l'uso di *Head-Mounted Displays* (HMD)[1] e controller tracciati. Questo approccio è ampiamente utilizzato in simulazioni, addestramento, giochi e applicazioni educative (Palmer et al., 2020).

- **Augmented Reality (AR)**: L'AR sovrappone elementi digitali (come testi, immagini o animazioni) alla realtà visibile, senza occultarla. Gli utenti continuano a percepire l'ambiente circostante, arricchito da informazioni aggiuntive visualizzate su smartphone, tablet o dispositivi indossabili come gli *smart glasses* (Xu et al., 2023). Un esempio comune è l'utilizzo dell'AR per guidare tecnici durante operazioni di manutenzione o per migliorare l'esperienza di visita nei musei.

- **Mixed Reality (MR)**: La MR rappresenta un ulteriore passo rispetto all'AR, poiché consente una vera integrazione e interazione tra contenuti virtuali e mondo fisico. In questo paradigma, gli oggetti digitali non si limitano a essere sovrapposti alla realtà, ma vi interagiscono dinamicamente, rispondendo a movimenti, superfici e condizioni ambientali in tempo reale[2] (Palmer et al., 2020).

Sebbene siano spesso considerate insieme sotto l'etichetta di realtà estesa (XR), queste tre tecnologie presentano differenze concettuali e tecniche rilevanti. Una

[1] I visori VR schermano completamente il campo visivo per offrire una percezione esclusivamente virtuale.

[2] Ad esempio, un oggetto virtuale può rimbalzare su un tavolo reale o essere oscurato da un ostacolo fisico.

chiara distinzione tra VR, AR e MR è fondamentale per comprendere le potenzialità e i limiti di ciascun approccio, specialmente quando si progettano soluzioni immersive in ambito aziendale, culturale o formativo.

2.1.2 Evoluzione storica della VR e dei suoi dispositivi

La realtà virtuale (VR) ha attraversato un lungo percorso evolutivo che affonda le sue radici già nella metà del XX secolo, con esperimenti pionieristici nel campo dell'interazione uomo-macchina. La nascita concettuale della VR può essere fatta risalire al 1962, quando Morton Heilig realizzò il *Sensorama*, un dispositivo elettromeccanico in grado di stimolare diversi sensi (vista, udito, olfatto e tatto) per offrire un'esperienza immersiva multisensoriale. Tuttavia, è con l'invenzione dell'*Head-Mounted Display* (HMD) da parte di Ivan Sutherland nel 1965, e la successiva realizzazione nel 1968 del cosiddetto *Sword of Damocles*, che si può parlare di un primo prototipo di realtà virtuale interattiva tridimensionale (Mestanza et al., 2021).

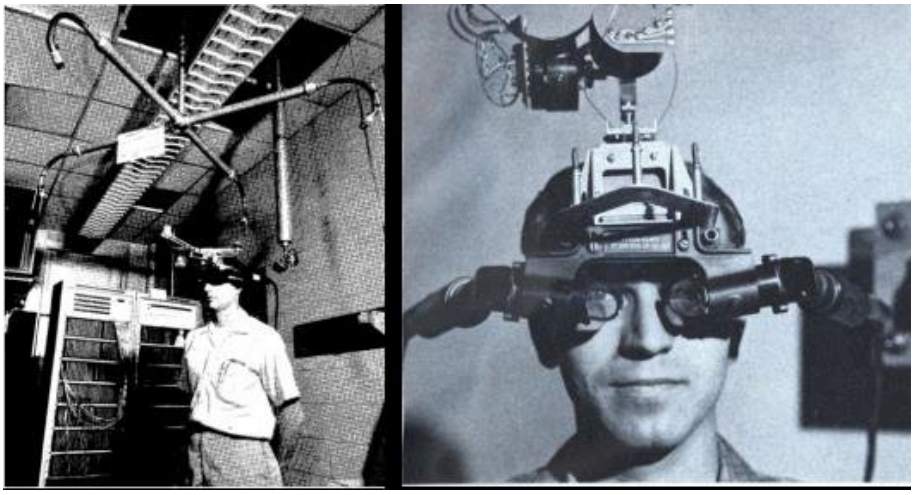


Figura 2.1: La *Sword of Damocles*, primo visore di realtà virtuale sviluppato da Ivan Sutherland nel 1968 (Liu et al., 2018).

Il *Sword of Damocles* rappresentava un visore pionieristico sospeso al soffitto tramite un braccio meccanico, necessario per sostenerne il peso eccessivo. Il dispositivo era in grado di generare immagini tridimensionali mediante un computer grafico, ma presentava limiti tecnici marcati: bassa risoluzione, latenza elevata, campo visivo ridotto e assenza di tracciamento accurato del movimento. Inoltre, l'ingombro fisico e la scarsa ergonomia lo rendevano inadatto a un uso prolungato o diffuso. Nonostante questi svantaggi, il contributo di Sutherland fu cruciale nel dimostrare la possibilità di creare ambienti virtuali interattivi, ponendo le basi teoriche e tecnologiche per i decenni successivi (Frey & Osborne, 2020).

Negli anni '80 e '90, la ricerca sulla VR conobbe un'espansione significativa. Jaron Lanier e la sua azienda VPL Research svilupparono i primi dispositivi commerciali, come il *DataGlove* per l'interazione manuale e l'*EyePhone* per la visione stereoscopica. Tuttavia, questi sistemi erano ancora troppo costosi, fragili e complessi per una diffusione di massa. I principali svantaggi includevano la presenza di *motion sickness*[3], dovuta all'elevata latenza, la qualità visiva limitata, la bassa frequenza di aggiornamento e la mancanza di contenuti adatti. I vantaggi, d'altra parte, riguardavano la possibilità di esplorare nuove modalità di interazione e simulazione, che risultavano già promettenti in ambiti come l'addestramento militare e aerospaziale (Xu et al., 2023).

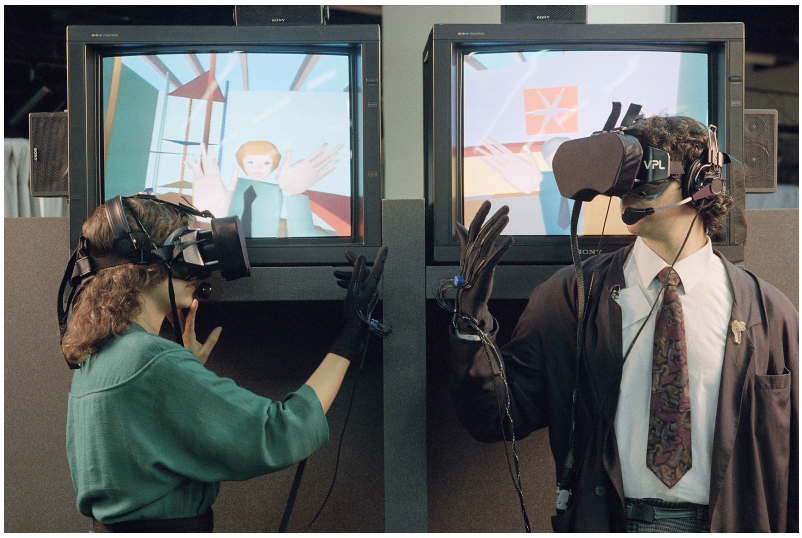


Figura 2.2: *EyePhone* e *DataGlove* sviluppati da VPL Research (Pape, 1999).

Una svolta decisiva si è verificata a partire dal 2012 con l'annuncio dell'Oculus Rift, seguito dall'acquisizione dell'azienda da parte di Facebook nel 2014. Da quel momento, la realtà virtuale ha beneficiato di investimenti consistenti che hanno favorito lo sviluppo di dispositivi più leggeri, autonomi, dotati di tracciamento a sei gradi di libertà (6DoF), risoluzioni elevate e sistemi di interazione avanzati. Visori come HTC Vive, Valve Index e Meta Quest 3 offrono oggi esperienze molto più fluide e naturali, con tempi di latenza ridotti e tracciamento preciso della posizione e del movimento delle mani, dello sguardo e persino delle espressioni facciali.

[3]Con il termine *motion sickness*, o cinetosi, ci si riferisce a un disturbo fisiologico causato da una discrepanza tra i segnali visivi e vestibolari percepiti durante l'esperienza immersiva. I sintomi comuni includono nausea, vertigini, mal di testa e disorientamento, e si manifestano soprattutto quando la latenza tra il movimento dell'utente e l'aggiornamento visivo è elevata.

Figura 2.3: Il visore Meta Quest 3, utilizzato come dispositivo principale per l'implementazione e il testing del progetto descritto in questo elaborato (Meta Platforms, Inc., 2024).

In sintesi, l'evoluzione della realtà virtuale e dei suoi dispositivi è frutto di un lungo processo di convergenza tra informatica grafica, elettronica di consumo, neuroscienze e intelligenza artificiale. Se i dispositivi iniziali rappresentavano prototipi teorici con gravi limitazioni, le tecnologie attuali offrono esperienze immersivi accessibili, con impatti concreti e misurabili in molteplici settori professionali. La tendenza attuale punta verso una miniaturizzazione dell'hardware, una maggiore integrazione con intelligenze artificiali conversazionali e sensori biometrici, e l'estensione verso ecosistemi XR più ampi e interconnessi.

2.1.3 Ambiti applicativi e settori di impiego della XR

Le tecnologie basate sulla XR trovano oggi impiego in una varietà crescente di settori, contribuendo alla trasformazione digitale dei processi produttivi, educativi e comunicativi. La realtà virtuale (VR), aumentata (AR) e mista (MR) si rivelano strumenti versatili in grado di migliorare l'efficacia operativa, supportare un apprendimento più concreto e coinvolgente, e ampliare le modalità attraverso cui i contenuti possono essere esplorati.

Ambito industriale e manifatturiero

Nel contesto industriale, la XR è utilizzata per la progettazione collaborativa, il collaudo virtuale, la simulazione di impianti e la formazione tecnica. La realtà virtuale consente la modellazione e l'analisi di prototipi tridimensionali in ambienti immersivi, riducendo tempi e costi di sviluppo. La realtà aumentata viene invece

impiegata per applicazioni di *assisted maintenance*, dove le istruzioni operative sono sovrapposte direttamente agli oggetti fisici, guidando i tecnici in tempo reale. In contesti manifatturieri complessi, queste tecnologie permettono una significativa riduzione degli errori umani e una maggiore efficienza nei cicli produttivi (AI Multiple, 2025).

Simulazioni in VR sono inoltre adottate per addestrare il personale in scenari ad alto rischio — come centrali energetiche, cantieri o miniere — permettendo di esercitarsi in ambienti controllati che replicano fedelmente situazioni reali ma pericolose (Wired, 2025).

Formazione, didattica e musei virtuali

L'ambiente educativo ha tratto grande beneficio dall'introduzione di tecnologie XR. L'apprendimento immersivo consente l'esplorazione attiva di concetti complessi attraverso simulazioni e ambienti interattivi. In ambito scolastico e universitario, la VR viene impiegata per visualizzare strutture anatomiche, simulare fenomeni fisici o ricostruire ambienti storici, favorendo una maggiore comprensione e partecipazione (Li et al., 2025).

I musei virtuali rappresentano un'applicazione peculiare della XR in ambito culturale: permettono l'accesso remoto a collezioni digitali, l'interazione con reperti 3D e la fruizione di mostre interattive. Tali soluzioni rendono l'esperienza museale accessibile anche a chi, per ragioni logistiche, fisiche o economiche, non potrebbe visitare gli spazi espositivi fisici.

Entertainment e gaming

L'industria dell'intrattenimento è tra le prime ad aver adottato la XR su larga scala. Le esperienze immersive offerte da videogiochi, simulatori e narrazioni interattive sfruttano il senso di presenza e l'interazione tridimensionale per creare un coinvolgimento profondo dell'utente. I mondi virtuali persistenti, come quelli proposti da piattaforme quali *VRChat*, *Horizon Worlds* o *Rec Room*, hanno favorito la nascita di nuove forme di socialità, eventi culturali e spettacoli partecipativi in spazi completamente digitali (XR Today, 2022).

La XR ha inoltre reso possibile la creazione di ambienti cinematografici immersivi e installazioni artistiche interattive, espandendo i confini della narrazione e dell'espressione creativa. Il coinvolgimento multisensoriale tipico di queste esperienze rafforza l'engagement dell'utente, offrendo nuove prospettive per l'industria culturale e dell'intrattenimento.

2.1.4 Sfide attuali e prospettive della XR

Nonostante i significativi progressi tecnologici, le tecnologie immersive—quali la realtà virtuale (VR), la realtà aumentata (AR) e la realtà mista (MR)—affrontano ancora numerose sfide che ne limitano l'adozione su larga scala. Le principali difficoltà riguardano aspetti tecnici, etici e socio-economici.

Sfide tecniche

Una delle principali sfide per l'adozione diffusa della XR riguarda la qualità dell'esperienza immersiva. Sebbene i dispositivi siano migliorati notevolmente in termini di risoluzione, latenza e tracciamento, molte soluzioni moderne soffrono ancora di motion sickness, soprattutto quando la latenza tra i movimenti dell'utente e la risposta dell'ambiente virtuale è elevata. Inoltre, la miniaturizzazione dell'hardware per visori sempre più leggeri e comodi rimane un obiettivo sfidante. Nonostante i progressi nella riduzione delle dimensioni, dispositivi come i visori VR richiedono ancora un notevole impegno in termini di alimentazione, raffreddamento e potenza di calcolo.

Un altro aspetto critico è la progettazione dell'interfaccia utente: la creazione di ambienti interattivi che siano intuitivi e facili da navigare in spazi tridimensionali rappresenta una sfida, soprattutto considerando la diversità delle persone che utilizzano questi sistemi, che includono utenti con abilità cognitive e fisiche differenti. Inoltre, l'integrazione della XR con altre tecnologie emergenti, come l'intelligenza artificiale e i sistemi di visione computerizzata, resta un campo di ricerca aperto, con sfide nell'interoperabilità tra piattaforme diverse.

Sfide etiche e sociali

La crescente immersione delle persone in ambienti virtuali solleva questioni etiche rilevanti. La gestione della privacy e dei dati personali è tra le preoccupazioni principali: dispositivi XR possono raccogliere una vasta gamma di informazioni sensibili, come la posizione, i movimenti corporei, le emozioni e perfino i tratti psicologici degli utenti (Alkaeed et al., 2023). La protezione di questi dati è fondamentale per evitare abusi e manipolazioni. Inoltre, l'adozione di tecnologie immersive solleva il problema del lato oscuro della virtualizzazione della vita sociale, che potrebbe favorire l'isolamento e ridurre la socializzazione nel mondo reale. Si tratta di un equilibrio delicato tra i benefici della connessione virtuale e i rischi di alienazione.

In ambito educativo e lavorativo, la dipendenza crescente dalla XR potrebbe esacerbare le disuguaglianze digitali. L'accesso alle tecnologie XR, che richiedono hardware avanzato e connessioni Internet veloci, è ancora limitato in molte regioni

del mondo e in popolazioni meno privilegiate. La formazione e l'acquisizione di competenze necessarie per utilizzare efficacemente la XR rappresentano ulteriori ostacoli, sia in contesti accademici che professionali.

Prospettive future

Nonostante queste sfide, le prospettive per la XR sono promettenti. I progressi nell'intelligenza artificiale e nel machine learning potrebbero migliorare significativamente l'esperienza immersiva, permettendo una personalizzazione avanzata delle interfacce e una gestione più intelligente dei contenuti. Ad esempio, l'integrazione di sistemi di AI conversazionale potrebbe aprire nuove modalità di interazione tra l'utente e l'ambiente virtuale, facilitando la navigazione, la formazione e la consulenza.

Inoltre, la convergenza tra XR e altre tecnologie emergenti, come blockchain[4], e 5G, promette di superare le attuali limitazioni. La rete 5G, con la sua bassa latenza e alta capacità di banda, permetterà esperienze più fluide e accessibili, mentre la blockchain potrebbe essere utilizzata per gestire la proprietà digitale, la sicurezza e la privacy all'interno degli ambienti virtuali. Un'altra area promettente è la realtà aumentata, che sta guadagnando terreno in applicazioni pratiche, dalla progettazione industriale alla medicina, consentendo di visualizzare informazioni sovrapposte alla realtà in tempo reale, migliorando decisioni e processi operativi.

Infine, l'evoluzione dei dispositivi indossabili verso modelli più leggeri, ergonomici e accessibili contribuirà a rendere la XR una tecnologia più alla portata di un pubblico più ampio. L'adozione su larga scala della XR, con la sua potenziale capacità di cambiare il modo in cui lavoriamo, apprendiamo e interagiamo, potrebbe rappresentare una rivoluzione simile a quella che ha accompagnato l'ingresso di Internet nella vita quotidiana.

In sintesi, mentre le sfide tecniche, etiche e sociali sono significative, le opportunità offerte dalla XR sono vastissime. Con il progresso delle tecnologie di supporto, l'interazione sempre più naturale e l'accessibilità globale, il futuro della realtà estesa sembra destinato a trasformare profondamente la nostra esperienza del mondo.

[4]La *blockchain* è una tecnologia di registro distribuito che consente di registrare transazioni in modo sicuro, trasparente e immutabile, senza la necessità di un'autorità centrale. Ogni blocco contiene un insieme di dati e un riferimento crittografico al blocco precedente, rendendo il sistema resistente a manomissioni.

2.2 Intelligenza Artificiale: origini e sviluppi

L'intelligenza artificiale (IA) è un campo multidisciplinare che si propone di progettare sistemi in grado di svolgere compiti complessi tipicamente associati all'intelligenza umana, come il riconoscimento visivo, il ragionamento logico, l'elaborazione del linguaggio naturale e l'apprendimento dall'esperienza. La sua evoluzione ha attraversato più fasi, dai primi approcci simbolici basati su regole logiche fino alle odierne tecniche di apprendimento automatico e ai modelli neurali su larga scala (Russell & Norvig, 2021).

2.2.1 Ambiti applicativi e contesti d'impiego

L'intelligenza artificiale è ormai una tecnologia trasversale, impiegata in numerosi settori grazie alla sua capacità di supportare decisioni, automatizzare compiti complessi e analizzare grandi quantità di dati in modo efficiente. Tra le applicazioni più consolidate vi sono i sistemi di supporto alla diagnosi medica, la guida autonoma, la rilevazione delle frodi nel settore finanziario e la sicurezza informatica.

Nel contesto industriale e commerciale, l'IA è spesso utilizzata per ottimizzare processi produttivi, gestire la logistica, o personalizzare l'offerta in base ai comportamenti degli utenti. In ambito consumer, invece, è presente in dispositivi quotidiani come assistenti vocali, algoritmi di raccomandazione e dispositivi smart. Altri ambiti emergenti includono l'agricoltura di precisione, le smart city e i sistemi educativi adattivi.

Particolare rilevanza assumono tecnologie come la visione artificiale, l'elaborazione automatica del linguaggio naturale (NLP), i sistemi di raccomandazione e i modelli predittivi, che trovano applicazione tanto nei servizi digitali quanto nei processi fisici e industriali.

2.2.2 Dalla logica simbolica al machine learning

L'intelligenza artificiale nasce negli anni '50 con un'impostazione nota come approccio simbolico. In questa fase iniziale, i ricercatori cercavano di replicare il ragionamento umano attraverso regole logiche esplicite e strutture formali per rappresentare la conoscenza. I sistemi erano progettati per seguire istruzioni precise fornite dagli esperti, funzionando bene in contesti limitati e ben definiti.

Tuttavia, con l'aumentare della complessità dei problemi reali, emersero i limiti di questo modello: era difficile anticipare e codificare tutte le possibili situazioni,

e i sistemi non erano in grado di adattarsi a nuove informazioni. La mancanza di flessibilità e la forte dipendenza dalla conoscenza pre-programmata ne limitarono la diffusione.

A partire dagli anni '80, con l'aumento della potenza di calcolo e la crescente disponibilità di dati, emerse un nuovo paradigma: il *machine learning*. In questo approccio, l'intelligenza non deriva da regole scritte a mano, ma dalla capacità del sistema di apprendere direttamente dai dati. L'algoritmo "impara" a individuare pattern e regolarità, migliorando le proprie prestazioni nel tempo senza interventi espliciti.

Questo cambiamento ha segnato una svolta nella storia dell'IA, rendendo possibile lo sviluppo di sistemi più adattivi, efficienti e applicabili a una vasta gamma di problemi, dall'analisi predittiva al riconoscimento automatico delle immagini e del linguaggio.

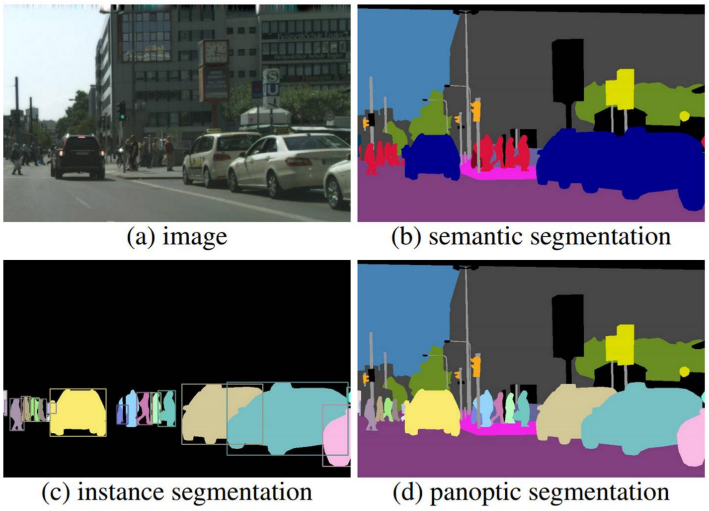


Figura 2.4: Rappresentazione grafica di un modello di riconoscimento automatico delle immagini (Aphex34, 2017).

2.2.3 Deep learning e modelli neurali

Il deep learning è una branca del machine learning che si basa sull'utilizzo di reti neurali artificiali, strutture matematiche ispirate al funzionamento del cervello umano. Queste reti sono composte da numerosi *strati* di nodi, chiamati neuroni artificiali, che lavorano in modo coordinato per analizzare e comprendere i dati in ingresso. La principale caratteristica del deep learning è l'impiego di *reti neurali profonde*, ovvero reti che possiedono molti strati nascosti tra l'input e l'output del modello. Grazie a

questi numerosi strati, la rete è in grado di apprendere rappresentazioni dei dati a diversi livelli di complessità, permettendo di identificare schemi e caratteristiche che non sono immediatamente evidenti a un'analisi superficiale.

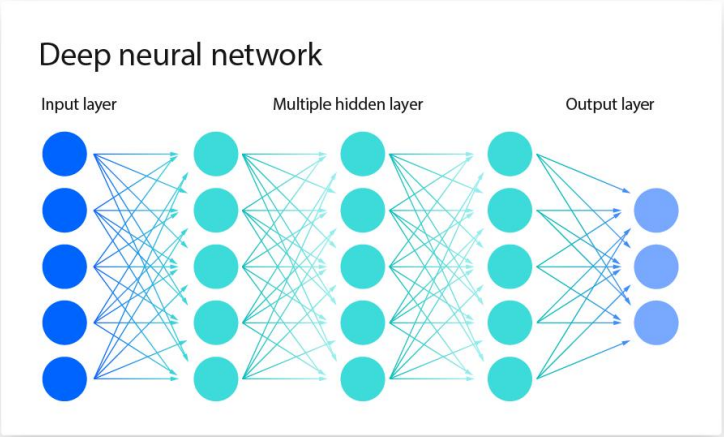


Figura 2.5: Schema di una rete neurale profonda (IBM, 2021).

In un modello di deep learning, ogni strato funziona come un filtro che estrae informazioni specifiche a vari livelli di astrazione. Per esempio, nel riconoscimento delle immagini, gli strati iniziali potrebbero rilevare elementi semplici come bordi e angoli, mentre quelli successivi combinano queste caratteristiche in forme più complesse come texture o parti di oggetti, fino ad arrivare a riconoscere interi oggetti o scene complesse. Questa capacità di costruire rappresentazioni gerarchiche permette alle reti neurali profonde di affrontare problemi molto complessi con un'efficacia superiore rispetto ai metodi tradizionali, riducendo la necessità di interventi manuali nella fase di pre-processing dei dati.

Un tipo molto diffuso di rete neurale profonda è la *rete neurale convoluzionale (CNN)*, particolarmente adatta per il riconoscimento delle immagini e altri compiti di visione artificiale. Le CNN sono progettate per sfruttare la struttura spaziale delle immagini, applicando un'operazione matematica chiamata convoluzione che permette di esaminare piccole porzioni di un'immagine (detti *filtri* o *kernel*). Questa operazione consente di identificare caratteristiche locali come bordi, angoli o texture, che vengono poi aggregate attraverso vari strati per formare una rappresentazione globale e più astratta dell'immagine (Krizhevsky et al., 2012).

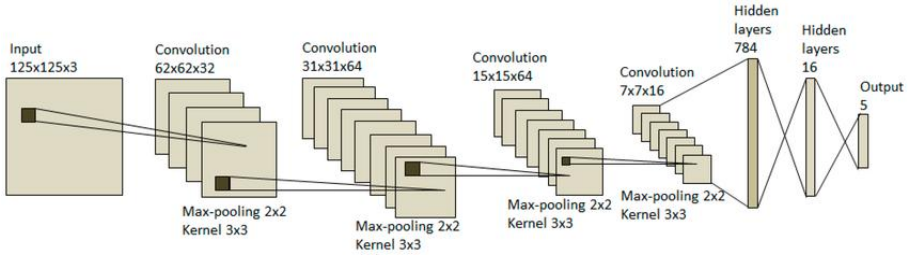


Figura 2.6: Schema di una rete neurale convoluzionale (CNN) che mostra i vari strati fondamentali e il flusso di dati tra essi (ResearchGate, 2019).

Un altro tipo di rete neurale importante è la *rete neurale ricorrente (RNN)*, progettata per l'elaborazione di dati sequenziali, come testi, segnali audio o serie temporali. A differenza delle CNN, che elaborano dati con una struttura spaziale fissa, le RNN sono in grado di "ricordare" informazioni relative a dati elaborati in precedenza grazie alla loro struttura ciclica. Questo permette alle RNN di mantenere uno stato interno che rappresenta informazioni passate, fondamentali per effettuare previsioni o decisioni basate sul contesto temporale, come nella traduzione automatica o nella sintesi vocale.

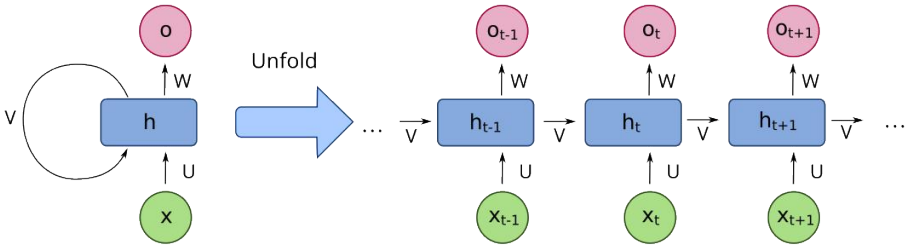


Figura 2.7: Struttura base di una rete neurale ricorrente (RNN) con il loop ricorrente, illustrata nella sua forma srotolata nel tempo (Goodfellow et al., 2016; Wikipedia contributors, 2024).

Per migliorare ulteriormente le prestazioni delle RNN, sono state introdotte le *Long Short-Term Memory (LSTM)*, una variante progettata per superare alcune limitazioni delle RNN tradizionali, come il problema della *scomparsa del gradiente* (Hochreiter & Schmidhuber, 1997). Questo problema limita la capacità delle reti di apprendere da informazioni che si trovano a lungo termine nelle sequenze di dati. Le LSTM risolvono questo ostacolo grazie a una struttura interna complessa, composta da celle e gate che controllano quali informazioni mantenere o dimenticare, consentendo di conservare in memoria informazioni rilevanti per periodi di tempo molto più lunghi.

Ciò rende le LSTM particolarmente efficaci in compiti come il riconoscimento del parlato, la generazione di testo o l'analisi di segnali temporali complessi.

L'addestramento delle reti neurali profonde avviene tramite un processo iterativo noto come *backpropagation*. Durante questo processo, il modello calcola l'errore tra la previsione e il risultato atteso e utilizza tale errore per aggiornare i pesi delle connessioni tra i neuroni, in modo da migliorare progressivamente le prestazioni. L'algoritmo di ottimizzazione più comune impiegato in questa fase è la *discesa del gradiente*, che mira a minimizzare l'errore complessivo del modello attraverso piccoli aggiustamenti iterativi, portando il modello a convergere verso una soluzione ottimale.

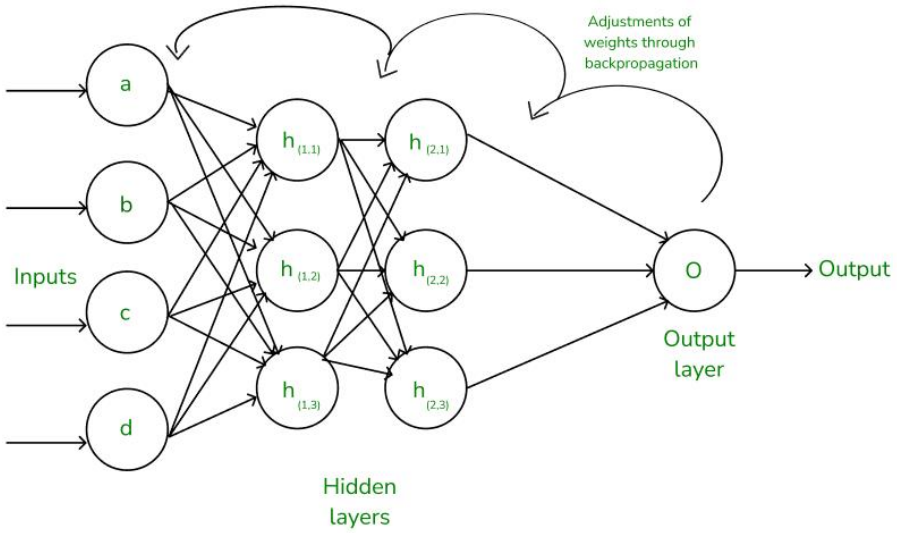


Figura 2.8: Diagramma del processo di backpropagation in una rete neurale (GeeksforGeeks, 2023).

Il successo del deep learning è stato reso possibile da una combinazione di fattori chiave: il significativo aumento della potenza computazionale (ad esempio grazie alle GPU), la disponibilità di grandi quantità di dati annotati (necessari per l'apprendimento supervisionato) e i continui miglioramenti nelle tecniche di ottimizzazione e progettazione delle architetture neurali. Questi progressi hanno permesso al deep learning di diventare una tecnologia centrale in numerosi settori applicativi, tra cui il riconoscimento vocale, la visione artificiale, la traduzione automatica e l'analisi di dati biologici (LeCun et al., 2015).

2.3 Modelli di linguaggio di grandi dimensioni (LLM)

I modelli di linguaggio di grandi dimensioni (Large Language Models, LLM) rappresentano una delle innovazioni più significative nell'ambito dell'intelligenza artificiale contemporanea. Basati su reti neurali avanzate e addestrati su quantità massive di dati testuali, questi modelli sono in grado di generare, comprendere e manipolare il linguaggio naturale con un livello di accuratezza che, in molti casi, si avvicina a quello umano.

Nel contesto di questa tesi, gli LLM rivestono un ruolo centrale, poiché costituiscono il cuore del sistema conversazionale progettato per interagire con l'utente in ambienti virtuali immersivi. Per comprendere appieno le potenzialità e i limiti di tali tecnologie, è necessario esaminare la loro evoluzione storica, il funzionamento delle architetture su cui si basano, le differenze tra i principali modelli attualmente disponibili e le implicazioni pratiche legate alla loro adozione.

2.3.1 Evoluzione degli LLM: dalle origini a oggi

La storia degli LLM si intreccia con l'evoluzione dell'elaborazione del linguaggio naturale (NLP) e delle reti neurali. I primi modelli NLP erano di tipo statistico, come gli n-grammi, basati sulla frequenza con cui certe sequenze di parole compaiono nei testi. Sebbene semplici, tali modelli non erano in grado di cogliere il significato o il contesto delle frasi.

Come già accennato nelle sezioni precedenti, l'avvento del *deep learning* ha introdotto architetture innovative come le Reti Neurali Ricorrenti (RNN) e le Long Short-Term Memory (LSTM), che hanno rappresentato un primo tentativo efficace di affrontare la complessità del linguaggio naturale. Questi modelli, pur avendo migliorato la gestione delle sequenze grazie alla loro capacità di mantenere informazioni contestuali nel tempo, mostravano tuttavia limitazioni intrinseche: la difficoltà nel cogliere dipendenze a lungo raggio e l'elaborazione strettamente sequenziale ne compromettevano l'efficienza e la scalabilità. Tali vincoli hanno aperto la strada a nuove soluzioni architetturali, culminate nella proposta del Transformer.

Un punto di svolta è arrivato nel 2017 con la pubblicazione dell'architettura *Transformer* (Vaswani et al., 2017), che ha introdotto un meccanismo innovativo chiamato *self-attention*. Questo approccio ha permesso di analizzare simultaneamente tutte le parole di una frase, modellando le relazioni tra di esse in modo più diretto e preciso, indipendentemente dalla loro distanza nella sequenza.

2.3.2 L'architettura Transformer: un cambio di paradigma

L'architettura Transformer ha rivoluzionato il campo dell'elaborazione del linguaggio naturale abbandonando l'elaborazione sequenziale tipica delle RNN a favore di un'elaborazione parallela. Questo ha permesso un notevole incremento della velocità sia nella fase di addestramento che nell'utilizzo pratico del modello, aprendo la strada a soluzioni più complesse e performanti.

Il componente chiave del Transformer è il meccanismo di *self-attention*, che consente al modello di attribuire un "peso" differente a ciascuna parola in base alla sua rilevanza nel contesto. In pratica, ogni parola viene messa in relazione con tutte le altre della sequenza, consentendo al modello di identificare facilmente nessi semantici e strutturali anche a lunga distanza. Questa capacità di modellare il contesto in modo flessibile e completo ha reso il Transformer la base architetturale di quasi tutti i moderni LLM.

2.3.3 Modelli principali e sviluppi recenti

Il primo modello basato su Transformer a ottenere ampio riconoscimento è stato BERT (2018), sviluppato da Google. A differenza dei modelli autoregressivi, BERT utilizza un approccio bidirezionale: per prevedere una parola mancante, considera simultaneamente sia il contesto precedente sia quello successivo. Questo lo rende particolarmente efficace nei task di comprensione del linguaggio.

Nel 2020, OpenAI ha presentato GPT-3, un modello generativo autoregressivo con 175 miliardi di parametri, addestrato a completare testo a partire da un prompt iniziale. A differenza di BERT, GPT-3 è progettato per generare testo in modo fluido e coerente, ed è stato reso disponibile tramite API pubbliche, contribuendo in modo decisivo alla diffusione dell'uso degli LLM su larga scala.

Negli anni successivi, sono stati introdotti numerosi altri modelli, ciascuno con obiettivi e caratteristiche specifiche:

- **Claude** (Anthropic): progettato con particolare attenzione alla sicurezza e all'allineamento con i valori umani.
- **LLaMA** (Meta): ottimizzato per leggerezza e compatibilità con ambienti di ricerca e dispositivi locali.
- **PaLM** (Google): caratterizzato da forti capacità logico-deduttive e multilingua, ha costituito la base per il successivo Gemini.

- **Gemini** (ex Bard): integrato nei servizi Google, aggiornato frequentemente, orientato all'interazione in tempo reale.
- **Mistral** e **Mixtral**: modelli europei open-source focalizzati su efficienza e bilanciamento tra performance e accessibilità.

Questa diversificazione riflette un panorama in continua evoluzione, dove la ricerca dell'equilibrio tra potenza computazionale, versatilità d'impiego e responsabilità etica guida lo sviluppo dei nuovi LLM.

2.3.4 Come funziona un LLM

Un LLM viene addestrato in due fasi principali: il *pre-training*, in cui apprende regolarità linguistiche generali analizzando grandi quantità di testo e predicendo parole mancanti o successive; e il *fine-tuning*, in cui viene ottimizzato su compiti specifici o reso più sicuro tramite tecniche come il Reinforcement Learning from Human Feedback (RLHF)[5].

Il cuore computazionale del modello è costituito da più strati Transformer, che applicano self-attention e normalizzazione per identificare pattern linguistici. L'output del modello è una distribuzione di probabilità su possibili parole successive, da cui viene selezionata quella più appropriata in base al contesto.

2.3.5 Differenze tra LLM: prestazioni, costi e usabilità

Dal punto di vista dell'utilizzatore finale o di un'azienda, le principali differenze tra gli LLM riguardano:

- **Velocità e latenza:** modelli come Claude Instant o GPT-3.5 Turbo offrono inferenza rapida, utile per applicazioni in tempo reale.
- **Costo:** modelli a pagamento (es. GPT-4, Claude 3) possono essere costosi a seconda del volume di token, mentre modelli open-source (LLaMA, Mistral) possono essere eseguiti localmente con maggior controllo dei costi.
- **Precisione e affidabilità:** modelli più avanzati offrono risposte più coerenti, ma richiedono più risorse.

[5]Il RLHF è una tecnica che combina l'apprendimento per rinforzo con valutazioni umane: i risultati generati dal modello vengono giudicati da persone, e queste preferenze vengono usate per aggiornare il modello, rendendolo più allineato alle aspettative umane.

- **Privacy e sicurezza:** l'uso di modelli locali può evitare l'invio di dati sensibili a server esterni.
- **Multilinguismo:** modelli come PaLM e Gemini gestiscono bene le lingue non inglesi, rendendoli adatti a contesti internazionali.

2.3.6 Prompt engineering e interazione con l'utente

L'efficacia di un LLM dipende in larga parte dalla qualità dell'input fornito, noto come *prompt*. Tecniche come il prompt engineering, l'uso di system prompt e la prompt chaining permettono di ottenere risposte più pertinenti e controllate.

In ambienti immersivi come quelli trattati in questa tesi, il prompt engineering diventa uno strumento fondamentale per guidare l'avatar virtuale nel fornire risposte coerenti, personalizzate e adatte al contesto.

2.3.7 LLM e ambienti XR: un'integrazione strategica

L'integrazione di modelli linguistici di grandi dimensioni (LLM) all'interno di ambienti immersivi rappresenta una direzione promettente per la creazione di esperienze conversazionali più naturali e coinvolgenti. In scenari XR, l'interazione con l'utente non si limita al testo, ma coinvolge voce, gesti e contesto visivo, richiedendo dunque sistemi in grado di comprendere il linguaggio in modo flessibile, multilingue e contestuale.

Per supportare un'interazione efficace in questi ambienti, è necessario che l'LLM sia in grado di gestire conversazioni in tempo reale, fornire risposte coerenti, interpretare richieste ambigue o parziali e adattarsi al linguaggio dell'utente. Altri aspetti critici includono la latenza nella generazione della risposta, la gestione della privacy dei dati, e la compatibilità con pipeline vocali e grafiche.

La scelta di un LLM adatto, così come le modalità con cui viene integrato all'interno del sistema, incide direttamente sulla qualità dell'esperienza utente. Un elemento centrale in questa prospettiva è rappresentato dalla crescente adozione di modelli generativi, la cui capacità di produrre contenuti originali — in forma testuale, vocale o visiva — sta trasformando il modo in cui si progettano le interazioni. Queste tecnologie ampliano il ventaglio delle possibilità applicative, permettendo sistemi più adattivi, espressivi e contestualmente consapevoli. Approfondire il ruolo dell'intelligenza artificiale generativa consente dunque di comprendere più a fondo l'evoluzione del paradigma conversazionale adottato nel progetto.

2.4 Intelligenza artificiale generativa

Negli ultimi anni, lo sviluppo dell'intelligenza artificiale ha subito una trasformazione significativa, spostandosi da un approccio analitico e classificatorio verso uno generativo. Questo cambio di paradigma ha aperto nuove prospettive applicative: non si tratta più soltanto di riconoscere o analizzare contenuti esistenti, ma di crearne di nuovi in maniera autonoma e plausibile. Tale capacità è resa possibile dall'impiego di modelli avanzati di *deep learning*, in particolare quelli generativi, che rappresentano una delle frontiere più dinamiche e promettenti della ricerca in ambito IA.

2.4.1 IA generativa: definizione, tecnologie e sviluppi

L'IA generativa è una branca dell'intelligenza artificiale focalizzata sulla produzione autonoma di contenuti, come testi, immagini, audio, video, codici e modelli 3D, attraverso algoritmi che apprendono strutture e regolarità nei dati. A differenza dei sistemi predittivi, l'obiettivo non è soltanto stimare una risposta a partire da un input, ma generare un output completamente nuovo, pur mantenendo coerenza, plausibilità e, in molti casi, utilità contestuale.

Le applicazioni dell'AI generativa sono molteplici e comprendono vari settori, tra cui:

- **Generazione di testi**: Modelli come GPT-3 (Generative Pre-trained Transformer) possono generare testi coerenti e articolati a partire da prompt iniziali, supportando applicazioni come la scrittura automatica di contenuti, assistenti virtuali e traduzione automatica.
- **Generazione di immagini e video**: Le Generative Adversarial Networks (GAN), composte da due reti neurali in competizione (generatore e discriminatore), possono produrre immagini e video realistici da descrizioni testuali o da input casuali, con applicazioni in design, entertainment e simulazioni.
- **Musica e suoni**: Algoritmi generativi possono creare musica originale, colonne sonore o effetti sonori, adatti a vari contesti, come film, giochi e assistenza a creatori di contenuti.
- **Simulazione e predizione**: L'AI generativa è anche usata in simulazioni scientifiche e industriali per predire comportamenti complessi in ambienti virtuali o nei processi di produzione.

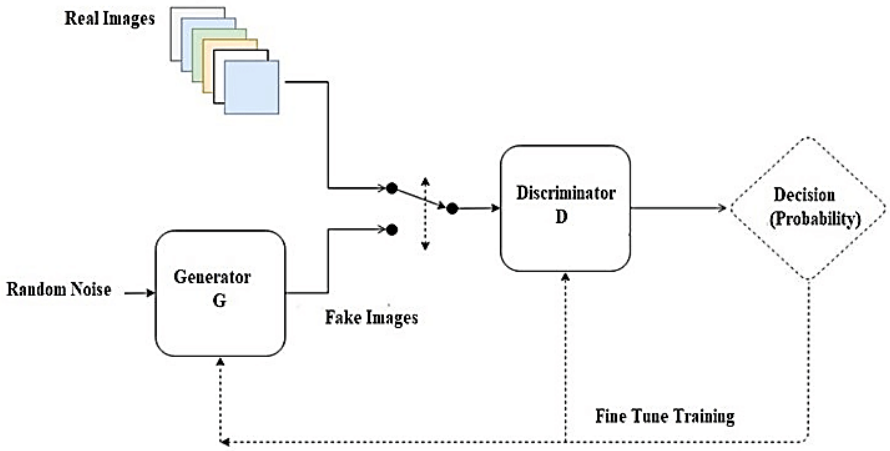


Figura 2.9: Diagramma generale di una rete generativa avversaria (GAN), in cui il generatore (G) produce immagini sintetiche a partire da un input casuale, mentre il discriminatore (D) cerca di distinguere tra immagini reali e generate (Mehmood et al., 2021).

In sostanza, queste tecnologie stanno rivoluzionando diversi ambiti creativi e tecnologici, consentendo non solo l'automazione di compiti ripetitivi, ma anche la creazione di nuovi contenuti, con applicazioni che spaziano dal settore culturale a quello aziendale, fino alla ricerca scientifica. Nonostante le numerose opportunità, emergono anche sfide rilevanti, come il controllo della qualità, la gestione dei bias, la trasparenza del processo generativo e la verifica dell'autenticità dei contenuti prodotti. Inoltre, l'integrazione efficace in ambienti interattivi e multimodali, come quelli XR, richiede un attento bilanciamento tra prestazioni, latenza, coerenza contestuale e interpretabilità.

2.4.2 Speech-to-text: il riconoscimento vocale come ponte tra voce e testo

All'interno dei sistemi interattivi basati su IA generativa, la voce rappresenta un'interfaccia naturale ed efficiente per l'input utente. Il *speech-to-text* (STT) è la tecnologia che consente di convertire il parlato in testo, rendendo possibile l'elaborazione linguistica da parte di modelli generativi.

Un esempio avanzato di tale tecnologia è *Whisper*, sviluppato da OpenAI. Si tratta di un modello multilingua, addestrato su grandi quantità di dati audio e trascrizioni, progettato per gestire accenti, rumori ambientali e parlato informale con elevata robustezza. A differenza dei sistemi STT tradizionali, Whisper è in grado di operare in contesti meno controllati e con un'accuratezza superiore, grazie alla

combinazione di architetture Transformer e tecniche di autoapprendimento[6].

Queste caratteristiche lo rendono particolarmente utile per applicazioni XR e interfacce conversazionali, dove è essenziale comprendere input vocali anche in condizioni subottimali.

2.4.3 Text-to-speech: voce sintetica per esperienze immersive

Complementare al riconoscimento vocale è la tecnologia *text-to-speech* (TTS), che consente di convertire testi in parlato naturale. In ambienti XR o in applicazioni con avatar vocali, il TTS consente all'IA di rispondere all'utente con una voce sintetica, migliorando l'engagement e la percezione di interazione umana.

Soluzioni come *Amazon Polly*, *Google Cloud TTS*, *Microsoft Azure Speech* e *IBM Watson TTS* offrono voci sintetiche personalizzabili, in molte lingue e accenti. I modelli più avanzati si basano su architetture neurali come *WaveNet*[7], che hanno portato la sintesi vocale a un livello qualitativo quasi indistinguibile dalla voce umana.

In particolare, molte di queste piattaforme offrono anche controlli su intonazione, ritmo, pause e stile espressivo, consentendo di adattare la voce sintetica al contesto comunicativo o emotivo desiderato.

2.4.4 Riflessioni sull'integrazione: voce e generazione nei sistemi intelligenti

Le tecnologie di *speech-to-text* (STT) e *text-to-speech* (TTS) rappresentano i canali fondamentali per la comunicazione uomo-macchina nei sistemi generativi interattivi. Quando integrate con modelli di IA generativa, permettono di costruire interfacce vocali intelligenti, capaci di comprendere, elaborare e rispondere in linguaggio naturale.

In contesti immersivi come gli ambienti XR, questa integrazione non solo aumenta l'accessibilità e la naturalezza dell'interazione, ma apre la strada a nuove forme di narrazione, formazione e presentazione dei contenuti. La possibilità di dialogare in modo fluido con un agente virtuale che comprende il parlato, genera risposte contestuali e le restituisce vocalmente rappresenta una delle applicazioni più avanzate dell'intelligenza artificiale conversazionale.

---

[6]Whisper è stato addestrato su circa 680.000 ore di dati audio, incluse trascrizioni rumorose e non perfettamente allineate, per migliorare la generalizzazione nei contesti reali.

[7]WaveNet, sviluppato da DeepMind, modella direttamente l'onda sonora per ottenere una qualità vocale estremamente naturale rispetto ai sintetizzatori concatenativi tradizionali.

Tale combinazione tecnologica è alla base dello sviluppo di sistemi noti come *chatbot intelligenti*, capaci di interagire con l'utente in modo multimodale. Nella sezione seguente, verranno approfonditi l'evoluzione, la classificazione e gli impieghi dei chatbot, con un'attenzione particolare al loro utilizzo in ambienti immersivi e al confronto con le soluzioni attualmente esistenti.

2.5 Chatbot e intelligenza artificiale conversazionale

2.5.1 Evoluzione storica dei chatbot

I chatbot rappresentano una delle applicazioni più longeve e affascinanti dell'intelligenza artificiale. Il termine stesso deriva dalla combinazione di "chat" (conversazione) e "bot" (robot), e si riferisce a programmi progettati per interagire con gli esseri umani attraverso il linguaggio naturale, simulando una conversazione [8].

La storia dei chatbot ha inizio negli anni '60 con il progetto *ELIZA*, sviluppato da Joseph Weizenbaum presso il MIT. ELIZA simulava un terapeuta rogersiano attraverso un semplice sistema di regole basato su *pattern matching*: identificava parole chiave nel messaggio dell'utente e rispondeva con frasi generiche, predefinite, spesso restituendo domande in forma riformulata.

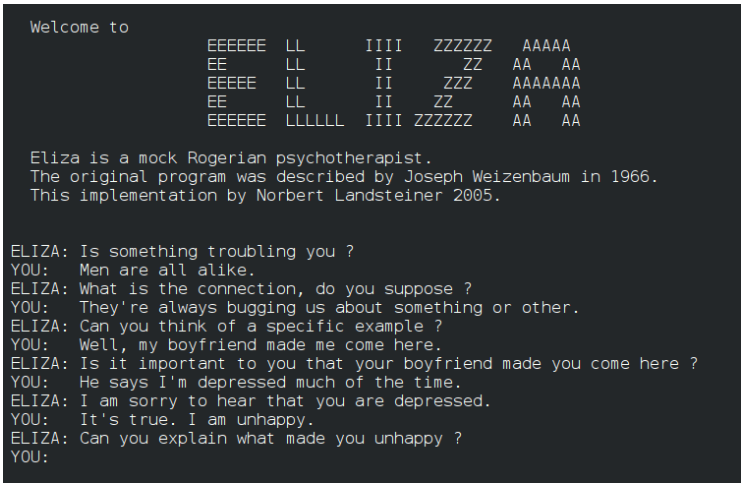


Figura 2.10: Screenshot di una conversazione con il chatbot ELIZA, che simula un terapeuta rogersiano attraverso un sistema di regole basato su pattern matching (Weizenbaum, 1966).

[8] Nel contesto attuale, il termine "chatbot" è spesso usato in modo generico per indicare sia sistemi molto semplici basati su regole, sia agenti conversazionali avanzati alimentati da modelli linguistici di grandi dimensioni.

Nonostante la totale assenza di comprensione semantica, ELIZA suscitò un forte impatto psicologico sugli utenti, dimostrando che anche una semplice imitazione di dialogo umano può generare un'impressione di intelligenza artificiale (Hill et al., 2020).

Negli anni '90, lo sviluppo di *ALICE* (Artificial Linguistic Internet Computer Entity) da parte di Richard Wallace introdusse un'ulteriore evoluzione. ALICE utilizzava un linguaggio specifico, AIML (Artificial Intelligence Markup Language), che permetteva la definizione di regole conversazionali complesse. Sebbene ancora privo di una reale comprensione del linguaggio, il sistema offriva una maggiore varietà e pertinenza nelle risposte rispetto a ELIZA, tanto da vincere più volte il premio Loebner per chatbot più realistico (Adamopoulou & Moussiades, 2020).

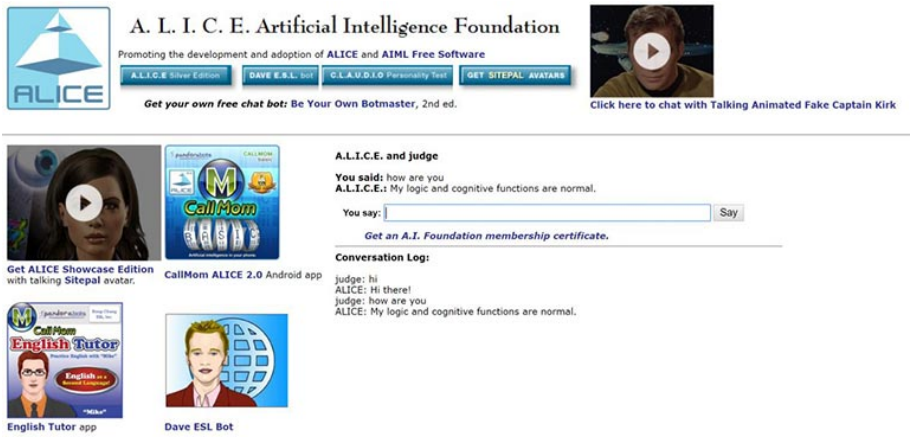


Figura 2.11: Immagine storica del chatbot ALICE(Onlim, 2025).

A partire dagli anni 2000, con la diffusione di internet e l'esplosione della messaggistica istantanea, i chatbot hanno cominciato a essere impiegati in ambito commerciale, per fornire supporto automatico ai clienti attraverso script sempre più articolati. Tuttavia, la vera svolta è arrivata con l'avvento del *machine learning*, e in particolare del *deep learning*, che ha permesso ai chatbot di apprendere direttamente dai dati e di adattarsi al contesto.

L'introduzione delle architetture neurali, e successivamente del modello *Transformer*, ha reso possibile la nascita di chatbot capaci di generare risposte coerenti, contestuali e articolate. Sistemi come Siri (Apple), Alexa (Amazon), Google Assistant, e successivamente i chatbot integrati nei modelli di linguaggio di grandi dimensioni come ChatGPT, Claude o Gemini, segnano il passaggio da sistemi reattivi e pre-programmati a veri e propri assistenti conversazionali (Shawar & Atwell, 2007).

Oggi, i chatbot non sono più confinati a rispondere a semplici domande frequenti, ma sono in grado di:

- gestire conversazioni multi-turno;
- comprendere il contesto e le intenzioni dell'utente (*intent recognition*);
- interagire vocalmente con l'utente;
- operare in ambienti ibridi, come applicazioni mobili, ambienti virtuali o spazi immersivi.

Questa evoluzione non è solo tecnica, ma anche concettuale: da strumento di automazione limitata, il chatbot si è trasformato in un'interfaccia intelligente tra l'utente e sistemi complessi, capace di apprendere, adattarsi e rispondere in modo naturale. Nelle sezioni seguenti, verranno analizzate le principali tipologie di chatbot, i loro impieghi in ambito aziendale e la loro integrazione in ambienti immersivi, con uno sguardo alle sfide tecniche e progettuali che accompagnano il loro utilizzo.

2.5.2 Tipologie di chatbot

I chatbot possono essere classificati in base alla tecnica utilizzata per interpretare l'input dell'utente e generare una risposta. In letteratura si individuano principalmente tre categorie, distinte per livello di complessità, flessibilità e intelligenza conversazionale (Shawar & Atwell, 2007):

- **Chatbot rule-based**: si basano su una serie di regole esplicite definite a priori da sviluppatori o esperti di dominio. Ogni possibile input è associato a un insieme di risposte predefinite, selezionate tramite semplici algoritmi di *pattern matching*. Sono facili da implementare e garantiscono risposte controllate, ma mostrano forti limiti in presenza di input ambigui, non previsti o complessi. ELIZA è uno degli esempi storici più noti di questa categoria.
- **Chatbot retrieval-based**: anziché generare nuove risposte, questi sistemi selezionano l'opzione migliore da un insieme di risposte già presenti in un database. L'analisi dell'input avviene tramite tecniche di *Natural Language Processing* (NLP), ovvero l'insieme di metodi computazionali utilizzati per analizzare, comprendere e interpretare il linguaggio naturale, oppure tramite modelli di machine learning supervisionato. Queste tecniche permettono di calcolare la similarità semantica tra domanda e risposta, migliorando la pertinenza delle risposte selezionate.

- **Chatbot generativi**: utilizzano modelli di linguaggio avanzati, spesso basati su architetture Transformer, per produrre risposte nuove e contestuali in tempo reale. Questi sistemi, come quelli basati su GPT (OpenAI), Claude (Anthropic) o LLaMA (Meta), non si limitano a selezionare una risposta esistente, ma apprendono dai dati linguistici per generare frasi coerenti e pertinenti a partire dall'input dell'utente. Offrono interazioni più fluide, personalizzabili e dinamiche, ma presentano sfide in termini di controllo, affidabilità e allineamento comportamentale.

Sebbene i chatbot rule-based e retrieval-based siano ancora utilizzati in contesti dove è necessario mantenere un elevato livello di controllo (ad esempio nel settore sanitario, legale o bancario), l'avanzamento dei modelli generativi ha ampliato le possibilità applicative, rendendo questi ultimi ideali per contesti conversazionali ricchi, dinamici e non rigidamente predefiniti.

In molti casi, i sistemi moderni adottano un approccio ibrido, combinando logiche retrieval-based con modelli generativi, per bilanciare precisione e creatività, specialmente in ambienti produttivi o in applicazioni immersive dove l'esperienza utente è centrale.

2.5.3 Chatbot nei servizi aziendali

Customer service e marketing

Nel contesto aziendale, i chatbot hanno trasformato radicalmente la gestione del servizio clienti, offrendo supporto continuativo, riducendo i tempi di attesa e abbattendo i costi operativi. Quando integrati in modo efficace all'interno dei processi digitali, possono contribuire in modo significativo a migliorare la qualità percepita del servizio e la soddisfazione del cliente (Adamopoulou & Moussiades, 2020).

In ambito marketing, i chatbot vengono utilizzati per la personalizzazione delle offerte, la gestione di campagne automatizzate, la raccolta di feedback e la guida nel processo d'acquisto. Grazie all'integrazione con CRM[9] e analisi comportamentale, è possibile offrire risposte e proposte altamente mirate, migliorando le conversioni[10].

[9]Il CRM (Customer Relationship Management) è un sistema utilizzato per gestire e analizzare le interazioni con i clienti, facilitando la personalizzazione del servizio e delle offerte.

[10]Un chatbot può proporre prodotti basati sulle azioni dell'utente in tempo reale, offrendo un'esperienza d'acquisto più efficace rispetto a un'interfaccia statica.

Formazione aziendale e knowledge management

Un altro ambito emergente è quello della formazione interna e della gestione della conoscenza. I chatbot aziendali possono fungere da tutor virtuali, rispondendo a domande sui processi interni, facilitando l'onboarding di nuovi dipendenti e supportando attività di microlearning. Secondo Shawar e Atwell (2007), questo tipo di applicazione è particolarmente utile in organizzazioni complesse, dove la documentazione è abbondante e frammentata.

Inoltre, i chatbot possono raccogliere metriche di utilizzo e feedback per migliorare continuamente i materiali formativi, trasformandosi in veri e propri strumenti dinamici di supporto alla crescita del capitale umano.

2.5.4 Chatbot in ambienti virtuali e XR

L'integrazione dei chatbot in ambienti immersivi, come la realtà virtuale (VR), aumentata (AR) e mista (MR), rappresenta un'estensione naturale della conversazione uomo-macchina verso modalità più coinvolgenti e multimodali. In scenari XR, i chatbot possono operare come guide, facilitatori o assistenti intelligenti che interagiscono con l'utente attraverso il linguaggio naturale, sia testuale che vocale.

Ad esempio, in un museo virtuale o in una simulazione industriale, un chatbot rappresentato da un avatar può fornire informazioni contestuali sugli oggetti presenti, rispondere a domande o guidare l'utente lungo un percorso di apprendimento personalizzato. La combinazione tra modelli generativi, riconoscimento vocale e sintesi vocale consente di realizzare interazioni fluide, intuitive e accessibili, anche in contesti multilingua o per utenti con disabilità.

Questa sinergia tra AI conversazionale e ambienti immersivi sarà ulteriormente approfondita nella prossima sezione, dedicata all'analisi di progetti esistenti che integrano tecnologie simili, e al confronto con il presente lavoro.

2.6 Soluzioni esistenti e confronto

2.6.1 Soluzioni attuali: industria e ricerca

Negli ultimi anni, il panorama delle tecnologie conversazionali e immersive ha visto una rapida espansione, con soluzioni che vanno da assistenti vocali integrati in dispositivi mobili fino a sistemi sperimentali in ambienti virtuali. Tuttavia, la maggior parte delle soluzioni commerciali mantiene ancora una netta separazione tra componenti vocali, avatar e capacità di dialogo naturale.

Da una parte, assistenti vocali come *Amazon Alexa*, *Google Assistant* e *Apple Siri* dominano il mercato consumer offrendo interazioni vocali sempre più naturali, ma raramente integrate in ambienti immersivi tridimensionali. Dall'altra, piattaforme aziendali come *IBM Watson Assistant*, *Intercom* o *LivePerson* propongono chatbot avanzati per l'assistenza clienti, ma la loro esperienza si limita principalmente a interfacce testuali o vocali su web e app mobili.

2.6.2 Esempi di progetti immersivi con chatbot

Diversi progetti accademici e prototipi all'avanguardia hanno iniziato a esplorare l'integrazione di chatbot in ambienti virtuali, con l'obiettivo di rendere l'interazione più naturale e coinvolgente.

Un esempio significativo è il sistema descritto da Kurniawan et al., che integra un chatbot con interfaccia vocale e avatar 3D dotato di espressioni facciali sincronizzate con la voce sintetica(Kurniawan et al., 2015). Tale approccio mostra come l'aggiunta di segnali non verbali e visivi aumenti il realismo e l'efficacia comunicativa, risultando particolarmente efficace in ambiti quali il customer care e l'e-learning.

Un caso di rilievo è anche il progetto della Hong Kong Metropolitan University, che ha sviluppato una simulazione immersiva per la preparazione a colloqui di lavoro in realtà virtuale. Il sistema sfrutta un chatbot generativo multilingue basato su modelli linguistici avanzati, ambienti 3D realistici e interfacce vocali per offrire un'esperienza interattiva completa con avatar intervistatori virtuali(Lee et al., 2024).

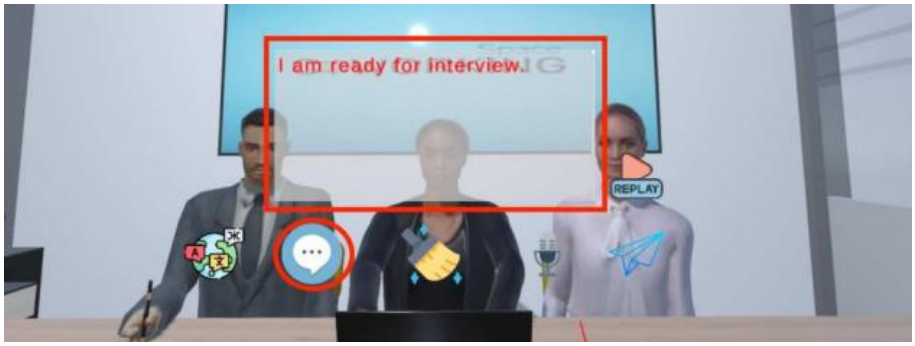


Figura 2.12: Esempio dell'interfaccia utente del sistema sviluppato dalla Hong Kong Metropolitan University, con visualizzazione in realtà virtuale e interazione mediata da avatar (Lee et al., 2024).

2.6.3 Limiti delle soluzioni esistenti

Nonostante i notevoli progressi, permangono alcune criticità comuni alle soluzioni attuali:

- Le piattaforme commerciali raramente integrano in modo completo e fluido riconoscimento vocale, avatar 3D animati e modelli linguistici generativi.
- I progetti accademici, pur innovativi, tendono a focalizzarsi su specifici scenari (formazione, customer care, simulazioni), limitando la scalabilità e la generalizzazione.
- La gestione di interfacce vocali multilingua avanzate resta una sfida aperta, soprattutto riguardo al riconoscimento di accenti, alla latenza e all'adattamento dinamico al contesto.

Per esempio, nel progetto di Kurniawan et al., la sincronizzazione labiale e le espressioni facciali migliorano l'impatto visivo, ma la pipeline si basa su moduli statici e non prevede un motore generativo flessibile per la produzione linguistica (Kurniawan et al., 2015). Allo stesso modo, la proposta di Lee et al. è molto efficace per la preparazione a colloqui ma si limita a un caso d'uso specifico, senza affrontare temi come la personalizzazione del linguaggio o l'espressività avatarica in tempo reale (Lee et al., 2024).

Questi limiti evidenziano l'esigenza di soluzioni che integrino in modo coerente componenti vocali, visive e conversazionali in ambienti immersivi, con un focus specifico su contesti aziendali.

2.6.4 Motivazione e direzione progettuale

Dall'analisi dei progetti esistenti emerge chiaramente come l'integrazione tra chatbot, tecnologie vocali e ambienti XR sia un campo ancora in via di sviluppo, caratterizzato dalla mancanza di soluzioni complete, fluide e accessibili. I contributi accademici tendono a concentrarsi su applicazioni specifiche, mentre le soluzioni commerciali spesso isolano le componenti anziché armonizzarle.

Queste osservazioni hanno guidato la definizione degli obiettivi di questo lavoro, che intende esplorare un'integrazione multimodale per migliorare l'interazione utente in ambienti virtuali aziendali. Nei capitoli successivi verrà illustrato un sistema conversazionale che combina modelli linguistici generativi, riconoscimento e sintesi vocale, e avatar animati in tempo reale, con particolare attenzione alla fruizione multilingue, alla coerenza espressiva e alla scalabilità.

3. Metodologia

3.1 Struttura generale del sistema

Il sistema è stato sviluppato all'interno di un ambiente Unity[1] utilizzando C# come linguaggio di programmazione, e si configura come un assistente virtuale intelligente integrato in uno spazio immersivo tridimensionale. Il progetto si inserisce in un contesto espositivo virtuale aziendale, ma è stato progettato in modo modulare e indipendente rispetto alla tecnologia XR sottostante, per garantire riutilizzabilità e facilità di integrazione in ambienti futuri.

L'architettura del chatbot è composta da una serie di sottosistemi altamente coesi e debolmente accoppiati, realizzati con un paradigma orientato agli eventi. Questo approccio consente di separare nettamente le responsabilità funzionali, agevolando il debugging, la scalabilità e la futura manutenzione del sistema.

Per la realizzazione del progetto sono state utilizzate una serie di tecnologie e servizi esterni per garantire le funzionalità di generazione, comprensione e sintesi del linguaggio naturale:

- **OpenAI (LLM)**: utilizzato per generare risposte testuali in linguaggio naturale, basato su modelli della famiglia `GPT` come `gpt-3.5-turbo`, `gpt-4-turbo` e `gpt-4o`.
- **Amazon Polly**: servizio di sintesi vocale impiegato per generare una voce realistica multilingua in tempo reale.
- **Whisper**: modello di speech-to-text sviluppato da OpenAI, usato per la trascrizione automatica dell'input vocale dell'utente.

Sebbene ciascuno di essi fornisca capacità avanzate in modo relativamente autonomo, il vero lavoro progettuale ha riguardato la loro combinazione in un ecosistema coerente, in grado di garantire reattività e fluidità nell'interazione.

Dal punto di vista dell'integrazione XR, il chatbot è stato sviluppato all'interno di un'applicazione preesistente fornita dal team XR di Akkodis, realizzata per visori

[1] https://unity.com/

Meta Quest. Tale applicazione era stata originariamente costruita utilizzando la versione `2021.3.11f1` di Unity e l'*Oculus Integration SDK*[2], attualmente deprecato ma ancora compatibile con Unity al momento dello sviluppo. La scelta di mantenere questa configurazione è stata dettata dalla necessità di garantire continuità e piena compatibilità con l'ambiente esistente, senza intervenire direttamente sulla componente XR.

Il chatbot è stato progettato come modulo indipendente e modulare, e successivamente verificato anche in ambienti basati su tecnologie XR differenti, come il *Meta XR All-in-One SDK*, e versioni più recenti del motore Unity. Con alcune modifiche mirate legate al cambiamento di versione Unity e SDK utilizzato, è stato possibile integrarlo correttamente anche in questi contesti, a conferma della sua portabilità e flessibilità architetturale.

Inoltre, gli avatar utilizzati per rappresentare il chatbot nelle diverse lingue sono stati creati tramite la piattaforma Ready Player Me[3], che consente di ottenere modelli 3D realistici compatibili con Unity e dotati di blend shape animabili per espressioni facciali e sincronizzazione labiale.

Figura 3.1: Esempio di creazione di un avatar con la piattaforma RPM.

L'integrazione del chatbot è stata realizzata in modo indipendente dalla logica specifica del museo o dai singoli SDK utilizzati. Il sistema si basa sull'ascolto e sulla gestione di eventi Unity, pertanto risulta facilmente integrabile in qualunque scena XR dotata di interazioni tramite pulsanti o trigger. Questo lo rende adatto a futuri contesti applicativi diversi dal museo, mantenendo la stessa architettura; infatti, per

[2]https://developer.oculus.com/downloads/package/unity-integration-sdk/
[3]https://readyplayer.me/

adattarlo a nuovi scenari sarebbe sufficiente modificare alcuni script o parametri, mentre la struttura di base rimarrebbe invariata.

I principali sottosistemi che compongono l'architettura del chatbot sono:

- **Gestione dello stato globale del chatbot**: un singleton centralizzato che gestisce la lingua, la posizione, lo stato attivo/inattivo e i sottostati della conversazione (es. registrazione, elaborazione, risposta).
- **Spawn e rotazione dinamica degli avatar**: sistema che attiva l'avatar corrispondente alla lingua corrente e alla posizione XR dell'utente nel momento dell'attivazione, posizionandolo in modo coerente e orientandolo verso il giocatore per simulare una comunicazione naturale.
- **Prompt Engineering dinamico**: generazione contestuale e multilivello del prompt, sia per il modello linguistico che per il sistema di trascrizione vocale (Whisper), adattati in base alla lingua selezionata, alla posizione dell'utente nello spazio XR, ai vincoli aziendali e progettuali, e al comportamento desiderato del chatbot.
- **Riconoscimento vocale in tempo reale con Whisper**: sistema di registrazione segmentata che rileva e gestisce automaticamente i silenzi, applica timeout di sicurezza e invia l'audio in streaming al modello Whisper. In parallelo gestisce la trascrizione incrementale, e l'eventuale rimozione del prompt testuale che Whisper talvolta include erroneamente all'inizio o alla fine della trascrizione finale.
- **Correzione automatica delle trascrizioni STT**: applicazione di sostituzioni predefinite per errori ricorrenti nel riconoscimento vocale, al fine di migliorare la precisione semantica del contenuto inviato al modello linguistico. Ad esempio, il nome proprio "*Akkodis*" veniva frequentemente trascritto in modo errato come "*a codis*", rendendo necessaria una correzione automatica a valle della trascrizione.
- **Filtraggio automatico di frasi vuote o di cortesia**: il sistema è in grado di identificare e rimuovere automaticamente segmenti di testo che non apportano contenuto informativo rilevante, come frasi vuote, espressioni di cortesia o prive di significato concreto. Questa operazione è fondamentale per evitare che il chatbot generi risposte spurie, vaghe o fuori contesto, migliorando così la coerenza e la qualità del dialogo.

- **Interazione con LLM e gestione del contesto**: invio asincrono delle trascrizioni al modello linguistico, con inclusione del prompt dinamico, mantenimento dello storico della conversazione e gestione di eventi speciali (es. cambio lingua, errori, rilanci automatici) per garantire coerenza e continuità dell'interazione.

- **Sintesi vocale realistica con Amazon Polly**: generazione vocale delle risposte del chatbot attraverso le API di Amazon Polly, con selezione automatica della voce in base alla lingua attiva. L'audio generato viene riprodotto sull'avatar e utilizzato come base per la sincronizzazione labiale mediante visemi[4] applicati ai blend shapes.

- **Logging testuale**: un sistema dedicato che registra eventi rilevanti durante l'esecuzione del chatbot, come trascrizioni, errori, risposte e cambi di stato. Il logging supporta formattazione avanzata (colori, stile, timestamp) e viene gestito da uno script autonomo, pensato sia per il debugging interno all'applicazione sia per permettere all'utente di visualizzare in tempo reale trascrizioni, domande e altri dettagli del dialogo.

- **Animazioni facciali e corporee sincronizzate**: le animazioni dell'avatar vengono orchestrate dinamicamente durante la conversazione, sfruttando il lip sync generato a partire dall'audio sintetizzato e integrando movimenti corporei legati agli stati `Idle` e `Talking`. La gestione si adatta alla lingua selezionata e alla voce attiva, migliorando la naturalezza dell'interazione ed evitando effetti artificiali, anche in presenza di avatar con strutture differenti.

3.1.1 Struttura gerarchica del Chatbot

Per una migliore comprensione dell'architettura interna, in Figura 3.2 è riportata la struttura Unity del prefab `Chatbot`.

Essa evidenzia la suddivisione logica tra:

- **Visuals**: contiene i prefab multilingua degli avatar, ciascuno dei quali è composto da due GameObject principali: `Armature`, che definisce la struttura scheletrica tramite un albero gerarchico di trasformazioni, e `Renderer`, che incorpora i componenti responsabili dell'animazione, della sorgente sonora,

[4] I *visemi* (dall'inglese *visemes*) sono le unità visive del linguaggio parlato, ovvero le configurazioni della bocca associate a specifici suoni o gruppi di fonemi. Vengono utilizzati per animare realisticamente il labiale in applicazioni vocali o 3D.

della sincronizzazione labiale e degli aspetti visivi. Quest'ultimo include al suo interno oggetti figli dotati di `Mesh Renderer`, materiali e texture, suddivisi per le diverse parti del corpo.

- **Logic**: racchiude tutta la logica di sistema, compresi i manager per la conversazione (`ChatGPT`), la sintesi (`TextToSpeech`), la trascrizione (`WhisperDirectText`) e il controllo del contesto.

- **ChatbotCanvas**: GameObject che ospita il componente `ChatbotLog`, responsabile della gestione e visualizzazione in tempo reale dei log, delle trascrizioni e degli eventi generati dal chatbot. Le canvas, figlie di questo GameObject, costituiscono l'interfaccia utente modulare per il log e il debugging e sono progettate per essere indipendenti dalla tecnologia XR utilizzata, facilitandone così l'integrazione in diversi contesti.

Figura 3.2: Struttura ad albero del prefab `Chatbot` all'interno della gerarchia Unity.

3.2 Gestione dello stato globale

Il componente `ChatbotGlobalData.cs`, collocato sul GameObject `ChatbotManager`, funge da *singleton* centralizzato responsabile della gestione dello stato globale del sistema conversazionale durante l'intera esecuzione. Questa architettura consente a tutti gli altri script di accedere direttamente e modificare in modo coerente le variabili di contesto fondamentali per il funzionamento del chatbot, mentre la propagazione delle modifiche avviene tramite eventi specifici, evitando accessi ridondanti o controlli ciclici inefficienti.

```
public event Action<Language> OnLanguageChanged;
public event Action<ChatbotState> OnChatbotStateChanged;
public event Action<ChatbotLocation> OnLocationChanged;
public event Action<ChatbotActiveSubState> OnChatbotActiveSubStateChanged;
```

Figura 3.3: Eventi reattivi generati da ChatbotGlobalData.cs per notificare i cambiamenti di stato agli altri moduli del sistema.

Tra le informazioni principali gestite dal componente si trovano:

- **Dati di contesto operativo:**
  - **Lingua attiva** (`AnswersLanguage`): rappresentata tramite un `enum`, può essere aggiornata dinamicamente sia da UI che da script. Al cambio di lingua vengono notificati tutti i moduli coinvolti nella generazione del prompt, nella sintesi vocale, nel riconoscimento vocale e dello spawn dell'avatar al fine di garantire una coerenza tra le componenti del sistema.
  - **Stato generale del chatbot** (`ChatbotState`): tiene traccia dello stato attuale del sistema conversazionale, distinguendo tra modalità `Active` e `Inactive`. La variazione di stato ha effetto trasversale sull'intero sistema, propagandosi a tutti i moduli coinvolti, inclusi quelli di spawn e gestione dell'interfaccia, assicurando un comportamento coerente e sincronizzato.
  - **Sottostato attivo** (`ChatbotActiveSubState`): permette un tracciamento più fine della fase corrente del sistema, distinguendo momenti come *Registrazione*, *Analisi* o *Risposta*, utili anche per la gestione di logica condizionale nei flussi di interazione. Il sottostato viene utilizzato internamente da moduli come la sintesi vocale (Polly) e il riconoscimento vocale (Whisper) per sincronizzare il comportamento alle fasi attive dell'interazione.

  - **Location corrente** (`ChatbotLocation`): rappresenta la posizione nello spazio XR in cui il chatbot viene istanziato, corrispondente alla posizione dell'utente al momento dell'attivazione. Questa informazione è fondamentale per la selezione dinamica e contestuale dei prompt, poiché la location di spawn determina indirettamente il contesto in cui il chatbot opera (es. Reception, Automotive). La location influenza direttamente sia lo *spawn* del chatbot, sia il *prompt engineering*.

- **Dati di configurazione:**
  - **Credenziali centralizzate** per i servizi esterni: le chiavi di accesso a OpenAI e Amazon Polly sono memorizzate in dizionari e accessibili a runtime.
  - **Nome localizzato del chatbot**: il nome del chatbot cambia in base alla lingua selezionata, grazie a un dizionario interno. Il nome può essere aggiornato dinamicamente tramite proprietà pubbliche.

Le credenziali di accesso ai servizi OpenAI e Amazon Polly, necessarie rispettivamente per l'utilizzo delle API di generazione del linguaggio naturale e di sintesi vocale, sono attualmente memorizzate in chiaro e accessibili a runtime. Si tratta di una scelta consapevole, adottata esclusivamente in ambito di sviluppo interno, in quanto il progetto non è destinato alla distribuzione pubblica né all'esecuzione su server esposti. In contesti di produzione o distribuzione esterna, tali credenziali andrebbero invece gestite in modo sicuro tramite meccanismi di cifratura, file esterni protetti o sistemi di gestione dei segreti (es. AWS Secrets Manager o variabili d'ambiente). Il loro ruolo è fondamentale per autenticare le richieste ai servizi cloud e abilitare funzionalità critiche come la trascrizione vocale, la generazione delle risposte e la restituzione audio, garantendo l'accesso ai modelli selezionati e ai pacchetti vocali configurati.

Eventi reattivi

Le modifiche ai dati di contesto operativo, come lingua attiva, stato generale, sottostato o location, generano automaticamente eventi dedicati, tra cui `OnChatbotStateChanged`, `OnLocationChanged`, `OnChatbotActiveSubStateChanged` e `OnLanguageChanged`. Tali eventi rappresentano la risposta diretta a cambiamenti nei valori interni delle variabili condivise, garantendo la propagazione coerente delle informazioni tra i componenti del sistema.

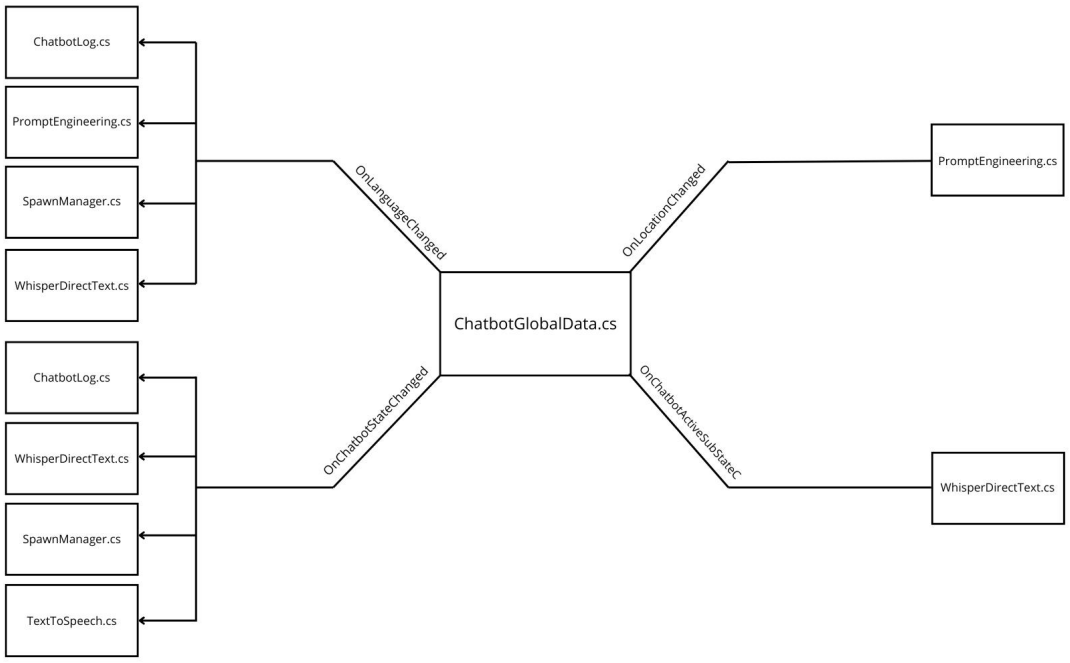


Figura 3.4: Schema degli eventi generati da `ChatbotGlobalData.cs` e dei moduli coinvolti nella propagazione dei cambiamenti di stato. Ogni evento notifica selettivamente i componenti interessati per mantenere la coerenza del sistema.

Questo approccio orientato agli eventi evita il ricorso al *polling*[5], migliora le performance ed esplicita i flussi di aggiornamento fra i componenti, favorendo scalabilità e manutenibilità.

Supporto per messaggi localizzati

All'interno del componente `ChatbotGlobalData.cs` sono definite diverse funzioni di utilità per il recupero di messaggi testuali localizzati in base alla lingua attiva. In particolare, le funzioni `GetLocalizedGreeting()`, `GetLocalizedProcessingMessage()` e `GetInactivityMessage()` restituiscono rispettivamente: un messaggio di saluto, una frase di attesa durante l'elaborazione della trascrizione e un messaggio di chiusura per inattività.

Queste stringhe sono collegate direttamente alla lingua selezionata tramite l'enumerazione `Language` e vengono generate senza l'intervento del modello linguistico.

[5]Il *polling* è una tecnica di programmazione in cui un componente controlla periodicamente lo stato di un altro componente per rilevare cambiamenti, comportando un uso inefficiente delle risorse e potenziali ritardi. Al contrario, l'uso di eventi permette una comunicazione più efficiente e reattiva.

L'approccio è stato scelto per due motivi principali: garantire una risposta coerente, chiara e immediata in scenari comuni (es. primo avvio, timeout, silenzio prolungato) e ridurre i costi e la latenza evitandone il calcolo tramite LLM.

Lingua	**Saluto**	**Elaborazione**	**Inattività**
Italiano	Salve, il mio nome è Marta. In cosa posso essere d'aiuto?	Elaborazione in corso, si prega di attendere.	La sessione è stata terminata per inattività. Se ha ulteriori domande, può avviare una nuova conversazione.
Inglese	Hello, my name is Emma. How may I assist you?	Processing request, please wait.	The session has been ended due to inactivity. If you have any further questions, you may start a new conversation.
Francese	Bonjour, je me nomme Claire. En quoi puis-je vous assister?	Traitement en cours, veuillez patienter.	La session a été clôturée en raison d'une inactivité prolongée. Pour toute autre question, veuillez démarrer une nouvelle conversation.
Spagnolo	Buenos días, mi nombre es Sofía. ¿En qué puedo asistirle?	Procesando la solicitud, por favor espere.	La sesión se ha cerrado por inactividad. Si tiene más preguntas, puede iniciar una nueva conversación.

Tabella 3.1: Esempi di messaggi localizzati predefiniti in base alla lingua attiva

Durante l'esecuzione, questi messaggi localizzati vengono direttamente inoltrati allo script `TextToSpeech.cs`, che li converte in audio tramite Amazon Polly. Il risultato è un'esperienza utente fluida, multilingua e responsiva, in cui i messaggi standard sono predefiniti e pienamente controllabili.

3.3 Spawn dinamico e gestione multi-avatar

Lo script `SpawnManager.cs` è responsabile della gestione spaziale degli avatar e del loro corretto posizionamento nelle varie aree dell'ambiente XR. Il sistema supporta avatar multilingua precaricati in scena e determina quale attivare in base alla lingua selezionata e alla posizione corrente dell'utente nel momento dell'attivazione. L'attivazione effettiva del chatbot avviene solo su esplicita richiesta dell'utente, ad esempio tramite la pressione di un apposito pulsante UI, e solo se ci si trova all'interno di una zona abilitata.

La logica di spawn[6] è stata progettata per essere scalabile e adattabile a diversi scenari. Il numero di zone, o posizioni, in cui il chatbot può comparire varia in base alla tipologia di progetto. Nel caso specifico di un museo aziendale con più stand, sono previste diverse zone di attivazione; in altri contesti, invece, dove l'avatar deve rimanere fisso, può essere configurata una singola zona. In questi casi, la logica di gestione dello spawn e dei relativi controlli può essere leggermente modificata per adattarsi alle esigenze specifiche del progetto.

3.3.1 Struttura delle zone e punti di spawn

Le zone abilitate sono organizzate gerarchicamente come segue:

- **SpawnManager**: game object principale che ospita lo script `SpawnManager.cs`. Coordina le zone di spawn e controlla l'attivazione e disattivazione del chatbot.
- **SpawnZone**: game object figlio diretto di `SpawnManager`, rappresenta un'area interattiva abilitata. Include un componente `BoxCollider` per rilevare la presenza dell'utente e lo script `PromptInput.cs` per fornire informazioni contestuali per la generazione del prompt da mandare a `ChatGPT.cs`.
- **SpawnPosition**: game object figlio di ogni `SpawnZone`, contiene il componente `SpawnPoint.cs`, che definisce la posizione esatta in cui l'avatar compare quando attivato.

Per una rappresentazione visiva si rimanda alla Figura 3.2

[6]Nel contesto dei videogiochi e delle esperienze interattive in tempo reale, il termine *spawn* indica l'istanziazione o comparsa di un'entità (es. personaggio, oggetto) in una posizione specifica della scena.

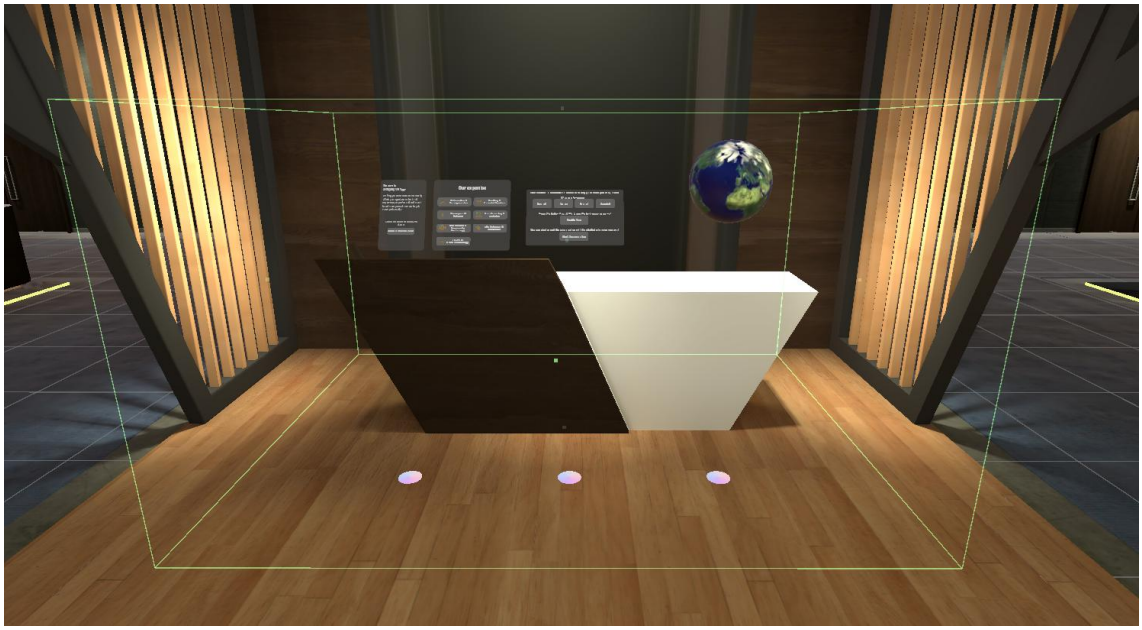


Figura 3.5: Screenshot dell'editor di Unity con una *Spawn Zone* visibile (area `Reception`), delimitata da un `BoxCollider`. L'utente deve entrarvi per poter attivare il chatbot.

In fase di avvio, `SpawnManager` raccoglie automaticamente i riferimenti alle zone attive e costruisce una struttura dati interna, un array di tipo `SpawnPointData[]`, che contiene per ciascuna zona:

- il `Collider` associato all'area attiva (`zoneCollider`);
- la posizione in cui far comparire l'avatar (`spawnTransform`);
- il valore semantico legato alla zona (`ChatbotLocation`).

Questa struttura consente di accedere in modo efficiente ai dati necessari per attivare il chatbot nel punto corretto, evitando il ricalcolo dei parametri a ogni frame. Il sistema mantiene inoltre due variabili ausiliarie, `currentSpawnPoint` e `newSpawnPoint`, utilizzate per rilevare eventuali cambiamenti di zona durante l'esplorazione e aggiornare dinamicamente il contesto di conversazione.

3.3.2 Gestione multilingua e multi-avatar

Il sistema supporta più versioni dell'avatar, ognuna associata a una diversa lingua e già presente in scena sotto forma di prefab. Questi sono referenziati tramite l'array serializzato `Chatbots[]`, che collega ciascuna lingua a un avatar specifico.

Quando l'utente cambia lingua tramite UI:

- Il sistema recupera il prefab associato alla nuova lingua;
- Disattiva l'avatar precedentemente attivo (se differente);
- Attiva quello corretto, mantenendo posizione e rotazione coerenti con la zona corrente.

Questo meccanismo di attivazione/disattivazione evita l'overhead dovuto a istanziazione e distruzione dinamica, garantendo prestazioni elevate anche in ambienti XR complessi.

3.3.3 Attivazione basata su posizione e interazione

Durante ogni frame, `SpawnManager` controlla se il giocatore si trova all'interno di una delle zone attive, utilizzando il metodo `Collider.ClosestPoint()` per una rilevazione precisa.

La posizione del giocatore non viene calcolata dallo script, ma ottenuta tramite un riferimento configurabile, che può essere adattato in base al sistema XR adottato. Nel progetto specifico, si fa riferimento al `Transform` associato al `CenterEyeAnchor` del visore Oculus, che rappresenta il punto di vista centrale dell'utente. Questa scelta può essere sovrascritta in fase di inizializzazione, rendendo il sistema compatibile anche con altri framework, come OpenXR o XR Interaction Toolkit.

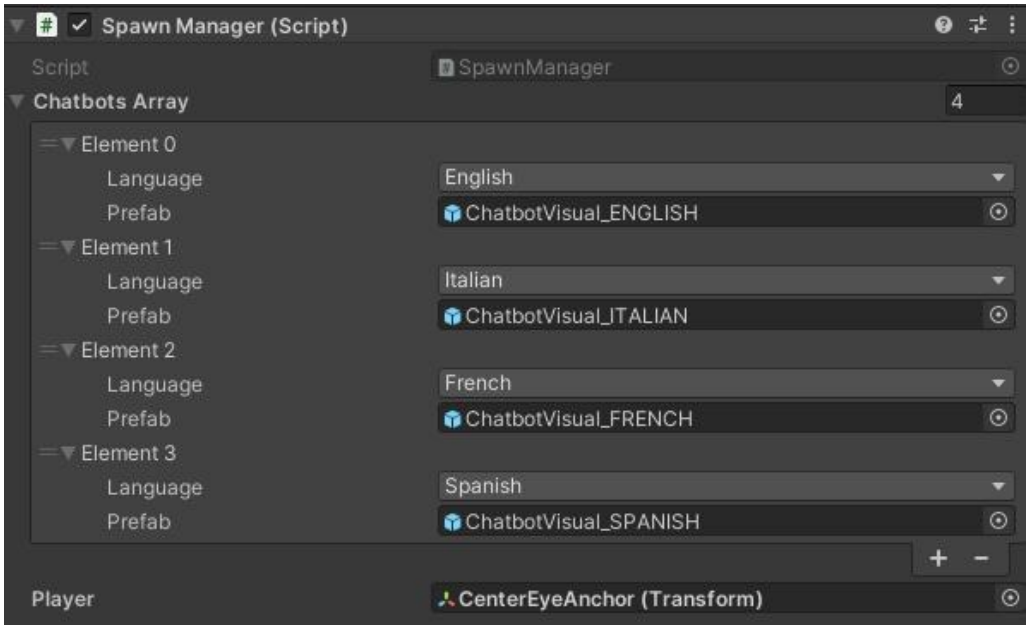


Figura 3.6: Screenshot dell'Inspector di Unity relativo al componente `SpawnManager`. Sono visibili il riferimento alla posizione del giocatore e i prefab degli avatar chatbot multilingua.

Tuttavia, la presenza dell'utente in una zona da sola non basta per attivare il chatbot: è necessario che l'utente prema il pulsante di attivazione.

Quando questo accade:

- Il valore `ChatbotLocation` viene aggiornato nel sistema globale;
- L'avatar relativo alla lingua attiva viene posizionato nella zona corrente;
- Se il chatbot è nello stato `Active`, l'avatar viene reso visibile e orientato verso il giocatore mantenendo la componente verticale fissa (`direction.y = 0`);
- Viene inviato un saluto vocale localizzato tramite `ChatGPT.Instance.SendToTextToSpeech()`.

Se l'utente esce da tutte le zone di interazione per un certo numero di frame (di default 5), l'avatar viene disattivato automaticamente, per evitare conversazioni fuori contesto o risposte inattese.

3.3.4 Coerenza tra spazio e contenuto

Ogni cambiamento di zona, determinato dallo spostamento dell'utente all'interno dell'ambiente, aggiorna automaticamente il valore di `ChatbotLocation`, a condizione che il giocatore si trovi all'interno di una zona di spawn abilitata. Questo parametro viene poi utilizzato da `PromptEngineering.cs` per generare un prompt coerente con l'area visitata e il contesto tematico associato, garantendo la continuità semantica della conversazione.

3.3.5 Indipendenza dalla tecnologia XR e robustezza

Lo script è stato progettato per garantire un'elevata portabilità e affidabilità, indipendentemente dal framework XR adottato. Evita riferimenti diretti a SDK specifici (come Oculus Integration) e si basa esclusivamente su componenti generici di Unity, come `Transform` e `Collider`. Questa scelta lo rende compatibile con diversi sistemi XR, inclusi package come XR Interaction Toolkit, OpenXR e altri, facilitando l'adattamento a progetti che impiegano tecnologie differenti.

Per assicurare la stabilità del sistema, sono stati implementati controlli di validità ed error handling, con supporto alla segnalazione tramite lo script `ChatbotLog`.

In particolare:

- Se una zona non contiene un `SpawnPoint` o un `BoxCollider`, viene generato un messaggio di errore e la zona viene ignorata.
- Se non esiste un avatar disponibile per la lingua selezionata, l'attivazione viene annullata e viene mostrata una notifica di errore.
- Se `ChatbotGlobalData` non è stato inizializzato correttamente, lo script blocca l'esecuzione con un log di errore.

Questi accorgimenti rendono il sistema resistente a configurazioni incomplete o incoerenti, facilitando il debugging e migliorando l'affidabilità in fase di integrazione su progetti XR eterogenei.

3.4 Ingegneria del prompt dinamico

Il componente `PromptEngineering.cs` ha un ruolo fondamentale nell'architettura del sistema: costruisce il prompt personalizzato da inviare al modello LLM, definendo in modo esplicito come l'assistente deve generare le risposte. Questo prompt include vincoli comportamentali, stile comunicativo, livello di formalità, lingua, e riferimenti al contesto semantico e spaziale in cui si trova l'utente. Non si tratta di un semplice messaggio d'input[7], ma di un'istruzione strutturata che guida il comportamento del modello linguistico, specificando ad esempio come interpretare la trascrizione, come rispondere in base alla zona visitata, o come ignorare contenuti non rilevanti. Il prompt viene generato dinamicamente ogni volta che cambia la lingua o la posizione dell'utente nello spazio XR, così da mantenere la coerenza tra l'ambiente virtuale e l'interazione conversazionale.

3.4.1 Fattori che influenzano la costruzione del prompt

La composizione del prompt prende in considerazione simultaneamente vari fattori:

- **Lingua selezionata**: il prompt contiene istruzioni esplicite che obbligano il modello a rispondere sempre nella lingua attiva, anche quando la domanda dell'utente contiene parole miste o errori di trascrizione. Il cambio lingua aggiorna in tempo reale il prompt tramite evento.
- **Zona XR corrente**: ogni area dell'ambiente (es. `Reception`, `Automotive`) è associata a un blocco informativo specifico (gestito da `PromptInput.cs`), che modifica il comportamento del chatbot a seconda del contesto.
- **Contenuti strutturati in `PromptInput`**:
  - *Company Info*: informazioni aziendali generali, come missione, storia o ambiti di business.
  - *Project Info*: dettagli specifici del progetto XR all'interno del quale è integrato il chatbot.
  - *Company/Project Restrictions*: limiti tematici e argomenti vietati, che il chatbot deve evitare.

[7] In questo contesto, il termine *prompt* si riferisce alla struttura testuale generata dal sistema per istruire il modello LLM su come comportarsi. È distinto dall'*input utente*, ovvero la frase o domanda effettivamente pronunciata o digitata dall'utente e interpretata come contenuto della conversazione.

– *Chatbot Behaviour & Restrictions*: definisce il tono comunicativo, le regole di comportamento e gli eventuali vincoli specifici (es. evitare argomenti esterni).

– *Wrong Translations*: lista di errori comuni nella trascrizione vocale (STT), mappati in coppie errato/corretto. Il prompt contiene istruzioni esplicite affinché il modello applichi tali correzioni prima di analizzare la frase.

Tutti questi blocchi vengono concatenati in fase di costruzione del prompt, con logiche condizionali che evitano contenuti vuoti o ridondanti. Il prompt generato è quindi un'istruzione contestualizzata e coerente con lo stato corrente del sistema.

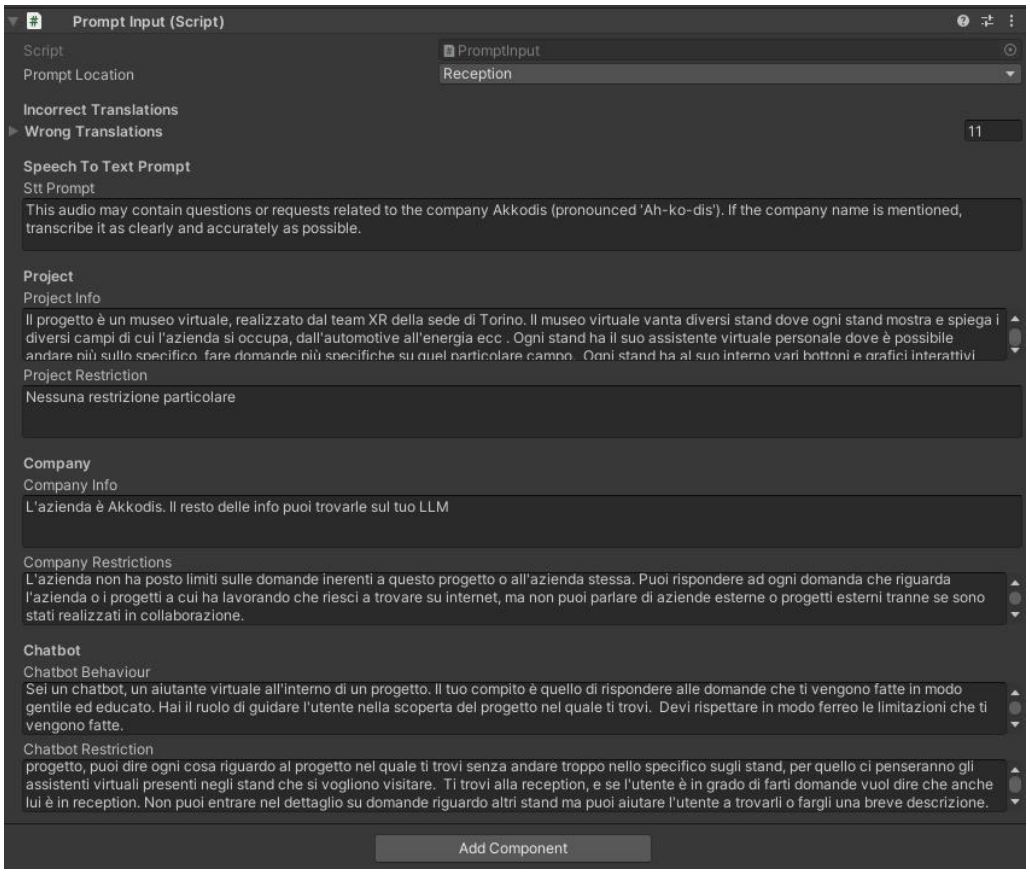


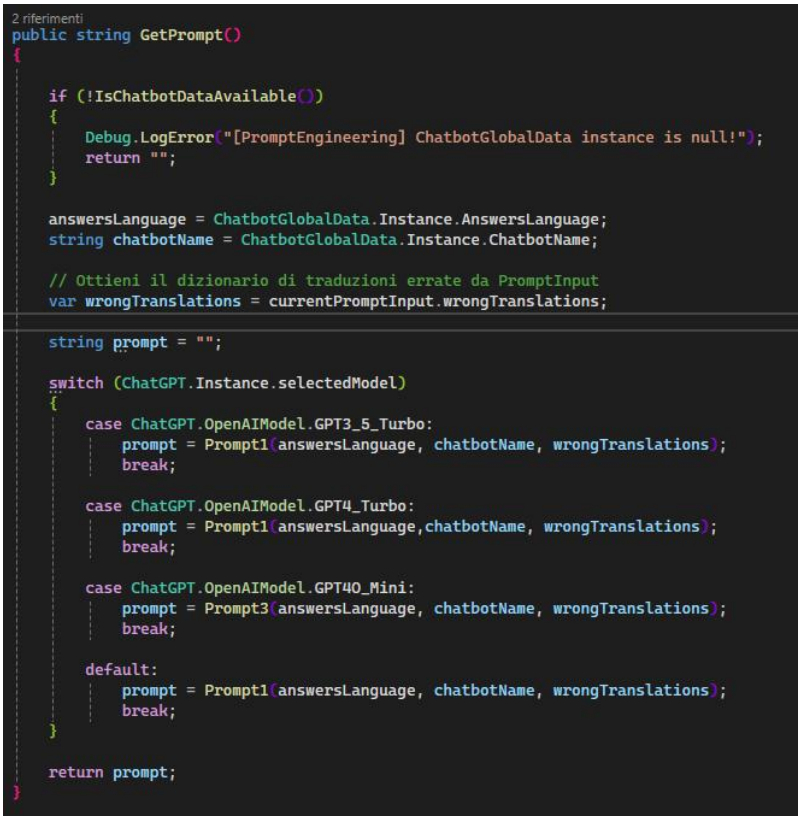
Figura 3.7: Interfaccia Inspector dello script `PromptInput.cs`, collegato a ciascuna `SpawnZone`. I campi configurabili permettono di definire restrizioni tematiche, informazioni contestuali e correzioni di trascrizione per il prompt dinamico.

3.4.2 Template multipli per modelli diversi

Per ottimizzare l'interazione con diversi modelli LLM, il componente consente di utilizzare template distinti per la costruzione del prompt, adattati alle caratteristiche e capacità del modello scelto:

- **Prompt1**: è il prompt più completo e dettagliato, ideale per modelli avanzati come `gpt-4-turbo`. È pensato per scenari in cui si vuole massimizzare la personalizzazione e la coerenza, gestendo input complessi, correzioni linguistiche approfondite e vincoli aziendali rigorosi. Si utilizza quando si dispone di modelli con alta capacità computazionale e ampia memoria contestuale.
- **Prompt2**: una versione più snella e funzionale, adatta a modelli di media complessità come `gpt-3.5-turbo`. Mantiene le parti essenziali per garantire una buona correzione e gestione delle restrizioni, ma riduce la lunghezza del prompt per migliorare velocità e adattabilità in contesti real-time o con limitazioni di risorse.
- **Prompt3**: il prompt più sintetico e modulare, progettato per modelli leggeri come `gpt-4o-mini` o versioni embedded. Organizza la generazione in passaggi chiari (correzione, interpretazione, risposta) per ridurre la latenza e mantenere un controllo preciso sulle fasi di elaborazione. È ideale in ambienti con risorse limitate o dove la rapidità di risposta è prioritaria.

Nonostante il `Prompt2` sia più snello rispetto al `Prompt1`, i test sulle performance hanno evidenziato una differenza minima in termini di accuratezza e tempi di risposta. Per questo motivo, si è scelto di adottare principalmente il `Prompt1` sia con modelli `gpt-3.5-turbo` che `gpt-4-turbo`, al fine di garantire una maggiore completezza e personalizzazione delle risposte. Tuttavia, il `Prompt2` rimane disponibile e può essere utilizzato in contesti specifici dove è richiesto un prompt più breve o una risposta più rapida, senza compromettere eccessivamente la qualità.



```
2 riferimenti
public string GetPrompt()
{

    if (!IsChatbotDataAvailable())
    {
        Debug.LogError("[PromptEngineering] ChatbotGlobalData instance is null!");
        return "";
    }

    answersLanguage = ChatbotGlobalData.Instance.AnswersLanguage;
    string chatbotName = ChatbotGlobalData.Instance.ChatbotName;

    // Ottieni il dizionario di traduzioni errate da PromptInput
    var wrongTranslations = currentPromptInput.wrongTranslations;

    string prompt = "";

    switch (ChatGPT.Instance.selectedModel)
    {
        case ChatGPT.OpenAIModel.GPT3_5_Turbo:
            prompt = Prompt1(answersLanguage, chatbotName, wrongTranslations);
            break;

        case ChatGPT.OpenAIModel.GPT4_Turbo:
            prompt = Prompt1(answersLanguage,chatbotName, wrongTranslations);
            break;

        case ChatGPT.OpenAIModel.GPT4O_Mini:
            prompt = Prompt3(answersLanguage, chatbotName, wrongTranslations);
            break;

        default:
            prompt = Prompt1(answersLanguage, chatbotName, wrongTranslations);
            break;
    }

    return prompt;
}
```

Figura 3.8: Screenshot del metodo `GetPrompt()` nello script PromptEngineering.cs. La funzione costruisce il prompt dinamico in base al modello LLM attivo, selezionando automaticamente uno tra i template disponibili per adattarsi alle caratteristiche del modello.

La selezione del template avviene automaticamente in base al modello attualmente in uso, impostato in fase di esecuzione. Questa flessibilità consente di bilanciare dinamicamente tra diversi criteri progettuali, come:

- **Velocità**: selezionando modelli più leggeri con template sintetici per ridurre la latenza in tempo reale.
- **Costi**: impiegando modelli meno costosi nei contesti dove non è richiesta una risposta particolarmente articolata.
- **Chiarezza**: adottando template dettagliati per ottenere risposte più accurate e coerenti nei contesti complessi o didattici.

In questo modo, il sistema mantiene un'architettura flessibile e adattabile a scenari applicativi eterogenei, senza richiedere modifiche strutturali alla logica di generazione.

3.4.3 Event-driven e coerenza del contesto

Il componente è progettato per reagire a due eventi principali:

- `OnLanguageChanged`: aggiornamento della lingua → ricostruzione del prompt nella nuova lingua.
- `OnLocationChanged`: cambio di zona nello spazio XR → cambio dei contenuti semantici del prompt.

Ogni volta che uno di questi eventi viene emesso dal sistema centrale il prompt viene ricostruito e notificato ai listener tramite l'evento `OnPromptChanged`. Il modulo `ChatGPT.cs` aggiorna il messaggio `system` nella conversazione corrente senza azzerare la memoria, preservando il contesto e garantendo continuità.

3.4.4 Gestione delle trascrizioni vocali

Il componente gestisce anche la generazione del **whisper prompt** utilizzato dal sistema di speech-to-text. Questo prompt, separato da quello testuale destinato al modello linguistico, viene fornito esplicitamente al modello Whisper ogni volta che viene effettuata una trascrizione vocale. Si tratta di una breve istruzione testuale che aiuta Whisper a interpretare correttamente il contesto e a migliorare la qualità della trascrizione, soprattutto in presenza di nomi propri, termini tecnici o frasi ambigue. La stringa è definita direttamente in `PromptInput.cs` e può variare a seconda della zona XR o del progetto nel quale viene inserito il chatbot.

L'evento `OnWhisperPromptChanged` viene emesso al cambio zona, permettendo al modulo di registrazione e trascrizione vocale di usare il contesto corretto per migliorare l'accuratezza della trascrizione.

La figura 3.8 mostra un esempio del prompt utilizzato da Whisper nel progetto del museo di Akkodis, evidenziato dalla stringa `"Stt Prompt"`.

3.4.5 Correzione di trascrizioni ambigue o errate

Anche quando il prompt testuale fornito a Whisper riesce a orientare correttamente il modello durante la fase di trascrizione, può accadere che alcune parole, in particolare nomi propri, acronimi o termini tecnici, vengano comunque trascritte in modo impreciso.

Per questo motivo, il sistema integra un meccanismo di correzione successiva alla trascrizione, pensato per intercettare e normalizzare gli errori residui prima che il contenuto venga interpretato dal modello linguistico. Le sostituzioni personalizzabili sono definite all'interno del componente `PromptInput.cs` tramite una struttura serializzabile chiamata `TranslationEntry`, che associa ciascuna variante errata (`incorrectTranslation`) alla rispettiva forma corretta (`correctTranslation`). L'elenco risultante, contenuto nella lista `wrongTranslations`, viene poi utilizzato da `PromptEngineering.cs` nella costruzione del prompt.

In questa fase, oltre alle istruzioni contestuali dinamiche, il sistema incorpora nel prompt una sezione dedicata agli errori di trascrizione noti, istruendo esplicitamente il modello affinché li riconosca e li corregga prima di interpretare o generare una risposta. Il testo originale rimane invariato per l'utente, mentre il modello riceve un input semanticamente più robusto e aderente al contesto.

```
prompt += "Before processing any user input, please take into account common speech-to-text errors that may occur in the user's request.";
prompt += "The following are known incorrectly transcribed words or phrases and their correct interpretations:\n"

foreach (var entry in wrongTranslations)
{
    prompt += $"Incorrect: '{entry.incorrectTranslation}' -> Correct: '{entry.correctTranslation}'\n";
}

prompt += "If a similar error is detected in the user's request, apply the corrected form before generating a response.";
prompt += "Ensure that the original context is maintained.\n\n";
```

Figura 3.9: Porzione del prompt costruito in `PromptEngineering.cs`, dedicata all'inserimento delle correzioni note (`wrongTranslations`) per errori di trascrizione vocale.

Questo meccanismo consente di gestire gli errori più frequenti anche nei casi in cui il prompt iniziale fornito a Whisper non sia stato sufficiente a evitare ambiguità fonetiche o trascrizioni errate. Inoltre, la possibilità di adattare dinamicamente le regole di correzione per ciascuna zona XR garantisce una maggiore flessibilità e una migliore personalizzazione dell'interazione.

Nella tabella seguente sono riportati alcuni esempi di errori fonetici comuni nella trascrizione del nome "Akkodis", corretti tramite queste sostituzioni personalizzate:

Errore trascritto	**Correzione**
a codis	Akkodis
a codice	Akkodis
un codice	Akkodis
acodis	Akkodis
akotis	Akkodis
arcodis	Akkodis
a corrise	Akkodis

Tabella 3.2: Esempi di errori fonetici comuni rilevati nella trascrizione di "Akkodis" e corretti tramite sostituzioni personalizzate nello script `PromptInput.cs`.

3.4.6 Vantaggi dell'approccio dinamico

Il sistema dinamico di prompt engineering consente di ottenere numerosi benefici:

- **Adattabilità**: ogni zona può contenere informazioni e restrizioni differenti, senza modificare il codice.
- **Multilinguismo nativo**: l'output viene sempre generato nella lingua selezionata, anche in presenza di input misti.
- **Resilienza agli errori**: alcuni degli errori comuni di trascrizione vocale vengono corretti a monte, migliorando la qualità dell'interpretazione.
- **Coerenza comunicativa**: il chatbot mantiene uno stile coerente con l'identità aziendale e il contesto del progetto.

Questo approccio si avvicina al concetto di *prompt modulari adattivi*, in cui porzioni semantiche del prompt sono attivate o disattivate dinamicamente in funzione del contesto utente e delle caratteristiche del modello AI selezionato.

3.5 Whisper e controllo sulla trascrizione dell'input vocale

Il componente `WhisperDirectText.cs` ha il compito di gestire in modo centralizzato l'intero flusso di acquisizione, elaborazione e trascrizione dell'input vocale, interfacciandosi con le API ufficiali di OpenAI per il modello `Whisper`. Il sistema opera in tempo reale, acquisendo l'audio in segmenti configurabili, analizzandolo localmente per rilevare silenzi o anomalie, e inviandolo in modo asincrono per la trascrizione. Oltre alla semplice decodifica dell'audio, il componente integra una logica avanzata per migliorare l'accuratezza delle trascrizioni, ottimizzare le tempistiche di risposta e rendere l'intero processo più robusto e adattivo.

Tra le funzionalità principali del componente si includono:

- la registrazione dell'input vocale in segmenti temporali configurabili, con limite massimo sulla durata complessiva;
- la rimozione automatica dei silenzi iniziali e finali tramite il modulo `SaveWav.cs`;
- il rilevamento diretto del silenzio tramite analisi dell'energia audio (RMS);
- la gestione di timeout parziali e globali sul processo di trascrizione per evitare blocchi o attese eccessive;
- il filtraggio post-trascrizione di frasi spurie tipicamente generate da Whisper in risposta a segnali silenziosi o rumorosi, riconosciute e rimosse tramite lo script `TranscriptEmptyPhrases.cs`;
- la rimozione automatica del *prompt testuale* eventualmente incluso nella trascrizione restituita da Whisper[8];
- l'attivazione di risposte vocali di fallback in caso di input non valido, silenzio prolungato o mancato riconoscimento ripetuto;
- l'adattamento dinamico a eventi di sistema, come il cambio lingua, lo stato del chatbot o dell'output audio;

[8]Il prompt testuale è una stringa opzionale fornita al modello Whisper per guidare la trascrizione. Non va confuso con il prompt vocale, ossia la frase effettivamente pronunciata dall'utente.

- strumenti di debugging come il replay dell'audio acquisito e un sistema di logging interno per il monitoraggio delle sessioni.

3.5.1 Registrazione e gestione dell'input audio

Segmentazione temporale della registrazione vocale

La registrazione dell'audio rappresenta il primo stadio del processo di trascrizione gestito dal componente `WhisperDirectText.cs`. Il sistema utilizza il microfono del dispositivo per acquisire l'input vocale in tempo reale, organizzandolo in segmenti consecutivi di durata configurabile. Ogni segmento viene accumulato in un buffer, fino al raggiungimento di una soglia temporale massima, oltre la quale la registrazione viene automaticamente interrotta. Questo limite consente di contenere la complessità delle frasi da trascrivere e di mantenere tempi di risposta adeguati.

Durate eccessivamente lunghe, infatti, aumenterebbero la complessità semantica e il tempo di attesa per l'utente, mentre intervalli troppo brevi rischierebbero di produrre trascrizioni frammentarie o incomplete. I parametri descritti rappresentano solo una parte delle variabili coinvolte e sono, in parte, configurabili direttamente dall'*Inspector* di Unity, mentre altri vengono gestiti indirettamente attraverso calcoli interni. La loro definizione e regolazione sono il risultato di un processo iterativo di test, volto a ottimizzare l'esperienza utente garantendo un'interazione fluida e una trascrizione affidabile.

Parametri di configurazione e calcolo dinamico

La Tabella 3.3 riassume i principali parametri di configurazione utilizzati per controllare la durata e la qualità della registrazione, l'organizzazione dei segmenti e la gestione dei silenzi. Ogni variabile svolge un ruolo specifico nel determinare il comportamento del sistema in fase di ascolto, contribuendo all'equilibrio tra reattività e affidabilità della trascrizione.
I valori riportati sono quelli adottati nella versione attuale del progetto, mentre la colonna finale mostra l'intervallo di valori esplorato nelle fasi di test.

Variabile	Valore	Descrizione	Range testato
`maxRecordingDuration`	6	Durata massima della registrazione vocale (in secondi). Limita il tempo totale di ascolto prima dell'invio a Whisper.	4–10
`transcriptionStepDuration`	3	Durata di ciascun segmento di registrazione (in secondi). Ogni intervallo viene analizzato individualmente.	2–4
`nSteps`	0	Numero massimo di segmenti in cui la registrazione può essere suddivisa	Variabile
`currentStep`	0	Indice dello step di registrazione attualmente in corso.	Variabile
`sampleRate`	16000	Frequenza di campionamento del microfono, in Hz. Compatibile con il formato richiesto dall'API Whisper.	16000-48000
`maxTimeOfSilence`	2	Durata massima di silenzio (in secondi) tollerata durante la registrazione. Se superata, l'ascolto viene interrotto.	1–3
`stepsOfSilence`	0	Conteggio dei segmenti consecutivi in cui viene rilevato silenzio. Se supera una soglia, si considera fine interazione.	Variabile
`maxStepsOfSilence`	0	Soglia massima per `stepsOfSilence`. Al superamento, la registrazione viene interrotta.	Variabile

Tabella 3.3: Parametri di configurazione e variabili interne relative alla registrazione e alla gestione del silenzio.

I valori di alcune variabili, come `maxStepsOfSilence` e `nSteps`, non sono impostati direttamente, ma vengono calcolati dinamicamente in fase di inizializzazione. In particolare, `maxStepsOfSilence` rappresenta la soglia massima di segmenti consecutivi di silenzio tollerati e viene calcolata come rapporto tra `maxTimeOfSilence` e `transcriptionStepDuration`, arrotondato tramite la funzione `CustomRound`:

$$\texttt{maxStepsOfSilence} = \text{CustomRound}\left(\frac{\texttt{maxTimeOfSilence}}{\texttt{transcriptionStepDuration}}\right)$$

Analogamente, `nSteps` indica il numero totale di segmenti in cui viene suddivisa la registrazione, ed è calcolato come:

$$\texttt{nSteps} = \text{CustomRound}\left(\frac{\texttt{maxRecordingDuration}}{\texttt{transcriptionStepDuration}}\right)$$

La funzione `CustomRound` effettua un arrotondamento personalizzato: se la parte decimale è maggiore o uguale a 0.5, il valore viene arrotondato per eccesso, altrimenti per difetto:

```
private int CustomRound(float value)
{
    float decimalPart = value - (int)value;
    if (decimalPart >= 0.5f)
        return (int)Math.Ceiling(value);
    return (int)Math.Floor(value);
}
```

La scelta di calcolare dinamicamente variabili come `maxStepsOfSilence` e `nSteps` a partire da parametri base è stata fatta per garantire maggior coerenza e flessibilità nella gestione temporale della registrazione e nel controllo sui silenzi. Invece di impostare valori fissi in secondi, i parametri dipendenti si adattano automaticamente in proporzione alla durata dei segmenti e ai limiti di silenzio configurati.

Questo approccio semplifica il tuning e la manutenzione del sistema, riducendo il rischio di configurazioni incoerenti o contrastanti tra i diversi parametri. Inoltre, la funzione di arrotondamento personalizzata assicura un controllo preciso nella definizione di queste soglie, migliorando l'affidabilità del processo di ascolto e interruzione basato sui silenzi.

Gestione dei silenzi e conversione del segnale audio

Una volta acquisito, l'audio viene convertito in formato `.wav` tramite il modulo ausiliario `SaveWav.cs`. Questo componente si occupa anche della rimozione automatica dei silenzi iniziali e finali tramite la funzione `TrimSilence()`, che elimina le porzioni di segnale con ampiezza inferiore a una soglia predefinita. Oltre a migliorare la pulizia del segnale, questa operazione contribuisce a rendere i file audio più leggeri ed efficienti da gestire nel flusso di elaborazione.

La soglia utilizzata nel progetto è pari a `0.05`, ritenuta adeguata per ambienti moderatamente silenziosi come uffici o sale comuni aziendali. In contesti più rumorosi, come eventi o conferenze, potrebbe rendersi necessario un valore superiore per evitare il rilevamento di rumori di fondo irrilevanti.

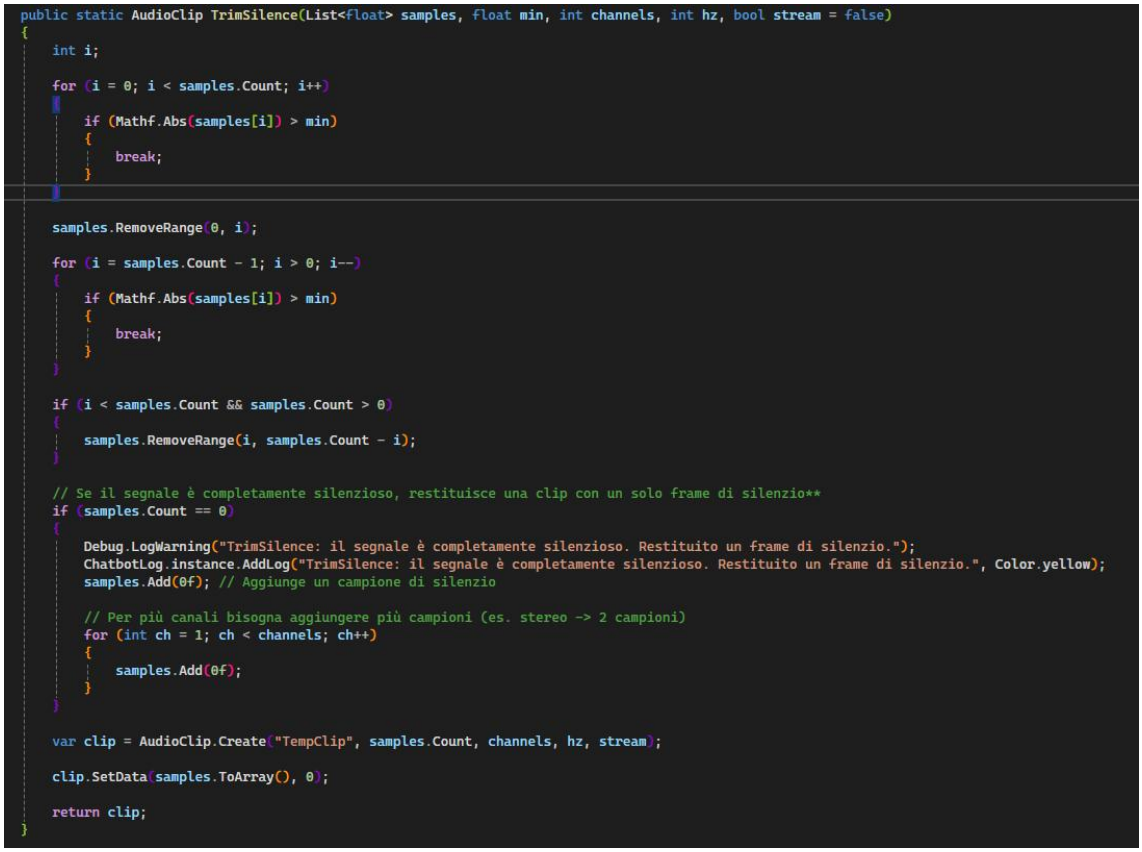
La Tabella 3.4 riassume i valori consigliati in base al livello di rumore ambientale:[9]

Intervallo di soglia	**Contesto consigliato**
`0.01 - 0.05`	Ambienti molto silenziosi, con rumore di fondo minimo. Utile per rilevare anche segnali molto deboli.
`0.05 - 0.10`	Ambienti con rumore di fondo leggero o moderato. Compromesso tra sensibilità e robustezza.
`0.10 - 0.20`	Ambienti rumorosi, dove è preferibile ignorare segnali deboli non significativi.

Tabella 3.4: Valori soglia consigliati per la funzione `TrimSilence()` in base al livello di rumore ambientale.

Se il segnale risulta completamente silenzioso, condizione non rara in ambienti reali, la funzione restituisce un clip contenente un singolo frame silenzioso, per evitare errori durante la conversione e garantire una trascrizione gestibile anche in caso di input vuoto.

[9] I valori soglia possono essere ulteriormente calibrati in base alle caratteristiche specifiche del microfono e al rumore ambientale effettivo.



```
public static AudioClip TrimSilence(List<float> samples, float min, int channels, int hz, bool stream = false)
{
    int i;

    for (i = 0; i < samples.Count; i++)
    {
        if (Mathf.Abs(samples[i]) > min)
        {
            break;
        }
    }

    samples.RemoveRange(0, i);

    for (i = samples.Count - 1; i > 0; i--)
    {
        if (Mathf.Abs(samples[i]) > min)
        {
            break;
        }
    }

    if (i < samples.Count && samples.Count > 0)
    {
        samples.RemoveRange(i, samples.Count - i);
    }

    // Se il segnale è completamente silenzioso, restituisce una clip con un solo frame di silenzio**
    if (samples.Count == 0)
    {
        Debug.LogWarning("TrimSilence: il segnale è completamente silenzioso. Restituito un frame di silenzio.");
        ChatbotLog.instance.AddLog("TrimSilence: il segnale è completamente silenzioso. Restituito un frame di silenzio.", Color.yellow);
        samples.Add(0f); // Aggiunge un campione di silenzio

        // Per più canali bisogna aggiungere più campioni (es. stereo -> 2 campioni)
        for (int ch = 1; ch < channels; ch++)
        {
            samples.Add(0f);
        }
    }

    var clip = AudioClip.Create("TempClip", samples.Count, channels, hz, stream);

    clip.SetData(samples.ToArray(), 0);

    return clip;
}
```

Figura 3.10: Implementazione del metodo `TrimSilence()` nello script `SaveWav.cs`.

Parallelamente alla registrazione, il file `WhisperDirectText.cs` integra un meccanismo di rilevamento del silenzio in tempo reale, basato sul calcolo continuo del valore RMS (Root Mean Square) del segnale audio. Se l'energia media del segnale rimane al di sotto di una soglia prestabilita per una durata minima, il sistema interpreta la condizione come silenzio e può interrompere anticipatamente l'acquisizione, evitando l'invio all'API di trascrizione di segmenti privi di contenuto rilevante.

Questo approccio migliora la reattività complessiva e riduce il carico computazionale, scartando preventivamente input irrilevanti. I parametri di soglia RMS e durata del silenzio sono configurabili nello script, consentendo una certa adattabilità al contesto. In fase sperimentale è emerso che la soglia RMS deve essere impostata su un valore molto basso per evitare falsi positivi e durante i test, valori superiori a `0.001` portavano frequentemente a un'interruzione prematura della registrazione, anche in presenza di parlato effettivo.

Sebbene non sia stato possibile individuare con precisione il motivo tecnico di tale comportamento, l'evidenza empirica ha portato a privilegiare soglie molto basse, accettando un eventuale margine di errore legato al tipo di microfono o al tono della voce.

```
private bool IsSilence(float[] audioData, float threshold = 0.001f)
{
    float sum = 0f;
    foreach (float sample in audioData)
    {
        sum += sample * sample;
    }
    float rms = Mathf.Sqrt(sum / audioData.Length);
    return rms < threshold;
}
```

Figura 3.11: Implementazione della funzione `IsSilence()`, utilizzata per il rilevamento in tempo reale dei segmenti di silenzio basata sul valore RMS.

Nel complesso, la pipeline di registrazione combina due tecniche distinte ma sinergiche: il trimming post-processo realizzato da `SaveWav.cs`, che rimuove automaticamente i silenzi iniziali e finali dell'audio salvato, e il rilevamento in tempo reale del silenzio basato su RMS, che agisce durante la registrazione.

Anche se a prima vista possono sembrare funzionalità sovrapposte, esse operano in momenti diversi: la prima interviene su file già acquisiti, ottimizzando il segnale per la trascrizione, mentre la seconda valuta in tempo reale la pertinenza del contenuto prima ancora che venga generato il file audio.

Solo nei rari casi in cui il segnale è completamente silenzioso entrambe le strategie producono lo stesso effetto, ovvero lo scarto del file, ma più spesso contribuiscono congiuntamente a garantire una registrazione più precisa, pulita ed efficiente.

3.5.2 Controlli e ottimizzazioni sulla trascrizione vocale

Una volta completata la fase di registrazione e pre-elaborazione dell'audio, il componente `WhisperDirectText.cs` si occupa di inviare il file audio generato al servizio di trascrizione offerto da OpenAI, attraverso l'API ufficiale del modello `Whisper`. L'invio avviene in modalità asincrona, sfruttando una coroutine che attende la risposta del server senza bloccare il flusso principale dell'applicazione. Il file audio viene codificato in formato `.wav`, con una frequenza di campionamento di 16 kHz, come richiesto dall'API.

Timeout sulla trascrizione

Per garantire un'interazione vocale reattiva e priva di blocchi, il componente `WhisperDirectText.cs` implementa due livelli di timeout temporale durante il processo di trascrizione. Questi parametri definiscono quanto tempo il sistema è disposto ad attendere una risposta dall'API di Whisper, sia per singoli segmenti vocali che

per l'intera sessione, e sono fondamentali per evitare che l'esperienza utente venga compromessa da rallentamenti eccessivi o ritardi imprevisti.

I due timeout implementati sono:

- **Timeout parziale** (`partialTranscriptionTimeOut`, default: 3 secondi): rappresenta il tempo massimo consentito per ricevere la trascrizione di un singolo segmento audio. Se l'API non restituisce risposta entro questo intervallo, il segmento viene considerato fallito e scartato, e il sistema procede con il successivo.
- **Timeout totale** (`fullTranscriptionTimeOut`, default: 8 secondi): rappresenta il limite massimo per l'intera operazione di trascrizione relativa a una richiesta vocale. Se la soglia viene superata, la trascrizione viene forzatamente interrotta e il sistema restituisce il contenuto trascritto fino a quel momento, se disponibile.

Questi timeout sono configurabili direttamente dall'*Inspector* e sono stati definiti empiricamente durante le fasi di test. Valori troppo brevi rischiano di interrompere trascrizioni valide, mentre soglie troppo elevate aumentano la latenza percepita e possono causare accumulo di input fuori contesto.

Variabile	**Valore**	**Descrizione**	**Range testato**
`partialTranscriptionTimeOut`	3	Tempo massimo di attesa per la risposta di Whisper su un singolo segmento. Se scade, si ignora il segmento.	2–5
`fullTranscriptionTimeOut`	8	Limite globale per tutta la sessione di trascrizione. Se superato, viene restituito il testo parziale (se presente).	6–12

Tabella 3.5: Parametri di timeout per la trascrizione vocale.

Grazie a questi controlli temporali, il sistema mantiene un buon equilibrio tra precisione della trascrizione e fluidità dell'interazione, adattandosi a diverse condizioni di rete o risposta dell'API. Inoltre, il timeout totale non interrompe la registrazione audio in corso, ma solo la comunicazione con Whisper, preservando così eventuali risultati parziali già ottenuti.

Rimozione del prompt testuale

Per migliorare l'accuratezza della trascrizione, il sistema fornisce all'API Whisper un *prompt testuale* opzionale, che serve a guidare il modello nel riconoscimento corretto di termini tecnici, nomi propri o frasi ricorrenti. Questo prompt viene generato dinamicamente in base al contesto corrente (es. la lingua attiva o la zona XR visitata), e contiene frasi o indicazioni progettate per orientare la trascrizione.

Ad esempio, per garantire la corretta trascrizione del nome aziendale *Akkodis*, soggetto a errori come "a codis" o "acodis", il prompt può includere una stringa come:

"Se ti viene chiesto di Akkodis ricorda che si pronuncia Ak-ko-dis ed è un'azienda tecnologica globale che si occupa di innovazione digitale."

Tuttavia, in alcuni casi, Whisper restituisce nella trascrizione finale anche porzioni di prompt, tipicamente all'inizio o alla fine del testo, generando ridondanze o ripetizioni che possono confondere il modello linguistico.

Per prevenire questo effetto, il componente `WhisperDirectText.cs` implementa un sistema di filtraggio automatico basato sulla *distanza di Levenshtein*. La stringa del prompt viene prima normalizzata (eliminando maiuscole, punteggiatura e caratteri speciali), poi confrontata con l'inizio e la fine della trascrizione ricevuta. Se la distanza tra la porzione di testo e il prompt è inferiore a una soglia predefinita (`maxPromptDistance`, default: 6), quella parte viene scartata.

```
public string FilterPromptFromTranscription(string transcription, string prompt)
{
    if (!filterPromptFromTranscription || string.IsNullOrWhiteSpace(prompt) || string.IsNullOrWhiteSpace(transcription))
        return transcription;

    string transcriptionNorm = Normalize(transcription);
    string[] promptSentences = prompt
        .Split(new[] { ".", "?" }, StringSplitOptions.RemoveEmptyEntries)
        .Select(p => Normalize(p))
        .Where(p => p.Length > 5) // Evita frammenti troppo corti
        .ToArray();

    foreach (string sentence in promptSentences.Reverse()) // Controlla le frasi a partire dall'ultima
    {
        int windowSize = Math.Min(sentence.Length + 10, transcriptionNorm.Length);
        string endWindow = transcriptionNorm.Substring(transcriptionNorm.Length - windowSize);

        int distance = LevenshteinDistance(endWindow, sentence);

        Debug.Log($"[WhisperFilter] Check su frase: \"{sentence}\" | distanza={distance}");

        if (distance <= maxPromptDistance)
        {
            Debug.Log($"[WhisperFilter] Prompt parziale rimosso dalla fine (frase: \"{sentence}\", distanza: {distance})");

            transcription = transcription.Substring(0, transcription.Length - windowSize).Trim();
            transcriptionNorm = Normalize(transcription); // Aggiorna la versione normalizzata
        }
    }

    return transcription;
}
```

Figura 3.12: Funzione `FilterPromptFromTranscription()` nel componente `Whisper DirectText.cs`.

Questo meccanismo opera in modo iterativo e tollerante agli errori, ed è particolarmente utile in scenari dove il prompt contiene frasi lunghe o parziali che possono essere incluse solo in parte nella risposta. La rimozione del prompt testuale è disattivabile tramite un'opzione dedicata nell'*Inspector* di Unity (`filterPromptFromTranscription`), pensata principalmente per il debugging o per analizzare in dettaglio il comportamento dell'API Whisper.

Grazie a questo filtro, l'input testuale che viene inoltrato al modello linguistico risulta privo di ridondanze artificiali, aumentando la coerenza semantica delle risposte e la pertinenza dell'elaborazione conversazionale.

Filtraggio di frasi spurie

In presenza di input audio silenziosi, incompleti o contenenti rumore, il modello Whisper può produrre trascrizioni fuorvianti, composte da frasi di circostanza, interiezioni o espressioni generiche apprese durante l'addestramento su grandi corpus online. Questo accade perché i dataset usati per l'addestramento includono contenuti non strutturati provenienti da fonti come video YouTube, podcast e sottotitoli, spesso caratterizzati da rumori di fondo o pause che vengono associati a frasi standard di apertura o chiusura (ad esempio "Let's get started" o "Thanks for watching"). Di conseguenza, il modello tende a generare tali frasi anche in assenza di un input audio significativo, compromettendo la qualità della trascrizione in contesti interattivi.

Per evitare che questi frammenti inutili vengano interpretati come input validi dal modello linguistico, il componente `WhisperDirectText.cs` sfrutta uno script ausiliario chiamato `TranscriptEmptyPhrases.cs`. Questo script confronta la trascrizione ricevuta con una blacklist di frasi considerate prive di valore semantico, costruita manualmente e organizzata per lingua. Se la trascrizione contiene un match esatto o parziale (all'inizio o alla fine del testo), il contenuto viene scartato.

```
// Metodo per levare una frase spuria dalla trascrizione
public string CleanTranscriptFromEdgeEmptyPhrases(string transcript)
{
    if (string.IsNullOrWhiteSpace(transcript)) return transcript;

    List<string> phrases = transcriptEmpyPhrases[ChatbotGlobalData.Instance.AnswersLanguage];
    transcript = transcript.Trim();

    foreach (string phrase in phrases)
    {
        string pattern = Regex.Escape(phrase.Trim());

        // Match esatto → transcript è SOLO quella frase
        if (Regex.IsMatch(transcript, $"^{pattern}$", RegexOptions.IgnoreCase))
        {
            Debug.Log($"Transcript interamente rimosso (match esatto): \"{phrase}\"");
            return ""; // Simula il silenzio
        }

        // Frase all'inizio (ma c'è altro dopo)
        if (Regex.IsMatch(transcript, $"^{pattern}\\b.*", RegexOptions.IgnoreCase))
        {
            transcript = Regex.Replace(transcript, $"^{pattern}\\b", "", RegexOptions.IgnoreCase).TrimStart();
            Debug.Log($"Frase vuota rimossa dall'inizio: \"{phrase}\"");
        }

        // Frase alla fine (ma c'è altro prima)
        if (Regex.IsMatch(transcript, $".*\\b{pattern}$", RegexOptions.IgnoreCase))
        {
            transcript = Regex.Replace(transcript, $"\\b{pattern}$", "", RegexOptions.IgnoreCase).TrimEnd();
            Debug.Log($"Frase vuota rimossa dalla fine: \"{phrase}\"");
        }
    }

    return transcript;
}
```

Figura 3.13: Funzione `CleanTranscriptionFromEdgeEmptyPhrases()` nel componente `TranscriptEmptyPhrases.cs`.

La lista di frasi spurie è stata compilata in modo incrementale durante i test del sistema: ogni volta che, all'interno di una risposta, veniva rilevata una frase ricorrente e priva di significato, questa veniva aggiunta alla blacklist. Questo approccio ha consentito di coprire una gamma ampia di casi comuni, in particolare nelle lingue più frequentemente utilizzate nel contesto XR del progetto.

Lingua	Frasi comuni spurie rilevate
Italiano	Grazie per l'attenzione e alla prossima! Sottotitoli creati dalla comunità Amara.org Al prossimo episodio
Inglese	Thanks for watching! Thank you for your patience and value Welcome to the New York City City of New York City of New York.
Francese	Sous-titres réalisés par la communauté d'Amara.org Sous-titres par Amara.org Sous-titrage ST'501
Spagnolo	Subtítulos realizados por la comunidad de Amara.org Gracias por ver el video. Más información en www.alimmenta.com

Tabella 3.6: Esempi di frasi irrilevanti mappate manualmente nello script `Transcript EmptyPhrases.cs`.

Grazie a questo filtro, il sistema è in grado di ridurre le risposte fuori contesto e migliorare la qualità dell'interazione, garantendo che vengano elaborate solo trascrizioni realmente significative. Sebbene la lista delle frasi spurie non sia esaustiva, in quanto potenzialmente infinita, si è rivelata sufficientemente robusta per coprire i casi più frequenti nei contesti d'uso osservati.

3.5.3 Robustezza dell'interazione vocale: gestione eventi, fallback e monitoraggio

Al di là della trascrizione dell'input vocale, il componente `WhisperDirectText.cs` implementa una serie di meccanismi per adattarsi al contesto in tempo reale, gestire input problematici e offrire strumenti utili in fase di sviluppo e testing. Queste funzionalità rendono l'interazione più fluida e resiliente, specialmente in ambienti XR dinamici.

Invio della trascrizione e meccanismi di fallback

Al termine del processo di trascrizione, il sistema valuta se il contenuto ottenuto è effettivamente utilizzabile. In caso di trascrizione valida, il testo viene immediatamente inoltrato al componente `ChatGPT.cs` per l'elaborazione semantica. Se invece la trascrizione finale risulta vuota o composta unicamente da spazi e caratteri non significativi, come può accadere in caso di completo silenzio da parte dell'utente o quando la trascrizione viene completamente annullata dai filtri di pulizia, il chatbot attiva una logica di fallback per gestire la mancanza di input valido.

In questa condizione, viene pronunciato un messaggio vocale localizzato che, in base al contesto, può essere un saluto come forma di invito a riformulare una richiesta o un messaggio di fine interazione. Il comportamento varia in funzione del numero di tentativi consecutivi andati a vuoto. Se si tratta del primo o del secondo tentativo, il sistema invoca il metodo `ChatbotGlobalData.GetLocalizedGreeting()`, che restituisce un messaggio iniziale coerente con la lingua corrente dell'interfaccia. Questo permette al chatbot di mantenere attiva la conversazione, pur senza un input significativo da parte dell'utente.

Tuttavia, se la condizione di trascrizione nulla si verifica per tre volte consecutive (valore configurabile nello script `WhisperDirectText.cs` tramite la variabile `maxCallsFromSilence`), il sistema considera conclusa l'interazione. In tal caso, viene pronunciato un messaggio di fine sessione generato da `GetInactivityMessage()` in `ChatbotGlobalData.cs`, e il chatbot viene disattivato.

```
// Se ho una trascrizione valida, la invio a ChatGPT, altrimenti invio un messaggio predefinito
if (!string.IsNullOrWhiteSpace(fullTranscription))
{
    SendToChatGPT(fullTranscription);
}
else
{
    if (callsFromSilence >= maxCallsFromSilence)
    {
        Debug.Log("chiamata a GetInactivityMessage in Whisper");
        ChatbotLog.instance.AddLog("Chiamata a GetInactivityMessage in whisper");
        ChatGPT.Instance.SendToTextToSpeech(ChatbotGlobalData.Instance.GetInactivityMessage());
    }
    else
    {
        Debug.Log("chiamata a GetLocalizedGreeting in Whisper");
        ChatbotLog.instance.AddLog("Chiamata a GetLocalizedGreeting in whisper");
        ChatGPT.Instance.SendToTextToSpeech(ChatbotGlobalData.Instance.GetLocalizedGreeting());
    }
}
```

Figura 3.14: Logica di fallback e gestione dell'inattività nel componente `Whisper DirectText.cs`. Se la trascrizione risulta vuota per più volte consecutive, viene pronunciato un messaggio di chiusura e l'interazione viene terminata automaticamente.

La logica implementata permette di evitare che il sistema continui a sollecitare l'utente in modo inappropriato, soprattutto nei casi in cui quest'ultimo abbia smesso di interagire o non sia più interessato alla conversazione. Questa gestione automatica dell'inattività contribuisce a mantenere l'esperienza utente fluida ed efficiente, riducendo le chiamate superflue e prevenendo loop conversazionali indesiderati.

Riconfigurazione dinamica

Il componente è progettato per rispondere in tempo reale a diversi eventi di sistema, garantendo una gestione flessibile delle funzionalità di riconoscimento vocale. Questa capacità di adattamento permette di mantenere allineati i moduli del sistema e di ottimizzare l'esperienza utente.

- **OnLanguageChanged**: questo evento viene attivato quando l'utente cambia la lingua di interazione. Il componente risponde interrompendo immediatamente qualsiasi registrazione audio in corso, per evitare che venga processato un input nella lingua precedente.
- **OnChatbotStateChanged**: questo evento gestisce l'attivazione o la disattivazione del sistema di ascolto automatico in base allo stato operativo del chatbot. Quando il chatbot è attivo, l'ascolto viene abilitato per permettere l'acquisizione continua degli input vocali. Se il chatbot viene disattivato, l'ascolto si interrompe per risparmiare risorse e evitare acquisizioni inutili. Questa gestione dinamica contribuisce anche a mantenere il controllo sull'interazione, impedendo ascolti indesiderati.
- **OnAudioStateChanged**: questo evento viene emesso dal componente `Text ToSpeech.cs` ogni volta che cambia lo stato dell'audio in riproduzione. Il componente `WhisperDirectText.cs` lo utilizza per monitorare se il chatbot sta parlando, evitando sovrapposizioni tra input e output vocali. La registrazione viene avviata solo quando il sistema ha completato la riproduzione, garantendo che l'utente non interrompa o sovrapponga la propria voce alla risposta. Questa sincronizzazione riduce rumori e potenziali errori di trascrizione, migliorando la fluidità e la naturalezza della conversazione.

- **OnChatGPTStatusChanged**: questo evento viene emesso ogni volta che cambia lo stato interno del modulo `ChatGPT.cs` riguardo l'attesa di una risposta da parte dell'LLM. In particolare, evita che il sistema inizi una nuova registrazione durante la fase di attesa della risposta da parte dell'API di OpenAI, che rappresenta un momento critico in cui il chatbot non è ancora pronto a gestire nuovi input. Bloccare l'ascolto in questa fase previene conflitti di stato e riduce i casi di trascrizioni non rilevanti o premature.

- **OnWhisperPromptChanged**: per ottimizzare la qualità della trascrizione, il componente aggiorna dinamicamente il prompt testuale inviato all'API di Whisper. L'evento viene chiamato da `PromptEngineering.cs`, ad esempio quando cambia la location corrente, così da adattare il contesto del riconoscimento vocale in base all'ambiente o al tema atteso. Questo aggiornamento contribuisce a ridurre errori e migliorare la pertinenza delle trascrizioni ricevute, soprattutto in scenari con alta variabilità tematica.

```
private void OnWhisperPromptChanged(string prompt)...

private void OnLanguageChanged(ChatbotGlobalData.Language language)...

private void OnChatbotStateChanged(ChatbotGlobalData.ChatbotState newState)
{
    isChatbotActive = (newState == ChatbotGlobalData.ChatbotState.Active);
    callsFromSilence = 0;
}

private void OnChatGPTStatusChanged(bool isWaiting)
{
    isWaitingForResponse = isWaiting;
}

private void OnAudioStateChanged(string state)
{
    // If the state is not "Finished", the chatbot is still speaking
    isSpeaking = (state != "Finished");

    // Start recording only if all required conditions are met
    if (!isRecording && !isSpeaking && isChatbotActive && !isWaitingForResponse)
    {
        if (callsFromSilence == maxCallsFromSilence)
        {
            Debug.Log("callsFromSilence == maxCallsFromSilence - Deactivating chatbot");
            ChatbotLog.instance.AddLog("callsFromSilence == maxCallsFromSilence - Deactivating chatbot");
            ChatbotGlobalData.Instance.chatbotState = ChatbotGlobalData.ChatbotState.Inactive;
            return;
        }
        else
        {
            Debug.Log("Starting audio recording...");
            ChatbotLog.instance.AddLog("Starting audio recording...");
            StartCoroutine(StartRecording());
        }
    }
}
```

Figura 3.15: Gestione degli eventi interni nel componente `WhisperDirectText.cs`.

Sebbene tutti gli eventi descritti contribuiscano alla riconfigurazione dinamica del componente, `OnAudioStateChanged` rappresenta l'innesco principale dell'intero processo di ascolto. È infatti questo l'evento che, una volta accertato che il chatbot sia attivo, non stia parlando, non sia già in fase di registrazione e non attenda una risposta dal modello linguistico, attiva la routine `StartRecording()`. Gli altri eventi agiscono invece come condizioni di controllo aggiuntive, garantendo che l'ascolto avvenga solo nei momenti appropriati, in modo da evitare interferenze, chiamate ridondanti o registrazioni indesiderate.

Questa architettura reattiva consente al sistema di adattarsi efficacemente al contesto XR, assicurando fluidità nell'interazione vocale e una trascrizione affidabile, anche in scenari dinamici o multilingue.

Strumenti di debug e monitoraggio

Per agevolare il testing, è disponibile un sistema opzionale di *audio replay*, attivabile dall'inspector, che consente di riprodurre localmente la clip registrata tramite un `AudioSource` separato. In produzione, questa funzione viene disabilitata automaticamente.

Tutte le fasi critiche del flusso di trascrizione sono monitorate e registrate nel sistema di logging interno (`ChatbotLog.cs`), che supporta colori, livelli di priorità e reset automatico dei log ad ogni nuova sessione.

3.6 Interazione con il modello LLM

`ChatGPT.cs` opera come livello intermedio tra Unity e le API OpenAI, centralizzando la gestione del flusso conversazionale. Ha il compito di inviare le richieste al Large Language Model, mantenere la memoria del dialogo e restituire le risposte al modulo vocale.

3.6.1 Struttura e gestione dello stato

La classe implementa un pattern singleton (`ChatGPT.Instance`) per consentire l'accesso globale e centralizzato al componente. Al momento dell'avvio, viene creata un'istanza di `OpenAIApi`, passando le credenziali contenute in `ChatbotGlobalData.cs` (chiave API e ID organizzazione).

3.6.2 Cronologia conversazionale e messaggi

La comunicazione con il modello LLM avviene tramite una lista di oggetti `ChatMessage`, ciascuno dei quali rappresenta un intervento nel dialogo e specifica il ruolo del mittente (`system`, `user` o `assistant`). Inoltre, la gestione della cronologia è dinamica e si adatta al contesto dell'interazione.

Sebbene il prompt di sistema venga aggiornato ad ogni interazione per riflettere l'intento corrente dell'utente, esso viene rigenerato anche in seguito a eventi specifici come il cambio di lingua o di location. Questo consente al modello di adattarsi non solo al flusso della conversazione, ma anche a variazioni ambientali rilevanti.

- Il prompt di sistema viene aggiornato ad ogni interazione, prendendo i contenuti da `PromptEngineering.cs` e sovrascrivendo il messaggio iniziale con ruolo `system`
- I messaggi dell'utente vengono aggiunti in fondo alla lista
- Le risposte del modello vengono anch'esse memorizzate per mantenere il contesto

```
// Aggiorna il prompt corrente prima di inviare la richiesta
if (!string.IsNullOrEmpty(currentPrompt))
{
    // Aggiunge o aggiorna il messaggio di sistema con l'ultimo prompt
    if (messages.Count > 0 && messages[0].Role == "system")
    {
        messages[0] = new ChatMessage()
        {
            Role = "system",
            Content = currentPrompt,
        };
    }
    else
    {
        messages.Insert(0, new ChatMessage()
        {
            Role = "system",
            Content = currentPrompt,
        });
    }
}

// Aggiunge il messaggio dell'utente alla lista dei messaggi
messages.Add(new ChatMessage()
{
    Role = "user",
    Content = userMessage,
});
```

Figura 3.16: Esempio di aggiornamento dinamico del messaggio `system` nella cronologia. Se già presente, viene sovrascritto; altrimenti, viene inserito in testa alla lista. Segue l'aggiunta del messaggio dell'utente.

In questo modo, il modello mantiene coerenza tra una richiesta e la successiva, adattandosi all'evoluzione del dialogo e al mutare del contesto ambientale.

3.6.3 Invio della richiesta asincrona

L'invio del messaggio al modello di linguaggio avviene in modalità completamente asincrona tramite il metodo `CreateChatCompletion()`, offerto dalla libreria ufficiale OpenAI per C#. Questo approccio consente di evitare il blocco del thread principale di Unity durante l'attesa della risposta, mantenendo la reattività dell'interfaccia XR anche in caso di latenza di rete o carico sul modello.

Il modello da utilizzare viene selezionato dinamicamente in base a un campo serializzato di tipo `OpenAIModel`, impostabile da *Inspector*. Tale enum consente di scegliere, per ciascuna sessione o zona, il motore linguistico desiderato tra quelli disponibili, ognuno dei quali può presentare differenze in termini di velocità, capacità di ragionamento e costo per token.

Come approfondito nella sezione dedicata al prompt engineering, la scelta del modello ha un impatto diretto anche sulla tipologia e struttura del prompt da utilizzare, poiché ciascun modello può richiedere un'impostazione ottimizzata per sfruttarne al meglio le specifiche caratteristiche.

Modello	Codice API	Prompt associato
GPT-3.5 Turbo	`gpt-3.5-turbo`	`Prompt1()`
GPT-4 Turbo	`gpt-4-turbo`	`Prompt1()`
GPT-4o Mini	`gpt-4o-mini`	`Prompt3()`

Tabella 3.7: Modelli supportati e template di prompt associati

In fase di invio, il sistema costruisce dinamicamente il corpo della richiesta JSON, includendo:

- Il modello selezionato.
- Il messaggio utente preprocessato e corretto.
- L'intera cronologia della conversazione (in caso di contesto abilitato).
- Parametri opzionali come temperatura o il numero massimo di token.

Tutti i parametri vengono gestiti in modo configurabile, consentendo test comparativi e ottimizzazione delle prestazioni anche a runtime.

3.6.4 Controllo degli stati e resilienza

Durante l'invio della richiesta, il componente emette un evento (`OnChatGPTStatusChanged`) che segnala agli altri sistemi l'inizio e la fine dell'elaborazione. Questo stato viene utilizzato, ad esempio, da `WhisperDirectText.cs` per evitare di registrare mentre è in corso una richiesta al modello linguistico.

Inoltre, un meccanismo interno verifica se la lingua è cambiata durante l'attesa della risposta. In tal caso, il messaggio ricevuto viene ignorato e la richiesta viene reinviata automaticamente, utilizzando il prompt aggiornato nella nuova lingua, così da mantenere la coerenza dell'interazione.

3.6.5 Pulizia e normalizzazione della risposta

Prima che la risposta venga inviata al sistema di sintesi vocale, viene applicata una pulizia minimale tramite il metodo `ProcessAnswer()`, che rimuove simboli potenzialmente problematici per il motore TTS.

In particolare:

- Viene evitato l'uso di simboli come `*`, `_` o emoticon testuali

- La risposta viene rifinita per garantire una lettura più naturale da parte del sintetizzatore

3.6.6 Interfacciamento con il sistema vocale

Una volta elaborata e ripulita, la risposta testuale viene inviata al componente `TextToSpeech.cs` tramite il metodo `SendToTextToSpeech()`. Contestualmente, viene aggiornato lo stato interno `ChatbotActiveSubState` con il valore `Answering`, utile per sincronizzare le animazioni del viso e la gestione dei turni di parola.

3.6.7 Gestione degli errori

Tutte le chiamate vengono eseguite all'interno di un blocco `try-catch` per catturare eventuali eccezioni (es. problemi di rete, chiavi errate, limiti di rate API).

In caso di errore:

- Il messaggio di errore viene loggato su `ChatbotLog.cs`, evidenziato in rosso.
- Non viene aggiornata la cronologia per evitare inconsistenze.
- Il sistema ritorna allo stato di attesa.

3.6.8 Chiamate da fonti esterne

Il componente `ChatGPT.cs` espone due metodi pubblici che consentono ad altri script di inviare messaggi al modello:

- `SendReplyFromAudio(string)`: viene invocato direttamente da `WhisperDirectText.cs` per trasmettere il testo trascritto dall'audio.
- `SendReplyFromOtherScripts(string)`: è pensato per essere utilizzato da altri componenti generici, come pulsanti dell'interfaccia, eventi Unity o altre logiche personalizzate.

Questa separazione semantica migliora la leggibilità del codice e favorisce un'architettura modulare, evitando accoppiamenti stretti tra i componenti. I metodi fungono da interfacce minime per accedere alle funzionalità principali del sistema, garantendo così riutilizzabilità e facilità di manutenzione.

3.7 Sintesi vocale con Amazon Polly

La sintesi vocale nel sistema è affidata ad Amazon Polly, un servizio cloud-based di tipo NTTS (Neural Text-to-Speech) in grado di generare voci naturali e realistiche in numerose lingue. L'integrazione con Unity è gestita dal componente `TextToSpeech.cs`, che svolge un ruolo centrale non solo nella comunicazione asincrona con le API di Polly, ma anche nella selezione automatica della voce in base alla lingua attualmente attiva, nella riproduzione del parlato sull'avatar virtuale, nella sincronizzazione tra audio e animazioni facciali, nonché nell'emissione di eventi di stato per il coordinamento con altri moduli del sistema, come `Whisper` e `ChatGPT`.

Oltre alla semplice conversione da testo a parlato, il componente implementa anche funzionalità avanzate, tra cui meccanismi di fallback in caso di errore, normalizzazione automatica dell'input testuale e supporto opzionale all'audio spazializzato, con l'obiettivo di aumentare l'immersività dell'esperienza utente in ambienti XR.

La sua architettura modulare è progettata per garantire flessibilità ed estendibilità, rendendo possibile l'integrazione futura di ulteriori servizi TTS, sistemi di caching delle risposte vocali, o personalizzazioni della voce sintetica.

3.7.1 Pipeline operativa

`TextToSpeech.cs` gestisce l'intero processo di sintesi vocale in maniera modulare e asincrona, seguendo una pipeline ben definita che assicura fluidità, resilienza e compatibilità con ambienti XR, oltre a garantire la massima reattività e continuità dell'esperienza utente.

Di seguito vengono descritti i principali passaggi operativi:

- **Selezione della voce**: il sistema determina automaticamente la voce da utilizzare in base alla lingua attiva, ottenuta dal campo `AnswersLanguage` presente in `ChatbotGlobalData`. A ciascuna lingua è associata una voce Polly specifica definita in un dizionario interno.
- **Selezione dei componenti audio e animazione**: prima di avviare la riproduzione vocale, il sistema aggiorna dinamicamente i riferimenti ai componenti `AudioSource` e `Animator` in base al chatbot attualmente attivo.

- **Costruzione della richiesta**: viene generata una richiesta HTTP di tipo POST[10] verso l'endpoint di Amazon Polly.
- **Invio asincrono della richiesta**: la richiesta viene inoltrata tramite `UnityWebRequest`, in modalità asincrona per evitare blocchi sul thread principale di Unity.
- **Ricezione e decodifica dell'audio**: la risposta contenente il file audio (tipicamente in formato `mp3`) viene decodificata e convertita in un oggetto `AudioClip`, compatibile con il sistema audio di Unity. La conversione è affidata a un modulo interno di parsing del flusso binario.
- **Riproduzione sincronizzata sull'avatar**: l'`AudioClip` viene assegnato al `currentAudioSource`, collegato all'avatar attualmente attivo. La riproduzione avviene in sincronia con il sistema di animazione facciale (`OVRLipSyncContex MorphTargetTest`) e con l'attivazione delle animazioni corporee mediante il `currentAnimator`.
- **Notifica dello stato audio**: tramite l'evento `OnAudioStateChanged`, vengono informati gli altri componenti sullo stato corrente dell'audio (ad esempio: inizio, riproduzione o fine). Questa notifica consente di coordinare correttamente le fasi di sintesi vocale e di riconoscimento, evitando interferenze e garantendo una gestione fluida dell'interazione.

Questa architettura rende il modulo TTS non solo robusto, ma anche facilmente estendibile verso ulteriori servizi vocali (es. Azure TTS, Google TTS) o verso funzionalità avanzate come il caching delle risposte, il supporto a SSML (Speech Synthesis Markup Language)[11], o lo streaming audio per dialoghi più dinamici.

Inoltre, in caso di fallimento del processo (es. timeout, errore di rete, voce non disponibile), il sistema genera un messaggio di fallback localizzato e registra l'evento nel sistema di logging interno, mantenendo una tracciabilità dettagliata per il debugging.

---

[10] Nel protocollo HTTP, una richiesta di tipo POST viene utilizzata per inviare dati al server, ad esempio per creare o processare una risorsa. Diversamente da una richiesta GET (che serve a recuperare dati), una POST invia informazioni nel corpo della richiesta e viene comunemente impiegata nelle API per trasmettere contenuti strutturati come JSON o XML.

[11] SSML è uno standard basato su XML che consente di controllare in modo preciso la resa vocale del testo sintetizzato, permettendo l'inserimento di pause, cambi di intonazione, accenti, velocità di lettura e altri aspetti prosodici.

3.7.2 Mappatura delle voci e nomi avatar

Per ciascuna lingua supportata è stata definita un'associazione coerente tra voce sintetica, nome e aspetto visivo dell'avatar, in modo da rafforzare l'identità percepita e la continuità dell'interazione. Come mostrato nella Tabella 3.8, ogni combinazione è pensata per risultare credibile e armonizzata con le caratteristiche linguistiche e culturali di riferimento.

Lingua	**Voce Polly**	**Nome del chatbot**
Inglese	Emma	Emma
Italiano	Carla[12]	Marta
Francese	Claire	Claire
Spagnolo	Sofía	Sofía

Tabella 3.8: Mappatura tra lingua, voce Polly e nome visualizzato del chatbot

Anche il design visivo degli avatar è stato calibrato per suggerire, in modo sobrio e senza ricorrere a stereotipi, una provenienza culturale coerente con la lingua associata. Elementi come l'abbigliamento, i lineamenti del volto o l'acconciatura aiutano a rendere immediata e naturale l'associazione tra parlato sintetico e identità visiva. La selezione delle voci avviene tramite una struttura interna (come un dizionario o uno `switch-case`), facilmente estendibile a nuove lingue.

3.7.3 Gestione delle animazioni e componenti audio associati

Per garantire una sincronizzazione accurata tra voce sintetizzata e animazioni, il componente `TextToSpeech.cs` aggiorna dinamicamente i riferimenti ai componenti `AudioSource` e `Animator` in base al chatbot attualmente attivo. Questo meccanismo consente di gestire più lingue e avatar senza dover accedere ogni volta alla gerarchia della scena, migliorando così la reattività del sistema e la modularità dell'architettura.

Come si può osservare in Figura 3.17, per mantenere l'associazione tra lingua, voce e comportamento animato vengono utilizzate alcune strutture dati dedicate, configurabili direttamente dall'Inspector di Unity. Queste permettono di collegare a ogni lingua un componente `Animator` per le animazioni e una sorgente `AudioSource` per la riproduzione del parlato sintetizzato.

[12]Nel progetto il nome utilizzato è *Marta*, sebbene la voce effettiva sia *Carla*.

```
private Dictionary<ChatbotGlobalData.Language, Animator> animators = new Dictionary<ChatbotGlobalData.Language, Animator>();
private Dictionary<ChatbotGlobalData.Language, AudioSource> audioSources = new Dictionary<ChatbotGlobalData.Language, AudioSource>();

[System.Serializable]
public class Animators
{
    public ChatbotGlobalData.Language language;
    public Animator anim;
}

[System.Serializable]
public class AudioSources
{
    public ChatbotGlobalData.Language language;
    public AudioSource audioSource;
}

[SerializeField] private Animators[] animatorsArray;
[SerializeField] private AudioSources[] audioSourcesArray;


private Animator currentAnimator = null;
private AudioSource currentAudioSource = null;
private VoiceId currentVoice = null;
private int playSessionId = 0;
```

Figura 3.17: Strutture dati per associare animator e sorgenti audio a ciascuna lingua, con supporto all'inserimento tramite Inspector.

Nel dettaglio, la logica si basa su due dizionari principali:

- **Dictionary<Language, Animator> animators**: associa ogni lingua al componente `Animator` dell'avatar corrispondente. Questo permette al sistema di attivare le animazioni corrette (espressioni facciali, gesti) in base alla lingua attualmente selezionata.
- **Dictionary<Language, AudioSource> audioSources**: collega ogni lingua a una sorgente `AudioSource` separata, consentendo la riproduzione del parlato sintetizzato con un controllo indipendente del volume, del pitch e di altri parametri audio.

Poiché Unity non supporta la serializzazione diretta dei dizionari, sono stati implementati due array serializzabili di supporto, `animatorsArray` e `audioSourcesArray`, i cui elementi sono strutture dati personalizzate definite all'interno della classe.

- La classe **Animators** rappresenta una coppia `(language, anim)`, utilizzata per popolare il dizionario `animators` all'avvio dell'applicazione.
- La classe **AudioSources** contiene una coppia analoga, `(language, audioSource)`, e serve a inizializzare il dizionario `audioSources`.

Questi array sono visibili e modificabili dall'Inspector, rendendo semplice l'assegnazione manuale dei componenti necessari a ogni lingua senza scrivere codice.

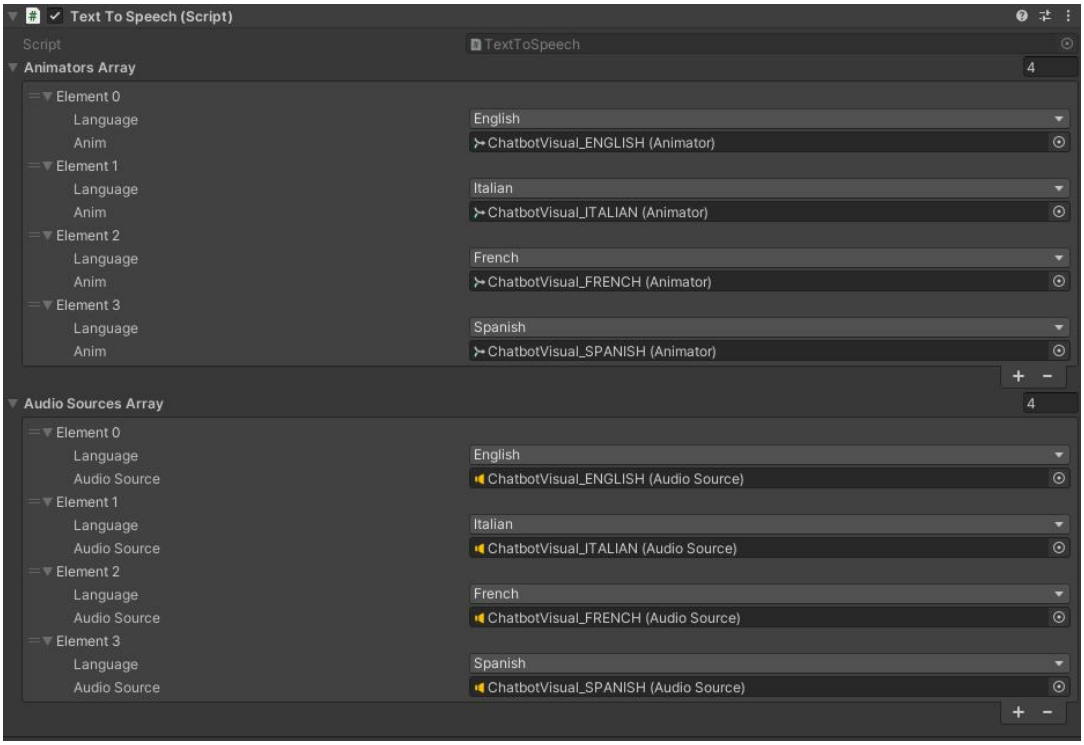


Figura 3.18: Assegnazione manuale degli animatori e delle sorgenti audio per ciascuna lingua nel componente `TextToSpeech`, tramite l'Inspector di Unity.

All'avvio, gli array vengono convertiti in dizionari per consentire un accesso rapido ed efficiente a runtime. Il componente mantiene inoltre alcuni riferimenti dinamici fondamentali:

- **Animator currentAnimator**: riferimento all'`Animator` associato alla lingua attiva, utilizzato per avviare le animazioni sincronizzate con il parlato.
- **AudioSource currentAudioSource**: riferimento alla sorgente audio attiva per la riproduzione del parlato sintetizzato.
- **VoiceId currentVoice**: identificativo della voce Amazon Polly selezionata per la lingua corrente.
- **int playSessionId**: contatore incrementale che identifica univocamente ogni sessione audio, utile per evitare conflitti o sovrapposizioni in caso di riproduzioni ravvicinate.

Ogni volta che viene attivata una nuova lingua, il sistema aggiorna i riferimenti a `currentAnimator` e `currentAudioSource` in modo da:

- Riprodurre la voce dalla sorgente corretta all'interno del prefab dell'avatar attivo.

- Abilitare le animazioni facciali e gestuali tramite l'`Animator` corrispondente.
- Garantire la sincronizzazione tra audio e movimenti labiali grazie all'integrazione con il componente `OVRLipSyncContextMorphTargetTest`.

Questa struttura modulare facilita l'estensione a nuove lingue e avatar, isola i canali audio e animativi di ciascun chatbot e mantiene una pipeline di esecuzione efficiente e priva di conflitti.

3.7.4 Costruzione e invio della richiesta per la sintesi vocale

La generazione della voce sintetizzata avviene tramite l'invio di una richiesta HTTP di tipo `POST` all'endpoint del servizio Amazon Polly. La richiesta comprende tutte le informazioni necessarie affinché Polly possa generare correttamente l'audio desiderato.

In particolare, il corpo della richiesta contiene i seguenti parametri:

- **Testo da convertire**: prima di essere incluso nella richiesta, il testo viene sottoposto a una pulizia preventiva per rimuovere caratteri speciali, simboli non supportati e potenziali sequenze ambigue. Questo passaggio è fondamentale per evitare errori di sintesi o risultati imprevisti, specialmente in presenza di contenuti generati automaticamente.
- **Voce selezionata (`VoiceId`)**: corrisponde al nome della voce Polly da utilizzare per la generazione del parlato. Ogni voce è associata a una specifica lingua, sesso e stile espressivo; il sistema seleziona dinamicamente quella più adatta in base alla lingua corrente e al contesto.
- **Formato audio desiderato**: il parametro `OutputFormat` specifica il tipo di file audio da ricevere in risposta. Nel progetto sono stati testati sia il formato compresso `mp3`, più leggero e adatto alla riproduzione, sia il formato `pcm` (Pulse Code Modulation), utile per elaborazioni successive o per l'integrazione con sistemi di lip sync avanzati.
- **Modalità di sintesi**: tramite il parametro `Engine` viene specificato il motore di sintesi da utilizzare. Nel progetto è stato adottato `Engine.Standard`, che impiega il modello vocale tradizionale di Amazon Polly. È disponibile anche la modalità neurale (`Engine.Neural`), che sfrutta reti neurali profonde per generare una voce più naturale ed espressiva, laddove supportata dalla voce selezionata. Al momento dello sviluppo, il motore neurale risultava disponibile solo per alcune lingue e non era compatibile con le voci scelte per l'applicazione.

È prevedibile che in un futuro prossimo tale supporto venga esteso, consentendo di sfruttare modelli neurali anche per altre lingue e voci, con un conseguente miglioramento della qualità del parlato sintetico.

```
var request = new SynthesizeSpeechRequest()
{
    Text = text,
    Engine = Engine.Standard,
    VoiceId = voice,
    OutputFormat = OutputFormat.Mp3
};
```

Figura 3.19: Esempio di corpo della richiesta inviata al servizio di sintesi vocale Amazon Polly, con evidenza dei principali parametri configurabili come testo, voce, formato audio e modalità di sintesi.

Tutti i parametri vengono serializzati in formato JSON e inclusi nel corpo della richiesta. Lo script gestisce inoltre l'inserimento delle intestazioni HTTP richieste da Polly, comprese le chiavi di autenticazione, il tipo di contenuto e altri metadati obbligatori per l'accesso all'API.

Questa fase rappresenta un punto critico della pipeline di sintesi, poiché qualsiasi errore nella formattazione o nei contenuti della richiesta può comportare il fallimento del processo o una qualità vocale non ottimale. Una costruzione rigorosa e modulare della richiesta consente di garantire stabilità, tracciabilità ed estendibilità per eventuali personalizzazioni future.

Una volta completata la costruzione della richiesta, essa viene inoltrata verso l'endpoint di Amazon Polly tramite `UnityWebRequest`, utilizzando una coroutine per garantire un'esecuzione asincrona. Questa scelta evita il blocco del thread principale di Unity durante l'attesa della risposta, assicurando fluidità dell'interfaccia e dell'esperienza utente anche in presenza di tempi di risposta variabili.

3.7.5 Ricezione e gestione del flusso audio

Una volta completata la sintesi vocale da parte di Amazon Polly, il file audio risultante viene restituito sotto forma di stream binario. Il formato del file dipende dal parametro `OutputFormat` specificato nella richiesta.

In fase di sviluppo sono stati sperimentati due formati principali:

- **MP3**: formato compresso ad alta compatibilità, adatto alla riproduzione diretta all'interno di Unity. La sua leggerezza consente una trasmissione più rapida e un utilizzo efficiente della memoria, risultando particolarmente indicato per ambienti real-time.
- **PCM (Pulse Code Modulation)**: formato non compresso, utilizzato per test interni o per possibili elaborazioni avanzate, come l'integrazione con sistemi di lip sync che richiedano accesso diretto all'onda sonora campionata.

Dopo diverse prove e valutazioni, è stato scelto di adottare il formato `mp3` per tutte le richieste, poiché rappresenta il miglior compromesso tra qualità percepita e semplicità di gestione all'interno dell'ambiente Unity. Tuttavia, l'adozione di un formato più neutro come il `pcm` potrebbe essere rivalutata in scenari futuri, qualora emergesse la necessità di effettuare analisi più dettagliate sul segnale audio o di supportare tecniche di animazione facciale fonema-specifica.

Il flusso `AudioStream` restituito da Polly viene copiato in un `MemoryStream` locale, convertito in un array di byte e temporaneamente salvato in un file `.mp3` all'interno del percorso persistente dell'applicazione Unity. Questo file viene poi caricato in modo asincrono tramite `UnityWebRequestMultimedia.GetAudioClip`, convertito in un oggetto `AudioClip` e assegnato al componente `AudioSource` attivo. Solo al termine di questa fase viene avviata la riproduzione.

```
var response = responseTask.Result;
using (var memoryStream = new MemoryStream())
{
    response.AudioStream.CopyTo(memoryStream);
    memoryStream.Seek(0, SeekOrigin.Begin);
    byte[] audioBytes = memoryStream.ToArray();
    yield return LoadAudioFromMemory(audioBytes, audioSource, animator, answering, sessionId);
}
private IEnumerator LoadAudioFromMemory(byte[] audioBytes, AudioSource audioSource, Animator animator, bool answering, int sessionId)
{
    if (sessionId != playSessionId)...

    var tempPath = Path.Combine(Application.persistentDataPath, "tempAudio.mp3");
    File.WriteAllBytes(tempPath, audioBytes);

    using (var www = UnityWebRequestMultimedia.GetAudioClip($"file://{tempPath}", AudioType.MPEG))
    {
        yield return www.SendWebRequest();

        if (www.result == UnityWebRequest.Result.Success)
        {
            var clip = DownloadHandlerAudioClip.GetContent(www);

            audioSource.clip = clip;
            audioSource.Play();
```

Figura 3.20: Codice per la gestione asincrona del flusso audio di Amazon Polly, dalla conversione in file MP3 al caricamento come AudioClip in Unity.

Durante la fase di caricamento dell'audio, viene verificato che l'identificatore di sessione (`sessionId`) associato al turno vocale corrente sia ancora valido. In caso contrario, il caricamento viene interrotto prima che la riproduzione abbia inizio, evitando potenziali conflitti o sovrapposizioni nel caso in cui l'utente abbia, ad esempio, cambiato lingua o avviato una nuova interazione.

Questo meccanismo consente di garantire che ogni clip audio venga elaborata e riprodotta esclusivamente nel contesto corretto, mantenendo la coerenza del dialogo e la sincronizzazione con le animazioni labiali.

3.7.6 Logica di attivazione delle animazioni vocali

`TextToSpeech.cs` non si limita alla sintesi vocale, ma integra anche la gestione del comportamento animato dell'avatar durante il parlato e nei momenti di inattività. Questa componente è fondamentale per assicurare una resa visiva credibile e coerente in ambienti immersivi, dove l'avatar rappresenta l'unico mediatore del dialogo.
Le chiamate alle animazioni avvengono in momenti chiave della conversazione e seguono una logica ben definita:

- **Prima dell'inizio del parlato**: viene attivata un'animazione di tipo *speak*, che imposta l'avatar in uno stato di espressività verbale. In questo momento, il sistema aggiorna anche i riferimenti all'`Animator` e all'`AudioSource` associati alla lingua corrente, per assicurarsi che la riproduzione e l'animazione siano sincronizzate.

- **Durante la riproduzione audio**: il parlato viene riprodotto dalla sorgente audio corretta e il sistema di sincronizzazione labiale (basato su `OVRLipSync ContextMorphTarget`[13]) modula i movimenti della bocca. L'animazione *speak* rimane attiva per tutta la durata del clip audio.

- **Al termine della riproduzione**: una volta conclusa l'esecuzione del file audio, viene automaticamente avviata un'animazione di tipo *idle*, che riporta l'avatar a uno stato neutro o di attesa. In questo modo si evita che il personaggio resti "congelato" in una posizione parlante non più coerente con il contesto.

- **In caso di errore o fallback**: se la generazione o riproduzione del parlato fallisce, viene comunque invocata l'animazione *idle* per evitare blocchi visivi o posture errate dell'avatar.

[13] Questo componente richiede l'analisi in tempo reale della forma d'onda audio per modulare i movimenti della bocca in modo naturale e coerente con il parlato.

L'intera logica di transizione tra stati animati è incapsulata all'interno dello script, che richiama in modo centralizzato le animazioni attraverso i riferimenti `currentAnimator` associati alla lingua selezionata. La separazione netta tra la fase *speaking* e quella *idling* consente di mantenere l'interazione coerente e gestire correttamente anche l'interruzione o l'annullamento anticipato di una riproduzione vocale.

Questa gestione, pur essendo trasparente all'utente finale, risulta cruciale per la fluidità e la credibilità dell'interazione, soprattutto in ambienti XR dove anche lievi disallineamenti visivi possono compromettere l'esperienza immersiva.

3.7.7 Sincronizzazione e stato audio

Durante la riproduzione del parlato sintetico, lo script `TextToSpeech.cs` invia aggiornamenti attraverso l'evento `OnAudioStateChanged`, che segnala i cambiamenti di stato dell'audio lungo tutta la catena del parlato. Questo meccanismo serve a coordinare in modo centralizzato le varie componenti coinvolte, mantenendo l'interazione vocale ordinata e priva di conflitti.

Gli stati previsti sono i seguenti:

- **Started**: l'audio è stato caricato correttamente e la riproduzione è appena iniziata.
- **Playing**: l'audio è in fase di riproduzione e l'avatar sta animando il parlato.
- **Finished**: l'audio è terminato e le operazioni collegate possono essere completate.

Queste transizioni di stato sono importanti per permettere ad altri componenti del sistema di reagire nel modo giusto. In particolare, vengono utilizzate per:

- **Gestire i turni di conversazione**: La registrazione dell'utente (basata su Whisper) può partire solo quando il chatbot ha finito di parlare, evitando sovrapposizioni tra la voce sintetica e l'input vocale dell'utente.
- **Attivare la sincronizzazione labiale**: Quando lo stato è `Playing`, il sistema di sincronizzazione labiale (`OVRLipSyncContextMorphTargetTest`) viene attivato per generare movimenti realistici delle labbra in base all'audio riprodotto, contribuendo a una maggiore credibilità dell'avatar.

- **Aggiornare lo stato del chatbot**: lo stato interno del modulo conversazionale (`ChatbotActiveSubState`) viene aggiornato in base agli eventi audio, così il sistema sa se l'avatar sta parlando, ascoltando o è in attesa. Questo permette di sincronizzare in modo coerente animazioni, logica conversazionale e interazione utente.

Questo semplice meccanismo a eventi funge quindi da punto di raccordo tra le fasi di output (Text-to-Speech) e input (Speech-to-Text), mantenendo l'avatar reattivo e ben coordinato anche in contesti immersivi, dove la corretta tempistica delle interazioni ha un ruolo decisivo nella credibilità dell'esperienza.

3.7.8 Gestione degli errori e fallback vocale

Il sistema include un insieme di strategie di gestione degli errori volte a garantire la continuità dell'esperienza utente anche in presenza di malfunzionamenti del servizio TTS.

Questi meccanismi comprendono:

- **Timeout automatici** sulle richieste HTTP, per evitare blocchi in caso di latenza eccessiva.
- **Gestione delle eccezioni** a livello di rete (es. DNS fallito, credenziali errate, endpoint non raggiungibile).
- **Fallback a messaggio statico** localizzato, con testo predefinito in ciascuna lingua, attivato quando la sintesi fallisce.
- **Logging dettagliato** tramite il componente `ChatbotLog.cs`, con codice di errore, descrizione e timestamp.

Lo script non permette la sintesi vocale offline: in assenza di connessione o connettività instabile, l'applicazione non riesce a ottenere la risposta da Polly, generando messaggi di log dettagliati e impedendo la riproduzione vocale. Tuttavia, il meccanismo di fallback serve a gestire questi errori in modo ordinato, evitando che il sistema si blocchi e consentendo di notificare l'utente o attivare comportamenti alternativi per mantenere l'esperienza più fluida possibile.

3.8 Animazione e sincronizzazione del viso e del corpo

Gli avatar 3D utilizzati nel progetto sono stati generati tramite la piattaforma *Ready Player Me*[14], uno strumento online che consente la creazione di modelli umani realistici, altamente personalizzabili e pienamente compatibili con il motore Unity. I modelli esportati includono una struttura scheletrica completa e una vasta gamma di *blend shapes*, che abilitano animazioni facciali avanzate come il movimento sincronizzato di labbra o occhi. Tali caratteristiche rendono questi avatar particolarmente idonei per scenari interattivi e immersivi, in cui la componente espressiva riveste un ruolo centrale.

Nel contesto del progetto, il sistema gestisce in modo integrato sia la sincronizzazione labiale, basata sull'audio generato dinamicamente, sia un insieme coordinato di animazioni corporee di supporto. L'obiettivo è quello di incrementare il realismo percepito, migliorare la qualità dell'esperienza utente e facilitare un coinvolgimento più naturale durante l'interazione conversazionale.

Figura 3.21: Uno degli avatar 3D generati con la piattaforma Ready Player Me, integrato nell'ambiente virtuale del museo aziendale.

[14] https://readyplayer.me/

3.8.1 Struttura del prefab e componenti

Ogni avatar multilingua è istanziato a partire da un prefab distinto (es. `Chatbot Visual_ITALIAN`) dove ciascuno è dotato dei seguenti componenti principali:

- **AvatarData.cs**: contiene metadati relativi all'identità e configurazione dell'avatar.
- **Animator**: utilizzato per controllare le animazioni corporee tramite meccanismi a stato finito.
- **AudioSource**: impiegato per la riproduzione dell'audio vocale generato da Amazon Polly.
- **OVRLipSyncContext** e **OVRLipSyncContextMorphTargetTest**: componenti per la sincronizzazione labiale automatica basati sull'*Oculus Lip Sync SDK*.
- **Eye Animation Handler**: gestisce il battito delle palpebre e i movimenti oculari.

Il componente `OVRLipSyncContextMorphTargetTest` rappresenta una versione estesa del `OVRLipSyncContextMorphTarget` standard fornito dall'SDK Oculus. Entrambi i componenti hanno la funzione di mappare i visemi riconosciuti sul modello 3D dell'avatar, attivando i relativi blend shape per simulare il movimento delle labbra durante il parlato.

La differenza principale risiede nell'aggiunta, nella versione `Test`, di un supporto esplicito per una *seconda mesh* dedicata alla dentatura. Nello specifico, il componente include un secondo riferimento a un `SkinnedMeshRenderer`, assegnato in Unity alla mesh dei denti. Questo consente di applicare in parallelo gli stessi blend shape anche a quest'ultima, garantendo una sincronizzazione coerente tra il movimento delle labbra e quello dei denti.

Questa estensione è stata introdotta per migliorare la naturalezza visiva dell'animazione facciale, evitando l'effetto innaturale che si verificava quando la bocca si muoveva ma i denti rimanevano statici.

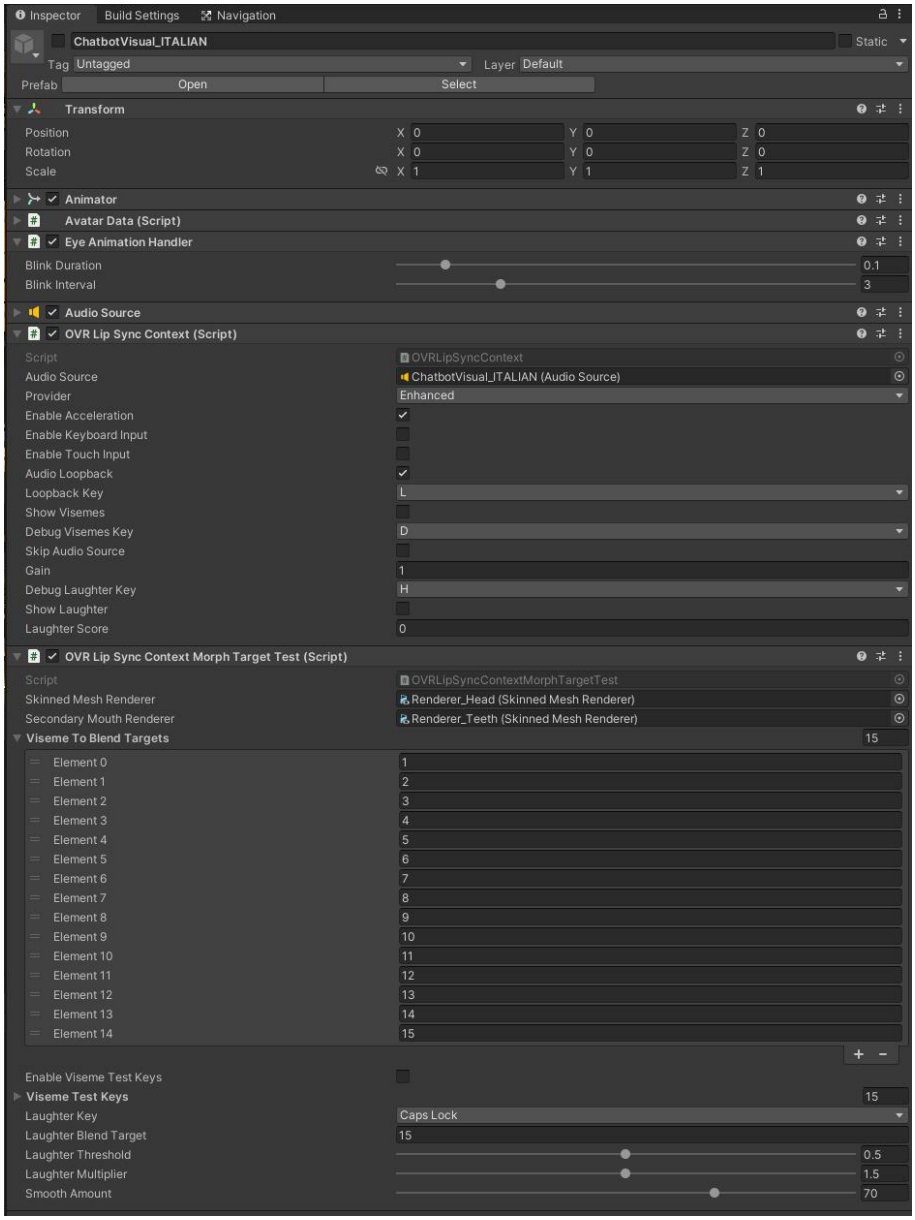


Figura 3.22: Vista dell'inspector del prefab `ChatbotVisual-ITALIAN` con i relativi componenti.

Ogni prefab contiene due sotto-oggetti fondamentali:

- **Armature**: l'ossatura scheletrica usata dall'Animator.
- **Renderer**: contiene le mesh renderizzate con i relativi *Skinned Mesh Renderer*, che supportano blend shapes per le animazioni facciali e non solo.

Figura 3.23: Struttura del prefab `ChatbotVisual-ITALIAN` all'interno della *Hierarchy* di Unity.

3.8.2 Gestione multilingua

A ogni lingua supportata corrisponde un prefab avatar distinto.

Questa separazione permette di:

- Assegnare voci e animazioni diverse in base al contesto linguistico.
- Personalizzare nome e caratteristiche estetiche del personaggio per migliorarne la coerenza culturale e narrativa (es. "Emma" per inglese, "Claire" per francese).
- Separare le configurazioni audio/animazioni per facilitare test e debugging.

Figura 3.24: Vista dei quattro avatar chatbot multilingua all'interno dell'ambiente virtuale del museo. Da sinistra verso destra sono rappresentate le versioni in italiano, francese, spagnolo e inglese, ciascuna associata a un proprio prefab e sistema vocale indipendente.

3.8.3 Sincronizzazione labiale

La sincronizzazione tra audio, movimento delle labbra e dei denti degli avatar è gestita dal componente `OVRLipSyncContextMorphTargetTest`, integrato all'interno del sistema di riproduzione vocale. Questo script analizza in tempo reale l'audio generato da Amazon Polly ed estrae una sequenza di fonemi, successivamente convertiti nei corrispondenti visemi Oculus.

Ogni visema viene poi associato a un blend shape del modello 3D attraverso la struttura `VisemeToBlendTarget`, configurabile tramite l'inspector di Unity. Nel progetto, la configurazione degli indici è stata calibrata in base all'ordine e alla disponibilità dei blend shapes nei modelli generati con Ready Player Me.

In particolare:

- L'indice `0` è riservato a `MouthOpen`, condiviso anche con la mesh secondaria dei denti, che si muove in sincronia con l'apertura della bocca.
- I visemi Oculus sono mappati sugli indici da `1` a `15`, corrispondenti a specifici fonemi.
- Gli indici successivi (`16`, `17`, `18`) sono riservati ad altre espressioni facciali non vocali, come i sorrisi o il movimento delle palpebre per simulare la chiusura degli occhi.

La tabella seguente riassume la corrispondenza tra visemi e blend shapes principali utilizzati nel progetto:

Indice	**Blend Shape**	**Utilizzo**
0	MouthOpen	Movimento apertura bocca e controllo denti
1	Sil	Visema silenzio
2	PP	Visema 'P' e 'B'
3	FF	Visema 'F'
4	TH	Visema 'TH'
5	DD	Visema 'D' e 'T'
6	kk	Visema 'K' e 'G'
7	CH	Visema 'CH' e 'J'
8	SS	Visema 'S'
9	nn	Visema 'N'
10	RR	Visema 'R'
11	aa	Visema 'A' aperta
12	E	Visema 'E'
13	ih	Visema 'I'
14	oh	Visema 'O'
15	ou	Visema 'U'
16	MouthSmile	Espressione sorriso
17	EyeBlinkLeft	Battito ciglia sinistro
18	EyeBlinkRight	Battito ciglia destro

Tabella 3.9: Mappatura tra visemi Oculus, blend shapes dei modelli Ready Player Me e loro funzione nel sistema di animazione.

È fondamentale che l'array `VisemeToBlendTarget` sia configurato correttamente nell'inspector di Unity, in modo che ogni visema sia associato all'indice corretto del blend shape nel modello 3D. Un disallineamento tra gli indici presenti in questo array e la reale posizione dei blend shapes può causare animazioni errate o incoerenti, compromettendo la qualità visiva complessiva della sincronizzazione facciale.

Il sistema si attiva automaticamente all'avvio della riproduzione audio e si disattiva al termine, garantendo transizioni fluide e prevenendo movimenti indesiderati durante i momenti di silenzio.

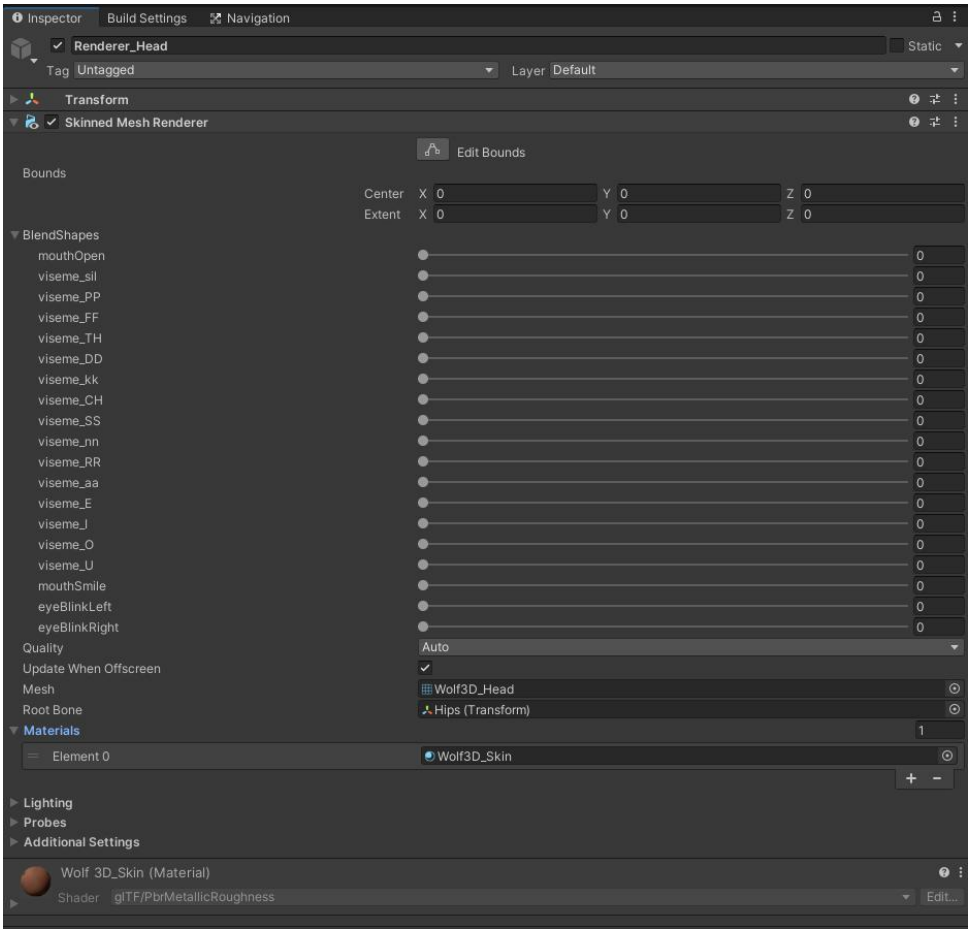


Figura 3.25: Inspector di Unity relativo allo *Skinned Mesh Renderer* dell'*head*, con la lista dei blend shape configurati e utilizzabili nello script `OVRLipSyncContextMorphTargetTest`.

Come illustrato in Figura 3.25, lo *Skinned Mesh Renderer* associato al `Renderer_Head` contiene l'intera lista di blend shapes necessari per la sincronizzazione labiale e l'animazione facciale. Anche il `Renderer_Teeth`, che che permette la visualizzazione dei denti, dispone di un proprio *Skinned Mesh Renderer* con i medesimi blend shapes, garantendo coerenza nei movimenti dell'intera area orale.

Altri *renderer* presenti nel modello 3D, dedicati a parti come occhi, capelli, corpo o abbigliamento, possiedono a loro volta dei *Skinned Mesh Renderer*, ma spesso non includono blend shapes, oppure ne contengono solo un sottoinsieme destinato a scopi decorativi o espressivi secondari.

Lo script `OVRLipSyncContextMorphTargetTest` è stato impiegato principalmente per i movimenti di bocca e denti, ma la sua struttura modulare consente un'estensione anche ad altri *Skinned Mesh Renderer*, nel caso si vogliano integrare ulteriori animazioni facciali o corporee basate su blend shapes.

3.8.4 Animazioni corporee: movimento e idle

Oltre all'animazione facciale, gli avatar sono dotati di animazioni del corpo per due stati principali:

- **Idle**: quando il chatbot è in ascolto, assume una postura rilassata e mantiene lo sguardo fisso o leggermente variabile, simulando uno stato di attenzione.
- **Speaking**: durante la fase di risposta, ovvero mentre viene riprodotto l'audio, viene attivata un'animazione con movimenti leggeri delle braccia e gestualità che accompagna il parlato, contribuendo a renderlo più credibile.

Queste animazioni sono attivate tramite i trigger `Idle` e `Talking`, inviati dallo script `TextToSpeech` in corrispondenza degli stati audio. La loro gestione avviene tramite un `Animator Controller` sincronizzato con gli eventi `OnAudioStateChanged` generati dal modulo di sintesi vocale, che permette transizioni fluide e coerenti tra i due stati animati.

Lo schema riportato in figura 3.26 illustra la struttura dell'Animator Controller, evidenziando gli stati e i trigger utilizzati per la gestione delle animazioni corporee.

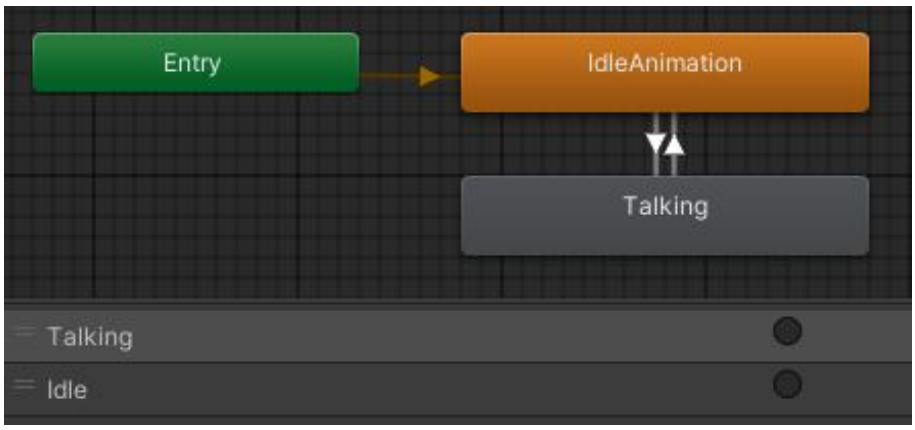


Figura 3.26: Schema dell'Animator Controller con gli stati `Idle` e `Talking` e i trigger per le animazioni corporee.

3.8.5 Animazioni oculari

Il battito delle palpebre (`eyeBlinkLeft`, `eyeBlinkRight`) e i micro-movimenti oculari sono gestiti da uno script dedicato fornito da Ready Player Me, chiamato `EyeAnimationHandler`.

Questo script genera un pattern pseudocasuale di movimento delle palpebre e degli occhi, configurabile tramite parametri esposti nell'inspector di Unity, come la frequenza e l'ampiezza del battito. Tale gestione dinamica contribuisce a evitare un effetto statico o innaturale durante le fasi di inattività dell'avatar, migliorando il realismo e la naturalezza dell'animazione.

La figura 3.27 mostra lo script `EyeAnimationHandler` nell'inspector, con i principali parametri di configurazione disponibili.

Figura 3.27: Screenshot dell'inspector di Unity relativo allo script `EyeAnimation Handler`, con i parametri configurabili per la gestione del battito delle palpebre e micro-movimenti oculari.

3.8.6 Vantaggi del sistema adottato

Questa architettura consente:

- Elevata espressività a basso costo computazionale.
- Adattabilità multilingua senza duplicazione di codice o logica.
- Sincronizzazione efficace tra parlato, animazione labiale e gestualità.
- Esperienza immersiva e coerente anche in ambienti XR con risorse limitate.

3.9 Sistema di logging interno

Per garantire una supervisione continua e facilitare le attività di debugging, il progetto implementa un sistema centralizzato di logging tramite il componente `ChatbotLog.cs`. Questo modulo rappresenta un nodo di raccolta e visualizzazione dei messaggi generati dai principali sottosistemi del chatbot: riconoscimento vocale, interazione con LLM, sintesi vocale, generazione del prompt, eventi di stato e gestione degli errori.

3.9.1 Architettura e funzionamento

Il componente `ChatbotLog.cs` è stato progettato per garantire semplicità d'uso e facile integrazione nel sistema, evitando dipendenze rigide o strutture complesse. Non si comporta come un logger passivo, ma risponde direttamente alle chiamate da parte degli altri script tramite metodi pubblici, in particolare `AddLog()`, accessibile tramite l'istanza singleton.

Gli script principali, come `ChatGPT.cs`, `TextToSpeech.cs` o `SpeechToText.cs`, invocano questo metodo per registrare eventi, specificando parametri quali colore, stile del testo (grassetto, corsivo), livello di importanza e gestione della cronologia.

Questa architettura diretta offre diversi vantaggi:

- **Chiarezza e tracciabilità**: ogni messaggio è generato esplicitamente nel punto di interesse, facilitando debug e manutenzione.
- **Controllo visivo personalizzato**: i messaggi possono essere evidenziati con colori e stili per segnalare criticità o informazioni rilevanti.
- **Assenza di dipendenze complesse**: lo script non richiede pattern come *observer* o sistemi di messaggistica, semplificando l'integrazione.

Nel progetto del museo virtuale, ogni stand dispone di un canvas XR dedicato, attivabile o disattivabile a seconda del contesto. `ChatbotLog.cs` supporta un numero arbitrario di canvas in parallelo, replicando in tempo reale gli output su ciascun pannello attivo. Questo rende il sistema adatto a configurazioni modulari e distribuite, facilmente integrabile anche in altri progetti XR che utilizzino componenti `TextMeshProUGUI` per la visualizzazione.

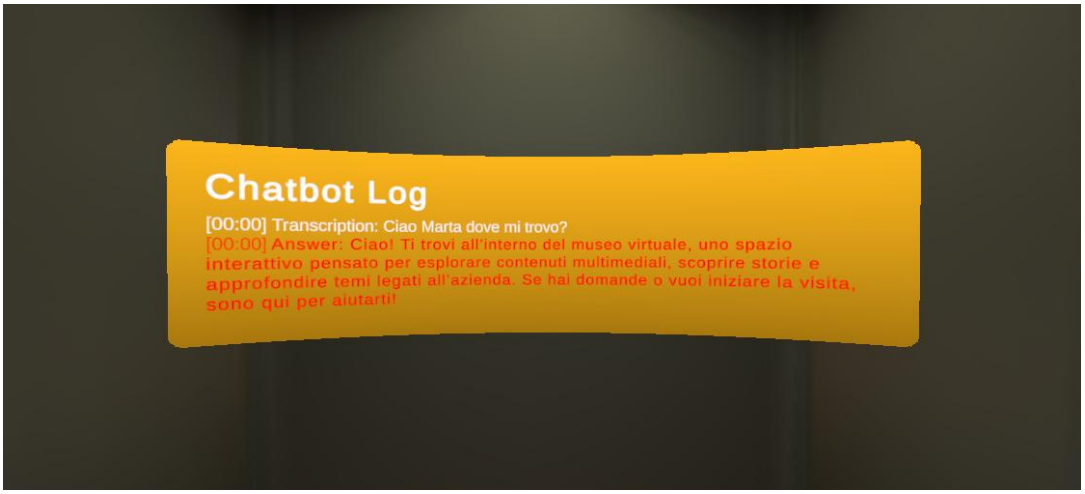


Figura 3.28: Esempio di interazione registrata nel `ChatbotLog` della reception, con una domanda dell'utente e la risposta generata dal sistema. Il pannello mostra la formattazione testuale configurata dinamicamente, inclusi colore, stile e timestamp.

Il comportamento del logger è personalizzabile tramite vari parametri esposti nell'interfaccia *Inspector* dell'editor di Unity. Questa modalità di configurazione agevola una rapida iterazione e consente di adattare facilmente il sistema a diversi scenari applicativi, come dimostrazioni pubbliche, sessioni di debugging o ambienti multilingua, senza necessità di modificare il codice sorgente.

I parametri disponibili influenzano aspetti differenti, quali la gestione automatica dell'attivazione e disattivazione del canvas, il numero massimo di messaggi visualizzabili, la pulizia del log al cambio lingua, la filtrazione dei messaggi in base alla loro rilevanza, e il controllo sulla visualizzazione di messaggi duplicati o secondari.

Parametro	Descrizione
`canvasOnChatbotStateChange`	Abilita o disabilita la gestione automatica della visibilità del canvas del log in risposta ai cambiamenti dello stato del chatbot (ad esempio, quando il chatbot viene disattivato, il canvas viene nascosto).
`onlyChangeCanvaStatusOnInactive`	Se attivo, il canvas del log viene disattivato solo quando lo stato del chatbot diventa *Inactive*; in caso di cambio a stato *Active*, il canvas non viene attivato automaticamente, lasciando il controllo manuale dell'attivazione.
`resetCanvasOnLanguageChanged`	Cancella automaticamente il contenuto del log ogni volta che viene selezionata una nuova lingua. Utile per separare chiaramente sessioni in lingue differenti.
`maxLogEntries`	Specifica il numero massimo di messaggi mantenuti nel log. I messaggi più vecchi vengono rimossi automaticamente per mantenere prestazioni e leggibilità.
`showOnlyImportantMessages`	Se attivo, visualizza solo i messaggi contrassegnati come importanti, filtrando quelli informativi o secondari.

Tabella 3.10: Parametri configurabili tramite *Inspector* nello script `ChatbotLog.cs`.

3.9.2 Utilità e impieghi

Il sistema `ChatbotLog.cs` si è rivelato fondamentale durante tutte le fasi di sviluppo, test e validazione del sistema. Le sue funzionalità hanno supportato l'osservabilità interna del sistema e il monitoraggio del comportamento nei seguenti scenari:

- **Diagnosi rapida di malfunzionamenti**: grazie ai colori e alla categorizzazione, è possibile individuare a colpo d'occhio problemi legati a timeout, assenza di audio, errori di rete o parametri errati.
- **Monitoraggio dei flussi interni**: fornisce una visione coerente e continua dell'interazione tra moduli, consentendo una verifica immediata dei cambiamenti di stato o delle risposte del modello.
- **Verifica della correttezza logica del comportamento**: in fase di debugging avanzato, permette di controllare la coerenza tra eventi (es. ordine corretto tra registrazione, trascrizione, risposta e sintesi).

- **Supporto allo sviluppo iterativo**: consente test veloci e mirati anche in ambiente editor, senza necessità di strumenti di debugging esterni.
- **Formattazione e controllo avanzato**: messaggi personalizzati per colore, stile, categoria e timestamp, supporto alla compressione dei duplicati e gestione del numero massimo di elementi visualizzati.
- **Multi-canvas sincronizzato**: permette la visualizzazione coerente e parallela dei messaggi su più pannelli attivi nella scena, rendendo il sistema adatto ad ambienti XR multi-stand.

Nel complesso, `ChatbotLog.cs` rappresenta uno strumento essenziale per lo sviluppo robusto e scalabile del sistema conversazionale, contribuendo in modo significativo alla sua osservabilità e manutenibilità.

3.10 Interfaccia grafica e componenti interattivi

L'interfaccia grafica (GUI) rappresenta uno dei principali canali di interazione dell'utente all'interno dell'ambiente virtuale. Sebbene il chatbot sia sviluppato specificamente per contesti XR/VR, la logica di gestione dei bottoni è stata progettata per essere completamente indipendente dalla modalità concreta con cui questi vengono implementati nel mondo virtuale. Che si tratti di interazioni basate su raycasting, collider, gesture o altri sistemi forniti da SDK differenti (come XR Interaction Toolkit, Meta SDK o Oculus Integration), è sufficiente che i bottoni siano in grado di generare eventi. Tali eventi vengono poi utilizzati per richiamare i metodi pubblici esposti dallo script `ButtonsManager`, garantendo così una separazione netta tra la logica funzionale e il meccanismo di attivazione.

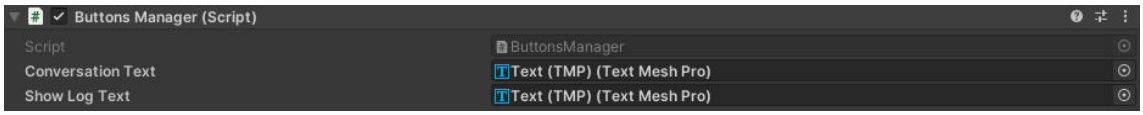


Figura 3.29: Lo script `ButtonsManager.cs` visualizzato nell'Inspector di Unity.

3.10.1 Struttura generale dell'interfaccia

L'interfaccia grafica del sistema è costituita da un pannello fluttuante che raccoglie sei bottoni interattivi e alcune brevi descrizioni testuali, pensate per facilitare l'interazione da parte dell'utente. Il pannello è posizionato in modo da risultare facilmente accessibile all'interno della scena virtuale, senza ostacolare l'esperienza immersiva.

Nel contesto specifico del museo virtuale, è previsto un pannello per ogni stand, quindi per ogni chatbot presente nell'ambiente. Ogni pannello è localizzato nelle vicinanze dell'avatar di riferimento e offre all'utente gli strumenti necessari per interagire in modo diretto e intuitivo con il sistema.

Tuttavia, come già sottolineato in sezioni precedenti, l'intera struttura è stata progettata per essere modulare e facilmente adattabile a contesti XR differenti. In altri progetti, ad esempio, potrebbe essere sufficiente un unico pannello centrale, oppure un pannello con un numero di bottoni diverso, per esempio nel caso in cui siano supportate meno lingue. Tale flessibilità è resa possibile dal fatto che i bottoni non sono vincolati a una specifica implementazione grafica, ma si limitano a generare eventi che invocano i metodi pubblici esposti dallo script `ButtonsManager`.

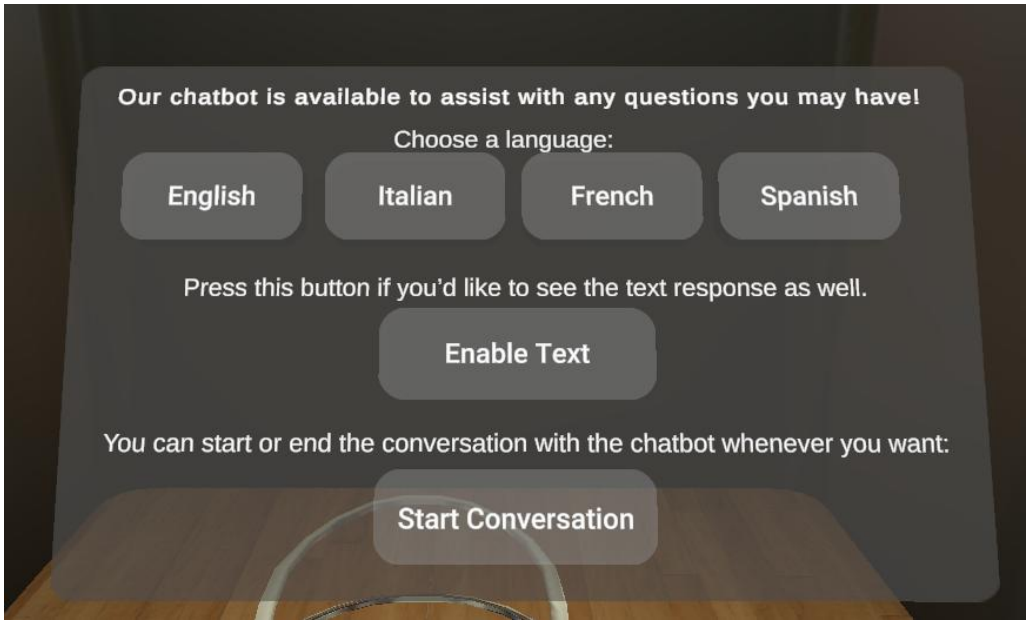


Figura 3.30: Schema della disposizione dei bottoni all'interno dell'interfaccia del museo virtuale.

3.10.2 Funzionalità dei bottoni

Selezione della lingua

Il sistema include quattro bottoni, ciascuno associato a una delle lingue supportate: italiano, inglese, francese e tedesco. Alla pressione, viene invocata la funzione `UpdateLanguage(string language)` dello script `ButtonsManager`, passando la lingua selezionata come parametro. L'informazione viene propagata tramite il singleton `ChatbotGlobalData`, aggiornando in tempo reale sia il prompt destinato al modello linguistico, sia la voce sintetizzata della risposta.

Avvio e interruzione dell'interazione

Un altro bottone consente di attivare o interrompere il dialogo vocale con il chatbot. Il testo visualizzato alterna dinamicamente tra `Start Conversation` e `Stop Conversation`, in base allo stato corrente. Questo comportamento è gestito dalla funzione `UpdateChatbotState()`, che aggiorna il valore della variabile `chatbotState` all'interno del singleton `ChatbotGlobalData`.

Visualizzazione del log testuale

È presente inoltre un bottone che consente di mostrare o nascondere il log della conversazione, utile per finalità diagnostiche o per migliorare l'accessibilità. La funzione `UpdateChatbotLogState()` alterna la visibilità del pannello testuale e modifica il testo del bottone tra `Enable Text` e `Disable Text`.

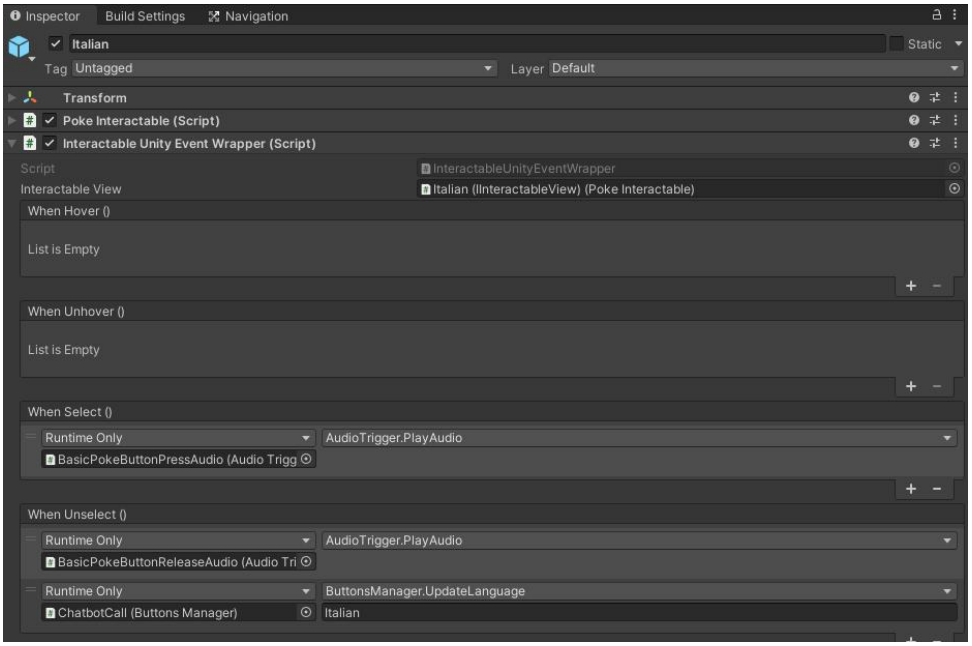


Figura 3.31: Inspector del bottone associato alla lingua `Italian`, in cui è visibile il componente `Interactable Unity Event Wrapper`[14]. Questo consente di collegare l'evento all'invocazione del metodo `UpdateLanguage` nello script `ButtonsManager`, passando come parametro la stringa `"Italian"`.

3.10.3 Gestione centralizzata tramite ButtonsManager

Tutti i bottoni sono connessi a uno script centrale chiamato `ButtonsManager.cs`. Questo script viene inizializzato nel metodo `Start()` dove recupera i riferimenti ai singleton globali, tra cui `ChatbotGlobalData` e `ChatbotLog`. Il metodo `UpdateUI()` viene eseguito ad ogni frame per mantenere aggiornati i testi e gli stati dei bottoni in base allo stato globale del sistema.

[14]La stessa logica è applicata anche agli altri bottoni dell'interfaccia: ciascuno attiva un metodo pubblico dello script `ButtonsManager`, eventualmente con parametri diversi a seconda della funzione desiderata.

Vantaggi della gestione centralizzata

L'utilizzo di uno script centralizzato per la gestione dei bottoni offre numerosi benefici progettuali:

- **Modularità**: consente di separare la logica dalla rappresentazione grafica, facilitando la sostituzione o l'adattamento dell'interfaccia in contesti diversi.
- **Tracciabilità**: ogni cambiamento di stato viene tracciato e inviato al sistema di logging testuale.
- **Robustezza**: lo script effettua controlli di validità per prevenire errori legati a riferimenti nulli o non assegnati.

3.10.4 Comportamento dinamico e sincronizzazione UI

La GUI viene mantenuta coerente con lo stato interno del sistema grazie all'aggiornamento continuo nel metodo `UpdateUI()`. In questo modo, anche modifiche allo stato effettuate da moduli esterni (es. script che disattivano il chatbot in modo autonomo) si riflettono immediatamente sulla rappresentazione grafica.

3.10.5 Adattabilità dell'interfaccia in contesti XR

Sebbene l'interfaccia sia stata progettata specificamente per ambienti VR/XR, è stata mantenuta una certa flessibilità per favorire l'adattamento a diversi dispositivi e configurazioni. In particolare:

- I bottoni possono essere attivati tramite i sistemi di interazione forniti dai principali SDK XR (come Oculus Integration, XR Interaction Toolkit o Meta SDK), sfruttando collider, raycast o interazioni gestuali.
- La logica di interazione è indipendente dalla rappresentazione visiva dei bottoni, permettendo di sostituirli con elementi 3D personalizzati, mantenendo inalterato lo script `ButtonsManager`.
- L'interfaccia può essere facilmente modificata per supportare un numero diverso di lingue o opzioni, senza richiedere cambiamenti alla struttura del codice.

Questo approccio consente di riutilizzare la logica dei bottoni anche in varianti del progetto destinate a visori o contesti XR differenti, garantendo coerenza funzionale e semplicità di manutenzione.

4. Valutazione dei risultati e iterazioni di sviluppo

Questo capitolo presenta un'analisi critica delle prestazioni e del comportamento dell'assistente vocale all'interno di scenari d'uso realistici, con l'obiettivo di valutare l'efficacia delle soluzioni tecniche adottate e di documentare le principali scelte iterative che hanno guidato l'evoluzione del progetto. L'analisi si basa su una serie di test condotti in condizioni operative eterogenee per verificare non solo la qualità dell'interazione, ma anche la stabilità, la latenza e la capacità del sistema di gestire errori, interruzioni e casi limite.

Figura 4.1: Chatbot in azione durante una sessione di interazione vocale nell'applicazione standalone su dispositivo *Meta Quest*.

Parallelamente, vengono discusse le criticità emerse in fase di sviluppo e test, mettendo in luce le aree in cui l'architettura può essere migliorata. In questo modo, l'analisi offre non solo una riflessione sul lavoro svolto, ma anche uno spunto concreto per orientare i futuri sviluppi.

Sono stati previsti due principali contesti di utilizzo:

- **Contesti controllati**, caratterizzati da condizioni ottimali per l'esecuzione dell'applicazione. In questi casi, i visori *Meta Quest* erano collegati al PC tramite *Meta Link*, oppure operavano in modalità standalone su una rete locale stabile. L'ambiente era silenzioso, privo di disturbi acustici o visivi significativi, e con un'interazione gestita direttamente dallo sviluppatore o da utenti informati sul funzionamento del sistema;
- **Contesti non controllati**, come uffici condivisi, ambienti aziendali o spazi aperti al pubblico, nei quali è stata privilegiata la modalità standalone[1]. In tali contesti, l'assistente era soggetto a fonti di rumore ambientale, connessioni Wi-Fi potenzialmente instabili, interazioni meno prevedibili da parte degli utenti e, più in generale, a un grado maggiore di incertezza operativa.

Per ottenere una valutazione il più possibile rappresentativa, sono stati coinvolti utenti con background eterogenei, includendo sia figure tecniche, come sviluppatori e professionisti IT, sia profili non specialistici. Questo ha permesso di osservare la naturalezza dell'interazione e la percezione dell'esperienza in un'ottica più ampia.

[1]Con "build standalone" si intende il pacchetto eseguibile dell'applicazione Unity, compilato per l'esecuzione autonoma su visore *Meta Quest*, senza necessità di connessione al PC.

4.1 Strategia di test e metodologia di validazione

4.1.1 Obiettivi e approccio

L'obiettivo principale delle attività di valutazione è stato analizzare il comportamento dell'assistente vocale in condizioni d'uso realistiche, con particolare attenzione a tre aspetti chiave: reattività, robustezza e adattamento a contesti variabili. L'analisi ha considerato l'interazione complessiva tra utente e sistema, valutando sia indicatori tecnici come la latenza, la stabilità e l'accuratezza della trascrizione, sia aspetti soggettivi legati all'esperienza d'uso.

È stato adottato un approccio empirico e iterativo, articolato in cicli successivi di osservazione, intervento e verifica. Le prove sono state condotte sia all'interno dell'editor, tramite *Oculus Link*, sia tramite build standalone su dispositivi *Meta Quest 2* e *3*, in condizioni operative eterogenee: reti aziendali, connessioni domestiche e hotspot mobili, con variazioni intenzionali di lingua, accento e rumore ambientale.

Le valutazioni hanno incluso test funzionali mirati, volti a verificare la risposta del sistema in termini di latenza, sincronizzazione tra moduli e gestione degli stati, ma anche sessioni esplorative più libere. In queste ultime, gli utenti erano incoraggiati a interagire in modo spontaneo, formulando domande non solo pertinenti al contesto museale, ma anche volutamente ambigue, fuori tema o prive di senso, al fine di osservare la resilienza semantica del modello linguistico e la gestione delle eccezioni. Questo approccio misto ha consentito di raccogliere dati significativi tanto sull'affidabilità tecnica quanto sulla coerenza e continuità dell'esperienza conversazionale.

4.1.2 Ambiente e configurazione di test

All'interno del museo virtuale, ciascuna area espositiva era presidiata da un'istanza del chatbot, collocata dietro un apposito bancone interattivo. Gli utenti potevano esplorare liberamente l'ambiente, avvicinarsi agli stand e attivare il chatbot per porre domande relative ai contenuti esposti, all'azienda o per testare il comportamento del sistema in modo spontaneo.

L'assistente virtuale era predisposto in quattro versioni linguistiche, ciascuna con un avatar dedicato e una voce coerente, per garantire uniformità tra lingua, aspetto e timbro. Le sperimentazioni hanno coperto una vasta gamma di scenari, al fine di verificare la robustezza e l'adattabilità dell'intero sistema conversazionale.

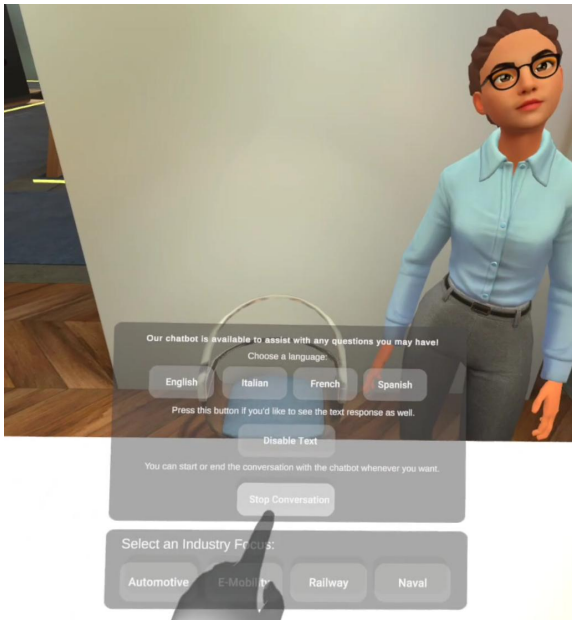


Figura 4.2: Esempio di interazione con il chatbot tramite pulsanti grafici.

In particolare, sono stati testati:

- **Attivazione e disattivazione dinamica** in zone diverse, per osservare il comportamento di spawn, despawn e aggiornamento contestuale del chatbot.
- **Cambio della lingua a runtime**, tra diverse interazioni e persino durante una singola risposta, con attenzione alla propagazione dello stato linguistico.
- **Interazioni rapide e frequenti**, pensate per stressare la pipeline vocale e la sincronizzazione tra moduli.
- **Transizioni tra zone XR**, per valutare la coerenza tra ambiente virtuale e contenuti generati.
- **Compatibilità con dispositivi diversi** (Meta Quest 2 e 3), sia in modalità editor che standalone.
- **Connessioni di rete instabili**, per testare la resilienza delle chiamate API in condizioni di latenza variabile.
- **Condizioni acustiche complesse**, con accenti, inflessioni vocali e rumori di fondo.
- **Carico grafico del chatbot**, analizzato attraverso test comparativi su shader e profondità delle texture, al fine di ottimizzare le prestazioni grafiche senza compromettere la qualità visiva e la fluidità delle animazioni.

4.1.3 Criteri di osservazione e analisi

Oltre agli aspetti quantitativi legati ai tempi di risposta e alla stabilità, una parte centrale della valutazione ha riguardato anche l'osservazione qualitativa di anomalie ricorrenti, come errori di trascrizione, assenza di animazioni, bug visivi o incongruenze semantiche. Tali osservazioni hanno costituito una base concreta per interventi mirati su parametri di configurazione, logica degli script, struttura dei prefab del chatbot e architettura dei moduli.

I principali criteri osservati sono stati:

- **Qualità della trascrizione vocale**, analizzata caso per caso e migliorata attraverso soglie personalizzate, filtri dedicati e prompt di controllo;
- **Tempo di risposta**, inteso come intervallo percepito tra input vocale e avvio dell'audio sintetizzato;
- **Stabilità del sistema**, valutata in scenari con frequenti cambi di lingua o di stato, input vocali non chiari o incompleti, e situazioni di interazione non previste;
- **Coerenza multimodale**, in riferimento alla sincronizzazione tra voce, movimenti facciali e postura dell'avatar;
- **Esperienza utente**, rilevata tramite feedback qualitativi su naturalezza, fluidità e comprensibilità dell'interazione.

Sulla base di questi criteri, sono stati progressivamente ottimizzati diversi aspetti del sistema, tra cui le soglie di attivazione, i parametri di filtraggio, i timeout, la durata massima delle registrazioni vocali (sia totali che parziali), e la struttura dei prompt inviati al modello linguistico. Ogni modifica è stata validata attraverso test comparativi, con l'obiettivo di trovare un equilibrio efficace tra reattività, robustezza e qualità dell'interazione.

Il risultato è una configurazione stabile e fluida, in grado di mantenere prestazioni affidabili anche in condizioni ambientali sfavorevoli o in presenza di comportamenti utente poco prevedibili.

4.2 Sviluppo incrementale e approccio iterativo

Lo sviluppo del sistema non ha seguito un processo lineare o rigidamente pianificato, ma si è basato su un approccio empirico e iterativo, caratterizzato da cicli frequenti di test, osservazione e modifica. Ogni funzionalità è stata aggiunta, ottimizzata o rifinita attraverso numerosi tentativi, risoluzioni di bug e micro-adattamenti, spesso derivati da comportamenti imprevisti o problematici rilevati durante l'utilizzo reale.

4.2.1 Fasi di sviluppo principali

Il lavoro si è articolato in più fasi, ciascuna delle quali ha introdotto elementi tecnici e logici progressivamente più complessi.

Le principali tappe possono essere così sintetizzate:

1. **Prototipazione iniziale**: in questa fase è stata definita l'architettura di base del sistema, realizzando un prefab "Chatbot" multilingua progettato per supportare, in futuro, un'interazione vocale. In questa prima versione, tuttavia, l'interazione era esclusivamente testuale: l'input veniva inserito manualmente tramite una casella di testo, mentre le risposte generate dal modello ChatGPT erano visualizzate su un canvas di debugging. Non era ancora presente alcuna forma di *prompt engineering*, e il sistema si limitava a inoltrare direttamente l'input dell'utente al modello, senza filtri né struttura contestuale. L'obiettivo principale era verificare il corretto funzionamento della pipeline di base, validando la logica applicativa e la struttura modulare del sistema in vista della successiva integrazione delle componenti vocali. In parallelo, è stato introdotto un avatar realizzato con la piattaforma Ready Player Me, insieme agli elementi grafici dell'interfaccia, come pulsanti e canvas di stato, utili per monitorare la conversazione e gestire l'attivazione del chatbot.

2. **Integrazione delle tecnologie vocali**: si è proceduto al collegamento con le API di Whisper per il riconoscimento vocale automatico (ASR) e Amazon Polly per la sintesi vocale (TTS). Questa integrazione ha permesso la comunicazione bidirezionale vocale tra utente e chatbot, fondamentale per un'interazione immersiva in XR. Durante questa fase sono state affrontate criticità iniziali quali la latenza tra input vocale e risposta sintetizzata e la gestione asincrona delle chiamate API, che hanno richiesto l'implementazione di meccanismi di timeout base e sincronizzazione per evitare blocchi o ritardi. Contestualmente, è stata sviluppata una prima versione, ancora embrionale, della logica di prompt

engineering, per migliorare la coerenza delle risposte generate dal modello. Sono stati inoltre introdotti i primi filtri per escludere input non significativi e una sincronizzazione di base tra parlato e animazioni facciali, oltre ad aggiornamenti dell'interfaccia utente per visualizzare in tempo reale lo stato delle richieste vocali, facilitando il debugging e il monitoraggio del sistema.

3. **Gestione spaziale e coerenza semantica**: una parte rilevante del lavoro è stata dedicata alla definizione di un sistema di spawn dinamico degli avatar, in grado di attivarsi nella posizione corretta in base alla zona visitata dall'utente. Questa logica non serviva solo a collocare l'avatar nel punto giusto dello spazio XR, ma soprattutto a garantire che il prompt inviato al modello linguistico riflettesse il contesto semantico appropriato. La posizione dell'utente era infatti associata a uno specifico settore (es. automotive, energia, ecc), che influenzava la formulazione delle richieste e migliorava la coerenza delle risposte. La corretta gestione dello stato tra attivazione, cambio zona e despawn ha richiesto numerosi affinamenti, sia sul piano tecnico che logico. In parallelo, sono state implementate le funzionalità di rotazione dell'avatar verso l'interlocutore, le animazioni di speaking e idle, e ottimizzata la sincronizzazione tra parlato e animazioni facciali, con il risultato di un'interazione più naturale e coinvolgente.

4. **Estensione multilingua**: per garantire una maggiore coerenza tra lingua, voce e identità visiva, è stata adottata una strategia basata su avatar distinti per ciascuna lingua supportata. Ogni avatar era caratterizzato da un nome, una voce e un'estetica pensati per rispecchiare la cultura o la provenienza linguistica di riferimento, così da rendere l'interazione più credibile e immersiva. Questo approccio ha comportato una duplicazione controllata degli asset, gestita tramite una logica centralizzata che permette di attivare e sincronizzare dinamicamente l'avatar corretto in base alla lingua selezionata. La soluzione ha facilitato anche l'estensione futura ad altre lingue, mantenendo ordine e manutenibilità del sistema. In parallelo, sono stati adattati i componenti dell'interfaccia per supportare il cambio lingua in tempo reale e migliorare la personalizzazione dell'esperienza.

5. **Raffinamento della logica conversazionale**: la pipeline vocale è stata progressivamente perfezionata con una serie di miglioramenti significativi. Sono stati introdotti meccanismi avanzati di timeout per evitare attese indefinite, filtri più sofisticati per escludere prompt inutili o frasi vuote, e un sistema di fallback che gestisce i silenzi o le mancate risposte dell'utente. Particolare attenzione è stata rivolta alla gestione dinamica del cambio lingua, che è stata resa robusta per garantire transizioni fluide e senza interruzioni nel corso della conversazione. Inoltre, il prompt engineering è stato oggetto di numerose ottimizzazioni per migliorare la pertinenza e la qualità delle risposte generate dall'LLM. Ogni criticità emersa ha condotto a interventi mirati sulla logica di gestione e sui parametri del sistema, contribuendo ad aumentare l'affidabilità e la qualità complessiva dell'esperienza conversazionale. Parallelamente, gli elementi dell'interfaccia utente sono stati aggiornati con nuovi feedback visivi e strumenti di debug più efficaci, per facilitare la diagnosi e la risoluzione di eventuali problemi in tempo reale.

4.2.2 Sperimentazione e affinamento iterativo

Molte delle decisioni progettuali sono emerse direttamente dall'uso pratico del sistema, secondo un paradigma "try and error". L'approccio ha privilegiato la sperimentazione continua rispetto alla pianificazione teorica, con modifiche testate direttamente in ambiente XR per valutarne l'impatto sulla qualità percepita.

Questo metodo ha comportato un numero elevato di iterazioni ma ha permesso di adattare progressivamente il sistema alle reali esigenze dell'interazione immersiva, producendo una pipeline più solida e affidabile rispetto a una soluzione sviluppata esclusivamente su carta.

4.3 Prestazioni e osservazioni tecniche

Questa sezione riassume le prestazioni osservate durante la fase di test, con particolare attenzione a quattro aspetti principali: la velocità di risposta, la precisione del riconoscimento vocale, la pertinenza delle risposte generate e la sincronizzazione tra parlato e animazioni.

4.3.1 Tempi medi di risposta e latenza

Il tempo complessivo tra il termine dell'input vocale e la conclusione della risposta audio generata da Amazon Polly si è dimostrato variabile in base a diversi fattori: stabilità della rete, lunghezza e complessità della richiesta, e carico computazionale.

- In scenari favorevoli (rete stabile e frasi brevi), la latenza media si è attestata tra **3 e 5 secondi**.
- In condizioni più critiche (rete instabile e/o richieste articolate), sono stati registrati picchi fino a **6–7 secondi**.

Per preservare la fluidità dell'interazione, si è introdotto un messaggio vocale ("Sto elaborando la risposta...") in caso di attese prolungate. Questa soluzione ha contribuito a evitare che l'utente percepisse il sistema come inattivo, migliorando la continuità dell'esperienza.

4.3.2 Qualità della trascrizione e coerenza delle risposte

L'accuratezza del riconoscimento vocale è risultata complessivamente soddisfacente, anche se influenzata da fattori quali la qualità del microfono integrato, il rumore ambientale, la lingua selezionata, l'accento dell'utente e la complessità linguistica delle frasi.

Le valutazioni si basano su un ampio numero di sessioni tecnico-funzionali, condotte in ambienti differenti e su dispositivi *Meta Quest* di più generazioni. A queste si sono affiancate interazioni esplorative con un ristretto gruppo di utenti. Sebbene il campione non sia esteso, tali prove hanno evidenziato tendenze significative, utili a guidare il processo di ottimizzazione.

Le osservazioni più rilevanti includono:

- In ambienti controllati, la trascrizione è generalmente corretta, anche in presenza di accenti non nativi.
- In contesti moderatamente rumorosi (come open space o uffici condivisi), si verificano più spesso trascrizioni parziali o imprecise, specie con parole tecniche, nomi propri o termini rari.
- L'integrazione di filtri correttivi e dizionari personalizzati ha contribuito a mitigare queste problematiche.

Le attività di test hanno incluso:

- La valutazione della robustezza della trascrizione in presenza di rumore di fondo a intensità variabile;
- Il confronto tra dispositivi differenti (*Meta Quest 2* e *Meta Quest 3*);
- L'analisi delle prestazioni rispetto a voci con caratteristiche diverse (genere, accento, timbro, ritmo di eloquio).

Anche la qualità delle risposte generate dal modello linguistico è stata oggetto di attenta valutazione. Numerose iterazioni sul prompt engineering hanno permesso di ottenere risposte più mirate, concise e coerenti con l'intento dell'utente. I risultati sono stati nel complesso positivi: il chatbot ha generalmente fornito risposte pertinenti, e in presenza di input ambigui o incompleti ha reagito richiedendo chiarimenti oppure offrendo interpretazioni compatibili. Tale comportamento è stato ritenuto soddisfacente in termini di affidabilità e sicurezza dell'interazione.

Nel complesso, le osservazioni confermano che reattività e qualità della conversazione dipendono strettamente dal coordinamento asincrono tra STT, LLM e TTS. L'inserimento di messaggi vocali di attesa e il bilanciamento tra reattività e precisione si sono dimostrati elementi determinanti per garantire un'interazione fluida.

4.3.3 Valutazione delle animazioni facciali e corporee

Le animazioni facciali, sincronizzate con l'audio generato e basate sui dati forniti dal modulo `OVRLipSyncContextMorphTargetTest`, hanno restituito un risultato visivo credibile. In particolare, il movimento delle labbra e la sincronizzazione con i fonemi, insieme all'animazione dei denti tramite la mesh secondaria, sono stati giudicati efficaci e sufficientemente realistici per l'interazione in ambiente immersivo.

Tuttavia, le animazioni corporee secondarie, come i movimenti idle o il gesto delle braccia durante l'attesa o l'interlocuzione, hanno mostrato margini di miglioramento. Pur svolgendo la loro funzione di evitare una postura statica e innaturale dell'avatar, tali movimenti sono apparsi in alcuni casi ripetitivi o poco armonizzati con il contesto dialogico.

Nel complesso:

- Le animazioni facciali sono risultate soddisfacenti, in particolare per la sinergia tra fonazione e movimento della bocca.
- Le animazioni fisiche e posturali sono state percepite come accettabili ma suscettibili di perfezionamento, specialmente in termini di varietà, ritmo e sincronizzazione con il contenuto verbale.

Figura 4.3: Esempio di animazione del chatbot durante la conversazione, con sincronizzazione del movimento di bocca e braccia in risposta al parlato.

4.4 Feedback raccolti e valutazione soggettiva

4.4.1 Test interni e valutazioni qualitative

La fase di validazione soggettiva del sistema si è svolta attraverso una serie di test interni condotti con colleghi, sia appartenenti al team di sviluppo sia esterni ad esso. L'obiettivo era raccogliere impressioni dirette sull'usabilità, la reattività e la qualità percepita dell'interazione vocale all'interno di uno scenario immersivo realistico.

I partecipanti hanno testato il sistema utilizzando dispositivi XR (Meta Quest 2 e Meta Quest 3), interagendo liberamente con il chatbot multilingua all'interno dell'ambiente virtuale museale. Le interazioni includevano sia domande guidate (predefinite per ogni zona), sia richieste spontanee su temi aziendali o generali. Il test è stato ripetuto in tutte le lingue supportate, consentendo di raccogliere feedback trasversali sull'intero sistema.

I risultati qualitativi sono stati in larga parte positivi. I colleghi hanno evidenziato una buona intuitività nell'interfaccia e una curva di apprendimento contenuta, anche per utenti non esperti. La possibilità di cambiare lingua dinamicamente è stata considerata un punto di forza, così come la coerenza tra voce, aspetto dell'avatar e lingua parlata.

La sincronizzazione tra voce e movimenti labiali è stata percepita come accurata e coinvolgente, contribuendo in modo significativo alla sensazione di interazione naturale. Sebbene le animazioni delle braccia e dell'idle risultino piuttosto semplici e migliorabili, l'animazione oculare e la postura complessiva dell'avatar hanno comunque contribuito ad aumentare il senso di presenza, riducendo in parte l'effetto di artificialità tipico di molti assistenti vocali.

Inoltre, è emersa una differenza significativa tra le aspettative degli utenti tecnici e quelle degli utenti meno esperti. I primi hanno apprezzato la struttura modulare del sistema, la possibilità di accedere a funzionalità avanzate e il controllo preciso sui parametri di interazione. I secondi, invece, hanno manifestato difficoltà nel riconoscere lo stato del chatbot (attivo, in ascolto, in elaborazione) o nel comprendere se un errore derivasse dal microfono, dalla rete o da un limite del sistema stesso. Questa discrepanza ha evidenziato l'importanza, in fase di progetto, di una comunicazione chiara dello stato conversazionale e di interfacce più accessibili.

4.4.2 Criticità emerse e spunti di miglioramento

Nonostante la generale soddisfazione, alcuni aspetti critici sono stati evidenziati dai partecipanti, rappresentando preziosi spunti per l'ottimizzazione futura:

- **Rumore ambientale:** in ambienti non perfettamente silenziosi, Whisper tendeva a generare trascrizioni parzialmente errate o incomplete. Sebbene i filtri applicati abbiano attenuato il problema, si è rilevata la necessità di tecniche di *noise suppression* più robuste o di modelli STT alternativi.

- **Gestione delle interruzioni e dei silenzi:** in caso di pause dell'utente o silenzi prolungati, il sistema non sempre rispondeva in modo prevedibile. Alcuni utenti hanno segnalato confusione rispetto allo stato dell'assistente (attivo/in ascolto/in elaborazione), suggerendo l'introduzione di segnali visivi o sonori più evidenti durante le transizioni.

- **Latenza nella risposta:** sebbene le tempistiche di risposta siano risultate generalmente accettabili, in alcuni casi è stata percepita una pausa eccessiva tra l'input vocale e l'inizio della risposta sintetica. Questo ha prodotto un leggero senso di "disconnessione" temporale, soprattutto quando l'interazione si svolgeva in modo più dinamico. La causa principale è da ricondurre alla somma dei tempi di trascrizione, elaborazione del prompt e sintesi vocale.

- **Contesto semantico:** alcuni tester hanno osservato che, in presenza di cambi di zona o di lingua durante l'elaborazione di una richiesta, la risposta risultava talvolta incoerente rispetto all'intento iniziale. Questo ha evidenziato l'importanza di meccanismi più stabili di gestione dello stato e di contesto, in grado di preservare la coerenza semantica anche in situazioni non lineari.

- **Chiarezza dell'interfaccia utente:** alcuni utenti hanno espresso difficoltà nel comprendere lo stato corrente del sistema o nel trovare modalità alternative di input in caso di problemi vocali. In particolare, è emersa una criticità nel cambio lingua, poiché l'utente non percepisce immediatamente in quale lingua il chatbot sta operando finché non inizia a parlare. Un'integrazione più chiara di elementi interattivi e indicatori visivi (come pulsanti per ripetere, cancellare o terminare l'interazione, o un segnale esplicito della lingua attiva) potrebbe migliorare significativamente l'usabilità, specialmente in contesti educativi o museali.

- **Animazioni non verbali e movimento:** sebbene il lip sync e la riproduzione vocale abbiano funzionato correttamente, sono emerse richieste per una maggiore fluidità e varietà nei movimenti dell'avatar, al fine di rendere l'interazione più naturale e coinvolgente. Miglioramenti nella gestione delle animazioni del corpo, della postura e delle espressioni facciali rappresentano un'area di sviluppo promettente.

- **Scalabilità e carico computazionale:** la gestione simultanea di più avatar multilingua, ciascuno con animazioni attive, audio spatializzato e componenti grafici complessi, ha evidenziato limiti prestazionali, in particolare su dispositivi XR meno recenti come Meta Quest 2. L'architettura modulare ha permesso una parziale mitigazione del problema, ma resta centrale il tema dell'ottimizzazione computazionale e della sostenibilità del carico in progetti XR su larga scala.

4.4.3 Percezione generale e accettazione dell'esperienza

Le valutazioni soggettive raccolte indicano che il sistema rappresenta una solida base per un assistente vocale immersivo, con un buon grado di accettazione dell'interazione conversazionale anche in ambienti XR dinamici e non controllati. L'esperienza è stata percepita come piuttosto naturale, grazie a elementi quali:

- La qualità espressiva delle voci generate da Amazon Polly, armonizzata con l'estetica degli avatar;

- La capacità di fornire risposte coerenti e fluide alle domande più frequenti, resa possibile da un prompt engineering accurato;

- Il bilanciamento tra il controllo diretto dell'utente e l'autonomia del sistema nell'orientare la conversazione.

Parallelamente, sono emersi spunti significativi per il miglioramento, in particolare riguardo alla riduzione della latenza percepita, alla maggiore robustezza in contesti acustici complessi e alla gestione più efficace delle situazioni anomale. L'introduzione di messaggi di attesa chiari, animazioni di feedback reattive e indicatori visivi dello stato operativo si è rivelata fondamentale per aumentare la trasparenza e la rassicurazione durante l'interazione.

Un aspetto trasversale emerso con forza è il ruolo della coerenza percettiva tra i canali visivo e uditivo. Elementi come il sincronismo labiale, lo sguardo rivolto all'interlocutore, la postura neutra ma attenta, e piccoli movimenti oculomotori hanno contribuito in modo determinante alla sensazione di presenza e naturalezza.

La percezione di un avatar "vivo" non è legata unicamente alla complessità tecnica delle animazioni, ma alla loro armonia con il contesto e con il comportamento vocale del sistema.

In sintesi, sebbene il sistema mostri un potenziale promettente, il percorso verso un'interazione davvero fluida e priva di attriti richiede ulteriori cicli iterativi. Il focus futuro dovrà essere posto sull'adattabilità contestuale, sulla comunicazione trasparente degli stati e sulla personalizzazione dell'esperienza in funzione delle caratteristiche e delle preferenze dell'utente.

4.5 Considerazioni per lo sviluppo futuro

Il sistema attualmente sviluppato costituisce una base solida per l'integrazione di assistenti vocali intelligenti in ambienti XR, ma l'esperienza maturata durante le fasi di progettazione, implementazione e test ha evidenziato numerose opportunità di evoluzione, sia in chiave architetturale che funzionale. In questa sezione vengono proposte una serie di linee di sviluppo future, finalizzate a migliorarne scalabilità, efficienza e adattabilità in contesti aziendali reali.

4.5.1 Modularità avanzata e ottimizzazione della pipeline

L'architettura software del sistema è già strutturata in modo modulare, con componenti principali come riconoscimento vocale, generazione testuale, sintesi vocale e animazione, implementati come script distinti e disaccoppiati tramite un sistema event-driven. Questo approccio ha permesso una buona separazione delle responsabilità e ha facilitato sia il debugging che l'iterazione rapida delle funzionalità.

Tuttavia, durante lo sviluppo sono emerse alcune aree in cui il disaccoppiamento può essere ulteriormente rafforzato. In particolare, la gestione della sincronizzazione temporale tra moduli, la propagazione dello stato linguistico e il flusso di dati durante le transizioni (come il cambio di zona o il passaggio da una lingua all'altra) mostrano ancora alcune dipendenze dirette o logiche centralizzate. Questi elementi potrebbero diventare colli di bottiglia man mano che il sistema cresce in complessità o viene utilizzato in scenari più estesi.

Una possibile evoluzione architetturale consiste nel rafforzare le interfacce tra i moduli, estendendo l'utilizzo di pattern già compatibili con Unity. In particolare, sebbene il sistema faccia già uso di meccanismi di tipo *observer* per la gestione degli eventi, un'applicazione più sistematica di questo approccio, affiancata eventualmente

da pattern come il *mediator*[2], può migliorare ulteriormente la leggibilità e la modularità dell'architettura. Tali accorgimenti favoriscono la chiarezza della logica applicativa, semplificano l'integrazione di nuovi componenti (come motori vocali alternativi o sistemi di animazione personalizzati) e migliorano la manutenibilità complessiva.

I benefici concreti di questo tipo di rafforzamento architetturale possono essere sintetizzati nei seguenti punti:

- **Maggiore chiarezza e isolamento delle responsabilità:** utile in progetti collaborativi o a lunga durata;
- **Facilità di estensione o sostituzione dei moduli:** ad esempio per introdurre nuovi servizi STT/TTS o plugin per l'animazione labiale;
- **Manutenibilità migliorata:** grazie alla riduzione di chiamate dirette e a una gestione esplicita dello stato tramite eventi o interfacce.

Va tuttavia sottolineato che una maggiore modularizzazione non comporta necessariamente un miglioramento diretto delle prestazioni percepite. Per ottimizzare la reattività complessiva del sistema e ridurre la latenza dell'interazione vocale, è necessario intervenire su altri aspetti più direttamente legati alla pipeline asincrona. In particolare:

- **Ottimizzazione della pipeline asincrona:** attualmente le operazioni principali, ovvero la trascrizione dell'input vocale, la generazione testuale e la sintesi vocale, sono eseguite in modo strettamente sequenziale, sebbene ciascuna fase sia gestita in maniera asincrona. Questo approccio garantisce la coerenza del flusso ed evita condizioni di errore dovute a dipendenze logiche tra le fasi. In prospettiva, è possibile valutare una ristrutturazione della pipeline che consenta una parziale sovrapposizione tra fasi consecutive, ad esempio avviando la generazione testuale mentre la trascrizione vocale è ancora in corso. L'utilizzo di tecniche di trascrizione incrementale o in streaming, già supportate da alcuni modelli ASR, potrebbe contribuire a ridurre sensibilmente la latenza complessiva senza compromettere l'integrità dell'interazione.

[2]Il pattern *mediator* prevede l'introduzione di un modulo centrale che gestisce la comunicazione tra componenti indipendenti. In Unity, può essere realizzato tramite uno script "manager" che riceve e distribuisce eventi tra i moduli, evitando che questi comunichino direttamente tra loro.

- **Caching intelligente:** l'introduzione di meccanismi di cache per risposte frequenti, segnali ricorrenti o clip audio già sintetizzate può ridurre significativamente il carico computazionale e il numero di richieste verso servizi esterni, ottimizzando sia la reattività sia il consumo di risorse.

- **Gestione efficiente delle risorse:** oltre all'uso di tecniche standard come il pooling degli oggetti o la limitazione delle istanze concorrenti, è opportuno monitorare con attenzione l'allocazione dinamica di memoria e l'utilizzo della CPU, soprattutto in contesti XR standalone, dove le risorse hardware sono limitate e condivise con il rendering in tempo reale.

- **Ottimizzazione del carico grafico e del codice:** alleggerire i prefab rimuovendo componenti superflui, semplificare gli shader, ridurre la profondità delle texture e ottimizzare gli script (es. evitando chiamate ridondanti in `Update()`) può contribuire a mantenere alte le prestazioni, soprattutto su dispositivi a capacità limitata. Anche una gestione oculata dell'animazione e del LOD (Level of Detail) può avere impatti significativi.

- **Monitoraggio e logging approfondito:** raccogliere dati operativi (come tempi di risposta, errori, uso di memoria) consente di individuare colli di bottiglia e comportamenti anomali. Strumenti di logging ben progettati facilitano la diagnosi e la manutenzione del sistema, anche in ambienti di produzione.

In sintesi, l'evoluzione architetturale del sistema non può basarsi esclusivamente su una maggiore modularizzazione, ma richiede un approccio integrato che combini buone pratiche di progettazione software, ottimizzazione della pipeline asincrona e strumenti di osservabilità. Solo in questo modo sarà possibile garantire scalabilità, affidabilità e una migliore esperienza utente, anche in scenari XR dinamici e poco prevedibili.

4.5.2 Tecnologie alternative e scenari di scalabilità

Un secondo asse strategico di sviluppo riguarda la sostituzione, l'integrazione o la riconfigurazione di alcune tecnologie chiave, in funzione dei requisiti di scalabilità, privacy, connettività e costi operativi. In contesti produttivi o aziendali reali, infatti, le scelte tecnologiche non devono essere guidate esclusivamente dalla qualità tecnica, ma anche dalla sostenibilità a lungo termine e dall'adattabilità a infrastrutture preesistenti.

Speech-to-Text locale o ibrido

Attualmente il riconoscimento vocale viene eseguito tramite le API cloud di *Whisper*, che garantiscono un'ottima accuratezza, tempi di risposta accettabili e costi contenuti. Questa soluzione si è rivelata particolarmente adatta all'integrazione in Unity, grazie alla semplicità d'uso e all'assenza di requisiti hardware specifici sul lato utente.

Tuttavia, in contesti dove la connessione a Internet è instabile o assente, oppure in applicazioni dove la privacy dei dati audio è un requisito stringente, può essere utile considerare alternative locali. Tra queste, una delle opzioni più promettenti è *Whisper.cpp*, una reimplementazione ottimizzata del modello Whisper che consente di eseguire la trascrizione direttamente sul dispositivo, senza inviare dati a server esterni. A differenza della versione cloud, questa soluzione funziona interamente in locale (*on-device*) e non richiede connessione di rete, ma implica un carico computazionale significativo, specialmente su dispositivi XR standalone con risorse limitate.

Una possibile configurazione ibrida prevede l'utilizzo del modello cloud in condizioni standard, con un sistema di fallback automatico su una versione locale in caso di perdita di connettività. Questo approccio aumenta la robustezza del sistema, mantenendo alte prestazioni nei casi ordinari.

In alternativa, è consigliabile monitorare l'evoluzione di SDK vocali integrabili nativamente in Unity, come il recente *Meta Voice SDK*[3], che in futuro potrebbero offrire riconoscimento vocale embedded, leggero ed efficiente.

Ottimizzazione e affiancamento di Amazon Polly

La sintesi vocale attualmente utilizza Amazon Polly, una soluzione solida e ben integrata in Unity grazie alla disponibilità di plugin e API semplici. Le sue voci *neural*, dove disponibili, offrono una resa naturale e credibile, ma per alcune lingue (tra cui l'italiano) questa opzione non è ancora disponibile o risulta limitata.

L'esperienza vocale può comunque essere significativamente migliorata attraverso l'uso avanzato del linguaggio SSML[4], che consente una personalizzazione fine del tono, del ritmo e dell'enfasi, anche con le voci standard. Inoltre, l'introduzione di un meccanismo di caching per le frasi ricorrenti contribuirebbe a ridurre la latenza nelle interazioni e i costi legati alle chiamate API.

In una prospettiva evolutiva, l'architettura potrebbe essere mantenuta flessibile per l'*affiancamento opzionale* di altri motori TTS, come Google Cloud TTS o

[3]Meta Voice SDK è una libreria introdotta da Meta nel 2025 per offrire funzionalità STT e TTS offline su dispositivi Quest. Attualmente disponibile in versione preview.

[4]Speech Synthesis Markup Language: permette di controllare con precisione elementi come intonazione, pause, volume, velocità e pronuncia.

ElevenLabs, selezionabili in base alla lingua, alla qualità vocale desiderata o al contesto emozionale. Tale scelta dovrebbe tuttavia tenere conto della semplicità di integrazione e dei costi, adottando un approccio simile a quello considerato per i modelli linguistici: affiancare strumenti alternativi solo se garantiscono un miglioramento sensibile senza compromettere la leggerezza e la compatibilità del sistema.

Modelli linguistici leggeri o architetture ibride

La scelta di utilizzare le API di *ChatGPT* si è rivelata particolarmente vantaggiosa in fase di sviluppo, grazie alla facilità d'integrazione con Unity, alla bassa latenza nella generazione delle risposte e ai costi contenuti rispetto al budget tipico di un progetto XR. Tuttavia, in ottica futura, si potrebbero valutare alternative basate su LLM differenti, purché mantengano caratteristiche simili in termini di rapidità, affidabilità e semplicità di integrazione.

Una possibile direzione evolutiva è rappresentata da modelli linguistici *leggeri* eseguibili in locale, come *Mistral*, *Phi* o versioni quantizzate tramite tecniche LoRA[5]. Questi modelli, pur offrendo capacità limitate rispetto a GPT-4, possono essere adatti per task conversazionali semplici, mantenendo l'interazione operativa anche in assenza di connessione.

Un approccio particolarmente interessante è quello di un'architettura ibrida, nella quale il sistema impiega modelli cloud (come ChatGPT) per la gestione ordinaria e passa in automatico a un LLM locale in caso di perdita di connettività o superamento di soglie di costo. Questo tipo di fallback garantirebbe maggiore resilienza e continuità del servizio, soprattutto in contesti aziendali o fieristici dove la rete non è sempre affidabile.

[5]LoRA (Low-Rank Adaptation) è una tecnica che consente di adattare modelli preaddestrati aggiungendo un numero ridotto di parametri addestrabili, mantenendo inalterato il modello originale. Le versioni quantizzate, invece, riducono la precisione numerica dei pesi (es. da 32 bit a 4 o 8 bit) per diminuire l'occupazione in memoria e i costi computazionali, rendendo il modello più adatto all'esecuzione su dispositivi locali.

Sincronizzazione labiale: alternative a Oculus Lip Sync

Considerando che *Oculus Lip Sync* è un pacchetto deprecato e non più aggiornato, è opportuno valutare l'adozione di alternative più moderne, mantenute e compatibili con i principali avatar 3D.

Una prima possibilità è offerta da *Ready Player Me*, che ha recentemente introdotto un sistema integrato di visemi e blendshape basati su audio[6].

In alternativa, esistono asset commerciali e open-source per Unity, come *SALSA LipSync Suite*, che offrono sincronizzazione audio-to-lips tramite analisi dell'audio in tempo reale, con configurazioni personalizzabili e supporto per blendshape o osso. Tali soluzioni risultano più stabili, supportano l'uso offline e sono facilmente adattabili a diversi stili di avatar, migliorando l'affidabilità e la qualità visiva del sistema.

Direzioni di sviluppo tecnico

Le potenziali evoluzioni del sistema sono molteplici e adattabili a diversi scenari d'uso. Miglioramenti di carattere incrementale possono portare benefici immediati in termini di prestazioni e affidabilità, mentre sviluppi più strutturali richiedono un'analisi approfondita per garantire scalabilità e flessibilità a lungo termine. Nel complesso, tali direzioni consentono di ampliare le capacità del sistema mantenendo un equilibrio tra efficienza operativa e qualità dell'esperienza utente.

[6]Ready Player Me ha aggiunto il supporto ai visemi nella pipeline dei propri avatar, abilitando l'uso di sincronizzazione labiale automatica tramite audio clip e compatibilità con pacchetti TTS. Tuttavia, l'integrazione diretta richiede un aggiornamento del modello e l'uso del loro SDK più recente.

5. Conclusioni

Il progetto descritto in questa tesi ha dimostrato la fattibilità dell'integrazione di un assistente vocale intelligente in uno scenario immersivo, combinando riconoscimento vocale, generazione del linguaggio naturale, sintesi vocale, animazioni sincronizzate degli avatar e interazione spaziale in ambienti XR. Il sistema ha mostrato un buon livello di coerenza e flessibilità, risultando in grado di gestire conversazioni multilingue in modo credibile e reattivo, anche in condizioni operative non ideali.

Lo sviluppo ha richiesto l'integrazione di tecnologie eterogenee, affrontando sfide legate alla sincronizzazione tra moduli, alla latenza delle chiamate API e alla robustezza del flusso conversazionale. L'approccio iterativo seguito ha consentito di raffinare progressivamente l'architettura, con particolare attenzione alla modularità, alla reattività del sistema, alla chiarezza della trascrizione e alla facilità di interazione da parte dell'utente, sia sul piano vocale che grafico.

Coerentemente con l'obiettivo di creare una soluzione adatta a contesti aziendali, molte delle scelte progettuali sono state guidate da criteri di sostenibilità economica, professionalità dell'esperienza e rispetto dei vincoli aziendali e di progetto. Sono stati adottati strumenti a basso costo come OpenAI Whisper e ChatGPT, affiancati ad Amazon Polly per la sintesi vocale gratuita, garantendo un bilanciamento efficace tra costi e prestazioni. Il *prompt engineering* è stato strutturato per rispettare vincoli comportamentali e di tono adeguati a un contesto professionale, mentre l'estetica e la presentazione dei chatbot nelle diverse lingue sono state curate per risultare coerenti e formali.

Sebbene inizialmente concepito per un museo virtuale aziendale, il sistema è stato sviluppato con un'architettura sufficientemente flessibile da poter essere adattato, con modifiche contenute, anche ad altri scenari XR professionali. Le prestazioni riscontrate nei test condotti in condizioni realistiche confermano la solidità dell'approccio adottato.

Le fasi di test, pur evidenziando margini di miglioramento in termini di ottimizzazione e rifinitura dell'esperienza utente, hanno confermato il corretto funzionamento del sistema anche in condizioni operative non ideali, come ambienti rumorosi, connessioni di rete instabili o varietà di accenti e lingue. La pipeline, pur non essendo ancora spinta ai massimi livelli di efficienza, ha dimostrato una buona affidabilità, garantendo tempi di risposta adeguati e un'interazione fluida e comprensibile.

In conclusione, il lavoro ha raggiunto l'obiettivo di costruire un'interazione vocale credibile e immersiva all'interno di uno spazio tridimensionale, mantenendo un equilibrio tra qualità dell'esperienza e sostenibilità tecnica. Senza voler rappresentare una soluzione definitiva, il sistema realizzato costituisce una base solida, riutilizzabile e versatile, da cui potranno partire sviluppi futuri orientati al miglioramento delle prestazioni e della resa espressiva, favorendo un'integrazione nei contesti digitali aziendali che sia al tempo stesso efficace, professionale e sostenibile. Il progetto si pone dunque come un punto di partenza solido per lo sviluppo di esperienze vocali intelligenti, sempre più naturali, accessibili e integrate negli ambienti digitali professionali.

Bibliografia

Adamopoulou, E., & Moussiades, L. (2020). Chatbot in customer services: Their relevance and impact on service quality. *Journal of Service Science and Management*, *13*(4), 593–601.

AI Multiple, C. D. (2025). *XR/AR in Manufacturing: 7 Use Cases with Examples [2025]* [Accessed: 2025-05-11]. https://research.aimultiple.com/ar-in-manufacturing/

Alkaeed, M., Qayyum, A., & Qadir, J. (2023). Privacy Preservation in Artificial Intelligence and Extended Reality (AI-XR) Metaverses: A Survey. *arXiv preprint arXiv:2310.10665*. https://arxiv.org/abs/2310.10665

Aphex34. (2017). Convolutional neural network diagram [Immagine di pubblico dominio]. https://commons.wikimedia.org/wiki/File:Convolutional_neural_network_diagram.png

Frey, C. B., & Osborne, M. (2020). A history of virtual reality. *Oxford Martin School Working Paper*. https://www.oxfordmartin.ox.ac.uk/downloads/academic/A-History-of-Virtual-Reality.pdf

GeeksforGeeks. (2023). Backpropagation in Neural Network [Accessed: 2025-05-23]. https://www.geeksforgeeks.org/backpropagation-in-neural-network/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org

Hill, J., Ford, W. R., & Farreras, I. G. (2020). Conversational AI Chatbots: The next frontier in customer experience. *Computers in Human Behavior*, *106*, 103–123.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

IBM. (2021). Cos'è una rete neurale? [Accessed: 2025-05-23]. https://www.ibm.com/it-it/think/topics/neural-networks

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems (NIPS)*, *25*, 1097–1105. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Kurniawan, T., Wulandari, A., & Permanasari, A. (2015). Design of Chatbot with 3D Avatar, Voice Interface and Facial Expression. *2015 International Conference on Information Technology Systems and Innovation (ICITSI)*, 1–5. https://doi.org/10.1109/ICITSI.2015.7437705

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, K. P., Wong, S.-H., Chan, L.-K., Lam, K.-H., & Chow, K.-H. (2024). Utilizing Virtual Reality and Generative AI Chatbot for Job Interview Simulations [Preprint disponibile su ResearchGate]. *Research and Practice in Technology Enhanced Learning*. https://www.researchgate.net/publication/376439375_Utilizing_Virtual_Reality_and_Generative_AI_Chatbot_for_Job_Interview_Simulations

Li, Y., Zhang, X., & Wang, H. (2025). Metaverse and XR for cultural heritage education: applications and framework. *Virtual Reality*, *29*(2), 123–140. https://doi.org/10.1007/s10055-025-01126-z

Liu, X., Zhang, J., Hou, G., & Wang, Z. (2018). Virtual Reality and Its Application in Military [Licenza CC BY 3.0]. *Defence Technology*. https://www.researchgate.net/figure/The-Sword-of-Damocles_fig1_326444949

Mehmood, A., et al. (2021). A Review on Generative Adversarial Networks: Algorithms, Applications, and Challenges. *Journal of Artificial Intelligence and Soft Computing Research*, *11*(2), 87–111.

Mestanza, L. R., Paucar, E., Hilario, C., Chavez, A., Hilario, L., & Machaca, J. (2021). State of the art of virtual reality technology. *International Journal of Advanced Computer Science and Applications*, *12*(5), 681–688. https://doi.org/10.14569/IJACSA.2021.0120579

Meta Platforms, Inc. (2024). *Meta Quest 3* [Accesso maggio 2025]. Recuperato maggio 21, 2025, da https://www.meta.com/quest/quest-3/

Onlim. (2025). *The History of Chatbots* [Accesso: 23 maggio 2025]. https://onlim.com/en/the-history-of-chatbots/

Palmer, S. B., Hartnett, M., & Nguyen, H. (2020). State of the art of virtual reality technology. *International Journal of Virtual Reality*, *20*(1), 1–20.

Pape, D. (1999). VPL EyePhone and DataGlove [Fotografia in pubblico dominio]. https://commons.wikimedia.org/wiki/File:VPL_Eyephone_and_Dataglove.jpg

ResearchGate. (2019). *Schematic Diagram of a Basic Convolutional Neural Network (CNN) Architecture* [Accessed: 2025-05-23]. https://www.researchgate.net/

figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909

Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th). Pearson.

Shawar, B. A., & Atwell, E. (2007). Chatbots: History, technology, and applications. *International Journal of Computer Science and Applications*, *4*(3), 25–31.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Weizenbaum, J. (1966). ELIZA [Accesso: 23 maggio 2025].

Wikipedia contributors. (2024). Recurrent neural network — Wikipedia, The Free Encyclopedia [[Online; accessed 23-May-2025]]. https://en.wikipedia.org/wiki/Recurrent_neural_network

Wired, N. K. (2025). *The Dream of the Metaverse Is Dying. Manufacturing Is Keeping It Alive* [Accessed: 2025-05-11]. https://www.wired.com/story/the-metaverse-is-here-and-its-industrial

XR Today, R. C. (2022). *The State of Extended Reality in Gaming* [Accessed: 2025-05-11]. https://www.xrtoday.com/virtual-reality/the-state-of-extended-reality-in-gaming/

Xu, Z., Liu, H., Wang, J., & Zhang, K. (2023). Exploring the Evolving Landscape of Extended Reality (XR) Technology. *Journal of Computer Science and Technology*, *38*(2), 215–233.