



ISTITUTO ISTRUZIONE SUPERIORE  
"E. FERMI - R. GUTTUSO"  
GIARRE - CT

 Microsoft Azure



 Meta



# AI Teamwork-oriented

Collaborazione passiva di 3 modelli per l'interazione vocale

“L’intelligenza artificiale  
è la nuova elettricità.  
Trasformerà ogni  
settore.”

—Andrew Yan-Tak Ng



# In cosa consiste questo progetto?



## Creatività

Prima degli effettivi modelli di Intelligenza Artificiale, la creatività e la logica connettiva tra loro è stata sviluppata da un essere umano.



## Collaborazione

L'elaborazione della tua tenera voce è frutto della collaborazione di 3 modelli distinti: Speech Recognition, Language Model e Speech Synthesizer.

# Principi di base del progetto

Il progetto è dedito a dimostrare come la collaborazione di più modelli possa portare ad un prodotto innovativo, con estrema versatilità.

<u>Infinite soluzioni</u>	Il principio di base del progetto può essere applicato per molteplici prodotti.
<u>Cloud-based</u>	Basarsi sul cloud per la realizzazione di questo progetto riduce i costi.
<u>Sempre up-to-date</u>	Ogni volta che un modello si aggiorna, il progetto migliora parallelamente ad esso.
<u>Codice open-source</u>	Il codice semplice e versatile permette di essere compreso e re-impiegato dagli sviluppatori.
<u>Implementazione rapida</u>	L'idea di base può essere ulteriormente sviluppata per permettere molteplici applicazioni.

For more info:  
[github.com/TheLastOxygen](https://github.com/TheLastOxygen)

Visita anche gli altri progetti:  
[TeleCast](#) | [PyChat](#)

# Indice della presentazione

01

## Modelli impiegati

Speech Recognition, Language Model, Speech Synthesizer

02

## Risorse hardware utilizzate

Osservazione di soluzioni «cloud-based» oppure «home-hosted»

03

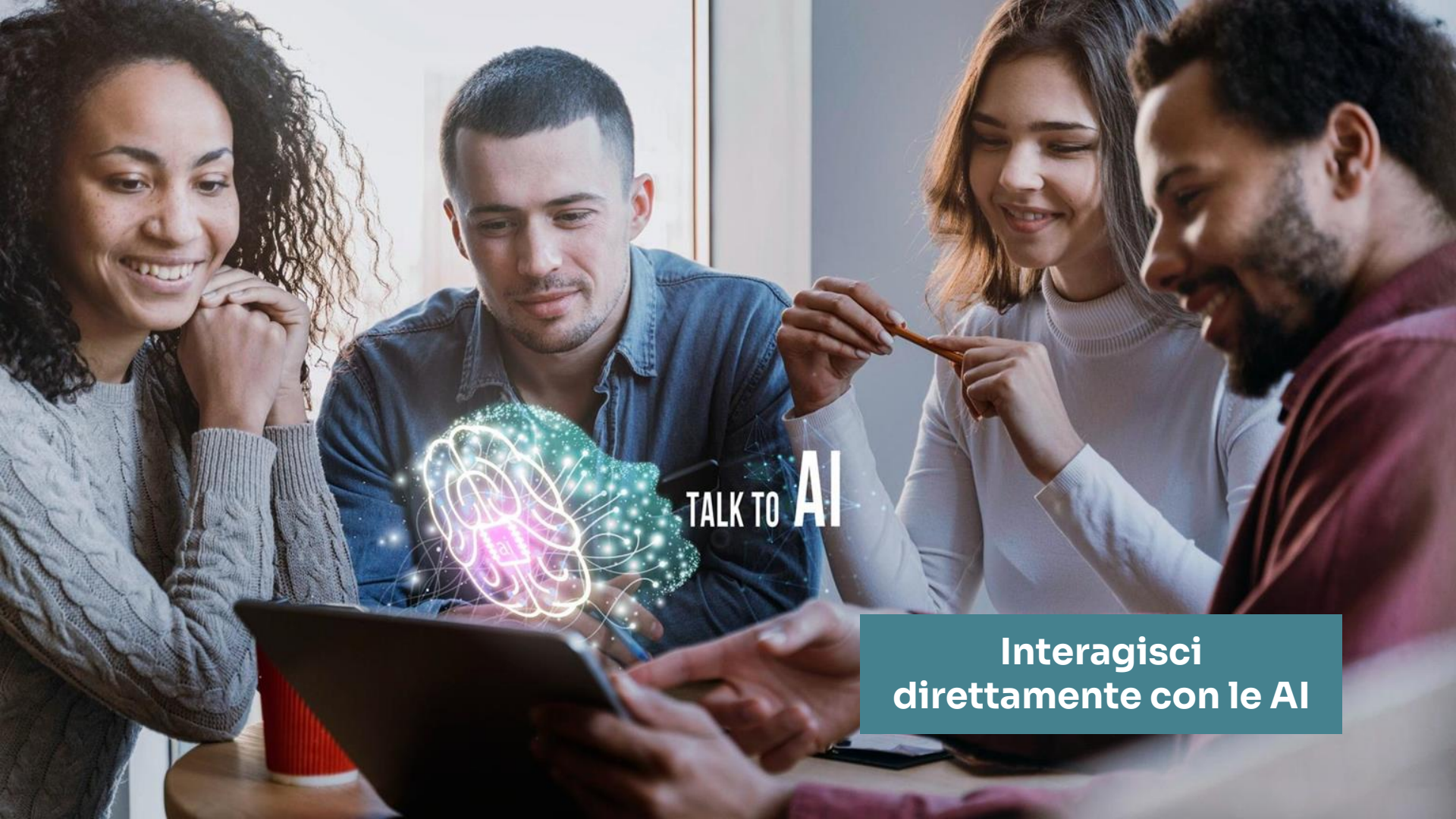
## Codice e API

Python, la logica applicativa e le librerie

04

## Integrazione con Hardware

Un potente assistente in un dispositivo IoT



TALK TO AI

**Interagisci  
direttamente con le AI**

01

# Modelli AI

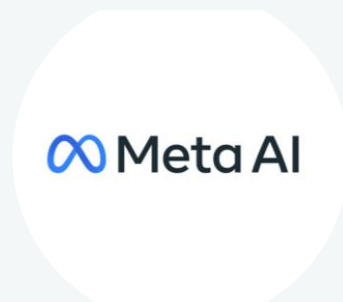
Scelta dei modelli per l'esempio di realizzazione





## Speech Recognition

Servizio di Azure per trascrivere l'audio in testo. Configurato per l'italiano (it-IT). Utilizza l'API SpeechRecognizer con recognize\_once per il riconoscimento di singole frasi.



## Language Model

Modello di linguaggio avanzato per creare risposte testuali. Configurato per aggiungere il carattere ~ alla fine di ogni frase per facilitare la sintesi vocale. Utilizza l'API ollama.chat con il modello LLaMa3.



## Speech Synthesizer

Servizio di Azure per convertire il testo in voce. Utilizza la voce neurale it-IT-ElsaNeural. Utilizza l'API SpeechSynthesizer con speak\_text\_async.





The infographic consists of three horizontal bars stacked vertically. Each bar has a dark teal top half with a large white number and a lighter teal bottom half with a white description. Lines connect the bars to a vertical line on the left and a vertical line on the right, with small teal circles at the ends of these lines. The top bar is connected to a line that goes up and left to a teal circle. The middle bar is connected to a line that goes up and left to a teal circle. The bottom bar is connected to a line that goes up and left to a teal circle. The right side of the infographic has a vertical line with three teal circles at the bottom.

**~9s**

Tempo medio da fine domanda a inizio risposta

**8,000**

Token disponibili per il modello LLI utilizzato

**4.7 GB**

Peso del modello LLI lightweight utilizzato come esempio

# Sperimentazione in locale

Download del modello LLaMa3	Configurazione dell'ambiente di sviluppo	Esecuzione del codice	Test e valutazione
Ho utilizzato Ollama, un programma che semplifica il download e l'utilizzo dei modelli LLaMA, per scaricare il modello LLaMA 3 direttamente sul mio PC. Questo mi ha consentito di ottenere facilmente una copia locale del modello addestrato.	Dopo aver scaricato il modello LLaMA 3, ho configurato l'ambiente di sviluppo sul mio PC, installando le librerie Python necessarie. Questo mi ha permesso di eseguire il codice e di interagire con il modello LLaMA utilizzando il linguaggio di programmazione Python.	Utilizzando il modello LLaMA 3 scaricato tramite Ollama, ho lanciato il codice sul mio PC. Il codice si connette al modello LLaMA 3 locale e utilizza il servizio di riconoscimento vocale e di sintesi vocale di Azure per l'interazione vocale con l'utente.	Dopo aver eseguito il codice, ho condotto vari test per valutare le prestazioni del sistema in locale. Questo includeva la valutazione del tempo di risposta, la qualità delle risposte generate e la gestione delle richieste vocali.



# Microsoft Cognitive Services

Per utilizzare le API di Microsoft Cognitive Services nel mio progetto, ho creato un account Azure e attivato i Cognitive Services. Ho configurato una risorsa "Speech" ottenendo la chiave di sottoscrizione e l'endpoint di servizio. Ho installato il pacchetto `azure-cognitiveservices-speech` tramite pip e configurato il codice con queste credenziali, impostando la lingua e la voce per la sintesi vocale. Ho implementato il riconoscimento vocale utilizzando la classe `SpeechRecognizer` per convertire il parlato in testo, e la sintesi vocale usando la classe `SpeechSynthesizer` per convertire il testo in parlato.

02

# Risorse Hardware

Analisi delle possibili soluzioni di hosting per un progetto simile



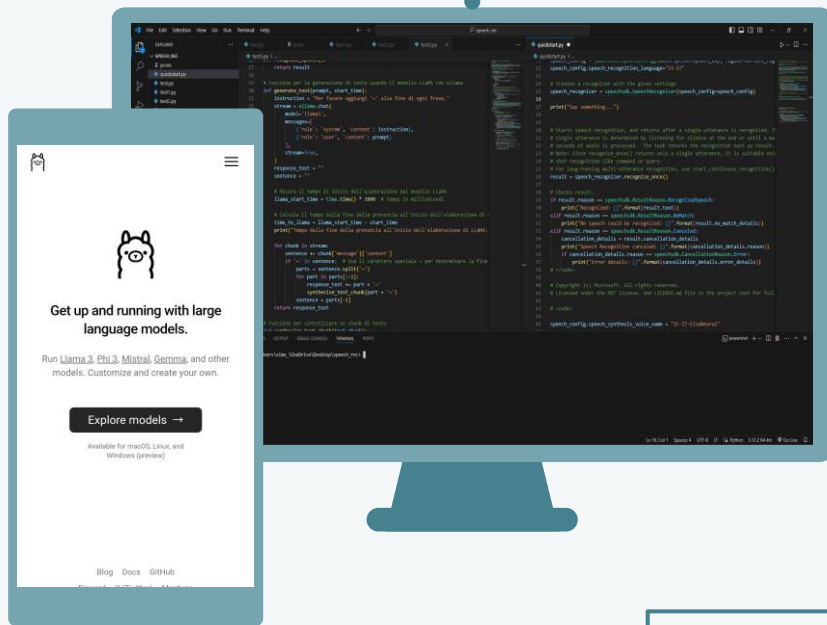
# Soluzioni Cloud-Based

Le soluzioni cloud, come Microsoft Azure, offrono numerosi vantaggi. In primo luogo, forniscono accesso a risorse computazionali scalabili, consentendo di gestire carichi di lavoro variabili senza dover investire in hardware costoso. Inoltre, i servizi cloud sono altamente affidabili, con tempi di inattività minimi e backup automatici, assicurando la continuità del servizio. L'integrazione con le API di Microsoft Cognitive Services è facilitata dalle piattaforme cloud, che offrono strumenti e SDK ben documentati. Tuttavia, le soluzioni cloud presentano anche alcuni svantaggi. I costi possono aumentare rapidamente con l'uso intensivo delle risorse, e c'è una dipendenza dalla connessione internet per accedere ai servizi. Inoltre, ci possono essere preoccupazioni relative alla privacy e alla sicurezza dei dati, poiché le informazioni vengono trasmesse e memorizzate su server remoti.



# Soluzioni Home-Hosted

Le soluzioni home-hosted, d'altra parte, offrono un maggiore controllo sull'hardware e sui dati. Utilizzando risorse locali, si può ridurre la dipendenza da fornitori di servizi esterni e potenzialmente risparmiare sui costi a lungo termine, specialmente per progetti di dimensioni contenute o con esigenze specifiche. Tuttavia, ospitare l'infrastruttura localmente presenta anche delle sfide. È necessario investire in hardware adeguato e gestire la manutenzione, il che può risultare costoso e richiedere competenze tecniche specifiche. Inoltre, l'affidabilità può essere un problema, poiché le risorse locali possono essere più suscettibili a guasti hardware e interruzioni di servizio.



# Soluzioni per il progetto di esempio



## Home

Nel mio progetto di esempio, ho scelto di utilizzare il mio PC per eseguire il modello LLaMa 3 localmente utilizzando Ollama, approfittando del controllo diretto sui dati e la flessibilità nella gestione del modello. Ciò richiede però dell'hardware performante.



## Cloud

Contemporaneamente, ho utilizzato il cloud di Microsoft Azure per il riconoscimento vocale e la sintesi vocale, beneficiando della scalabilità e dell'affidabilità del cloud per questi servizi intensivi dal punto di vista computazionale.

03

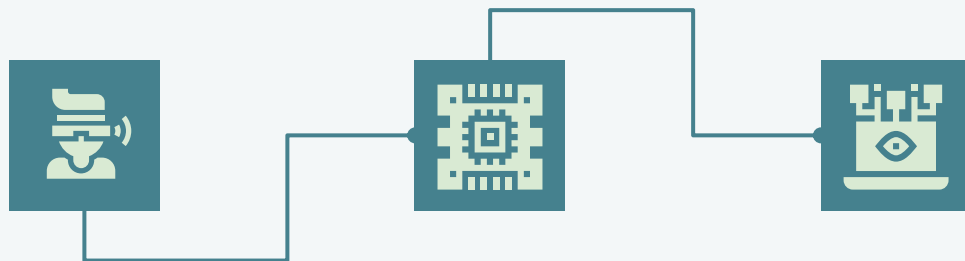
# Codice ed API

Illustrazione del codice per la realizzazione del progetto





# Schema di Funzionamento



## Riconoscimento Vocale

L'utente parla e il sistema cattura l'input vocale. Il servizio di riconoscimento di MCS converte l'input in testo.

## Generazione Testo

Il testo riconosciuto viene inviato a LLaMa 3 tramite Ollama, che genera una risposta testuale.

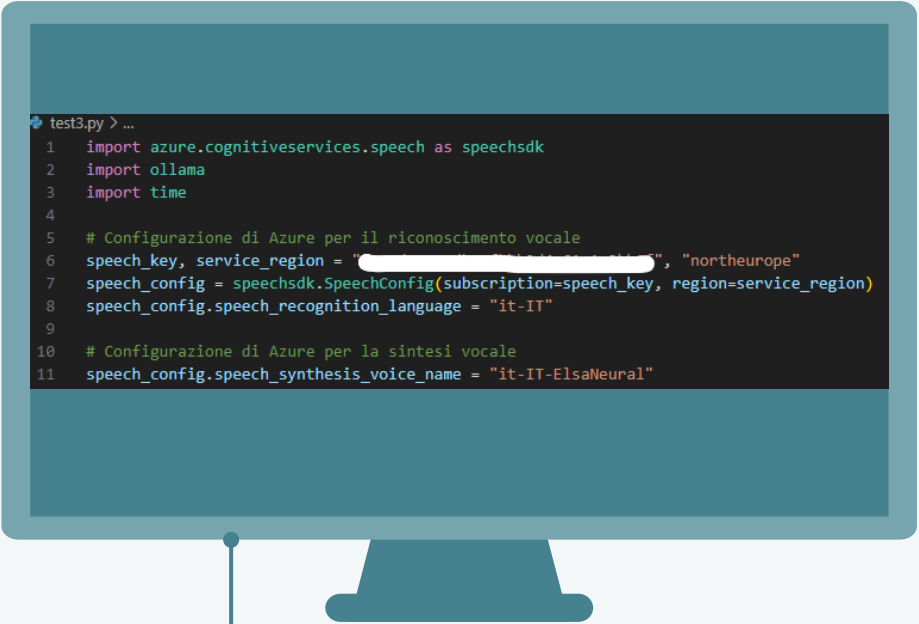
## Sintesi Vocale

La risposta testuale viene trasformata in parlato utilizzando la sintesi vocale di MCS. Il sistema riproduce la risposta vocale all'utente.

# Configurazione delle API di Microsoft Cognitive Services

Passaggi per Ottenere la Chiave di Sottoscrizione e l'Endpoint:

- Creazione di un Account Azure: Per utilizzare le API di Microsoft Cognitive Services, è necessario un account Azure. Visita il sito di Azure e crea un account se non ne possiedi già uno.
- Creazione di una Risorsa Cognitive Services: Dopo aver effettuato l'accesso al portale di Azure, crea una nuova risorsa Cognitive Services. Vai su "Crea una risorsa" e cerca "Cognitive Services". Segui le istruzioni per configurare la tua nuova risorsa.
- Ottenere la Chiave di Sottoscrizione e l'Endpoint: Una volta creata la risorsa, vai alla sezione "Chiavi e endpoint" nella pagina della risorsa. Qui troverai due chiavi di sottoscrizione e l'endpoint del servizio. Copia la chiave e l'endpoint per utilizzarli nel tuo codice.

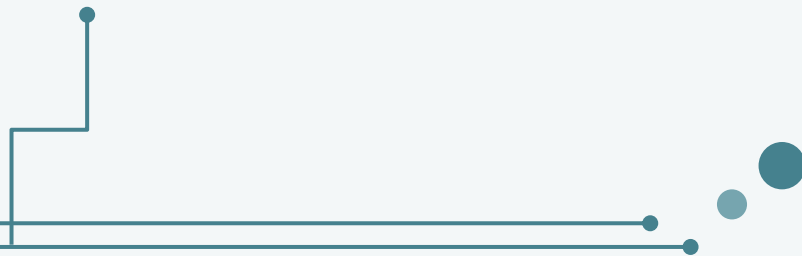


```
test3.py > ...
1  import azure.cognitiveservices.speech as speechsdk
2  import ollama
3  import time
4
5  # Configurazione di Azure per il riconoscimento vocale
6  speech_key, service_region = "XXXXXXXXXXXXXXXXXXXX", "northeurope"
7  speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)
8  speech_config.speech_recognition_language = "it-IT"
9
10 # Configurazione di Azure per la sintesi vocale
11 speech_config.speech_synthesis_voice_name = "it-IT-ElsaNeural"
```

# Funzione di riconoscimento vocale

```
# Funzione per il riconoscimento vocale
def recognize_speech():
    speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config)
    result = speech_recognizer.recognize_once()
    return result
```

Il codice presentato definisce una funzione denominata `recognize_speech`, che utilizza le API di Microsoft Cognitive Services per eseguire il riconoscimento vocale. All'interno della funzione, viene creata un'istanza di `SpeechRecognizer` utilizzando la configurazione di riconoscimento vocale precedentemente impostata in `speech_config`. Questa configurazione include dettagli come la chiave di sottoscrizione e la regione del servizio, nonché le impostazioni di lingua e voce. La funzione chiama quindi il metodo `recognize_once()` sull'istanza di `SpeechRecognizer`, il quale avvia il processo di riconoscimento vocale e attende fino a quando una frase viene riconosciuta o il timeout è raggiunto. Il risultato del riconoscimento, contenente il testo trascritto, viene poi restituito dalla funzione. Questo approccio permette di catturare e convertire input vocali in testo scritto, facilitando l'interazione vocale con l'applicazione.



# Funzione per la generazione del testo

```
# Funzione per la generazione di testo usando il modello LLaMA con ollama
def generate_text(prompt, start_time):
    instruction = "Per favore aggiungi '~' alla fine di ogni frase."
    stream = ollama.chat(
        model='llama3',
        messages=[
            {'role': 'system', 'content': instruction},
            {'role': 'user', 'content': prompt}
        ],
        stream=True,
    )
    response_text = ""
    sentence = ""

    # Misura il tempo di inizio dell'elaborazione del modello LLaMA
    llama_start_time = time.time() * 1000 # Tempo in millisecondi

    # Calcola il tempo dalla fine della pronuncia all'inizio dell'elaborazione di LLaMA
    time_to_llama = llama_start_time - start_time
    print("Tempo dalla fine della pronuncia all'inizio dell'elaborazione di LLaMA: {:.2f} millisecondi".format(time_to_llama))

    for chunk in stream:
        sentence += chunk['message']['content']
        if "~" in sentence: # Usa il carattere speciale ~ per determinare la fine di una frase
            parts = sentence.split('~')
            for part in parts[:-1]:
                response_text += part + '~'
                synthesize_text_chunk(part + '~')
            sentence = parts[-1]
    return response_text
```

La funzione `generate_text` utilizza il modello di linguaggio LLaMA tramite Ollama per generare testo basato su un prompt fornito dall'utente. All'inizio della funzione, viene definita un'istruzione specifica per il modello: aggiungere il carattere speciale '~' alla fine di ogni frase. Viene quindi avviata una conversazione con il modello LLaMA 3, passando l'istruzione e il prompt come messaggi. La funzione misura il tempo di inizio dell'elaborazione del modello LLaMA in millisecondi e calcola il tempo trascorso dalla fine della pronuncia dell'utente all'inizio dell'elaborazione da parte del modello. Questo tempo di latenza viene stampato a schermo per monitorare le prestazioni. La risposta generata dal modello LLaMA viene ricevuta in streaming, e la funzione accumula i chunk di testo in una variabile chiamata `sentence`. Quando viene rilevato il carattere speciale '~', che indica la fine di una frase, la funzione divide la `sentence` in parti, aggiungendole alla variabile `response_text`. Ogni parte completata viene anche inviata alla funzione `synthesize_text_chunk` per la sintesi vocale immediata. Infine, la funzione restituisce il testo generato completo. Questo processo permette di generare e sintetizzare testo in modo efficiente, gestendo la latenza e assicurando che le risposte siano fornite tempestivamente.

# Funzione di vocalizzazione

```
# Funzione per sintetizzare un chunk di testo
def synthesize_text_chunk(text_chunk):
    # Utilizza una nuova istanza del sintetizzatore per evitare di interrompere la sintesi in corso
    local_speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)
    local_speech_config.speech_synthesis_voice_name = "it-IT-ElsaNeural"
    local_synthesizer = speechsdk.SpeechSynthesizer(speech_config=local_speech_config)
    local_synthesizer.speak_text_async(text_chunk).get() # Aspetta che la sintesi sia completata
```

La funzione `synthesize_text_chunk` è progettata per sintetizzare un segmento di testo (chunk) in parlato utilizzando il servizio di sintesi vocale di Microsoft Cognitive Services. La funzione inizia creando una nuova istanza della configurazione del servizio di sintesi vocale (`local_speech_config`) utilizzando la chiave di sottoscrizione e la regione del servizio specificate. Successivamente, viene impostata la voce da utilizzare per la sintesi vocale, in questo caso "it-IT-ElsaNeural", che rappresenta una voce in italiano. Una nuova istanza di `SpeechSynthesizer` viene poi creata con questa configurazione locale. Questa nuova istanza è utilizzata per sintetizzare il testo fornito come argomento della funzione. La sintesi del testo è eseguita in modo asincrono tramite il metodo `speak_text_async`, che viene poi seguito da `.get()` per assicurarsi che l'operazione sia completata prima di procedere. Questo approccio permette di evitare interruzioni nella sintesi vocale quando si chiamano più funzioni di sintesi contemporaneamente, creando istanze locali separate per ciascun chunk di testo da sintetizzare.

# Flusso di Lavoro Completo

Il codice presentato rappresenta un flusso di lavoro completo per un sistema interattivo di riconoscimento e sintesi vocale, utilizzando Microsoft Cognitive Services e il modello LLaMA tramite Ollama. Il flusso inizia con un loop infinito che controlla se il sistema è in modalità di ascolto (`listening_mode`). Se non lo è, il sistema istruisce l'utente a dire "ascolta" per iniziare. Una volta rilevato il comando "ascolta", il sistema attiva la modalità di ascolto e riproduce un messaggio vocale per chiedere all'utente di fare una domanda. Quando il sistema è in modalità di ascolto, il riconoscimento vocale viene effettuato nuovamente. Se viene rilevato il comando "adesso puoi andare", il sistema interrompe l'esecuzione. Altrimenti, il testo riconosciuto viene utilizzato come prompt per generare una risposta tramite il modello LLaMA. Il tempo di inizio dell'elaborazione del modello LLaMA viene misurato e viene calcolata la latenza tra la fine della pronuncia e l'inizio dell'elaborazione del modello. La risposta generata viene quindi sintetizzata in parlato e riprodotta all'utente. Infine, il tempo totale di elaborazione dalla domanda alla risposta viene calcolato e stampato. Dopo aver risposto, il sistema torna in modalità di ascolto solo se il comando "ascolta" è presente nel testo riconosciuto, permettendo un'interazione continua e ciclica con l'utente. Questo flusso di lavoro completo mostra come un sistema interattivo può essere implementato utilizzando tecnologie avanzate di riconoscimento e sintesi vocale, combinato con la generazione di testo tramite modelli di linguaggio avanzati.

```
while True:
    if not listening_mode:
        print("Say 'ascolta' to start listening.")
        result = recognize_speech()

        # Controllo del risultato del riconoscimento vocale
        if result.reason == speechsdk.ResultReason.RecognizedSpeech:
            recognized_text = result.text.lower()
            print("Recognized: {}".format(recognized_text))

            if "ascolta" in recognized_text:
                print("Listening mode activated. Please ask your question.")
                listening_mode = True
                # Riproduzione del prompt per fare una domanda
                synthesize_text_chunk("Ciao Simone, fammi una domanda.")
            elif result.reason == speechsdk.ResultReason.NoMatch:
                print("No speech could be recognized: {}".format(result.no_match_details))
            elif result.reason == speechsdk.ResultReason.Canceled:
                cancellation_details = result.cancellation_details
                print("Speech Recognition canceled: {}".format(cancellation_details.reason))
                if cancellation_details.reason == speechsdk.CancellationReason.Error:
                    print("Error details: {}".format(cancellation_details.error_details))
                break
        else:
            result = recognize_speech()

        if result.reason == speechsdk.ResultReason.RecognizedSpeech:
            recognized_text = result.text.lower()
            print("Recognized: {}".format(recognized_text))

            if "adesso puoi andare" in recognized_text:
                print("Stopping execution as per user request.")
                break

            start_time = time.time() * 1000 # Tempo in millisecondi

            response_text = generate_text(recognized_text, start_time)

            # Calcola il tempo trascorso da domanda a risposta
            end_time = time.time() * 1000 # Tempo in millisecondi
            processing_time = end_time - start_time
            print("Tempo di elaborazione da domanda a risposta: {:.2f} millisecondi".format(processing_time))

            print("Risposta generata: {}".format(response_text))

            # Dopo aver risposto, torna alla modalità di ascolto solo se il comando "ascolta" viene ripetuto
            if "ascolta" not in recognized_text:
                listening_mode = False
```

**Video  
dimostrativo  
disponibile in  
allegato.**



# 04

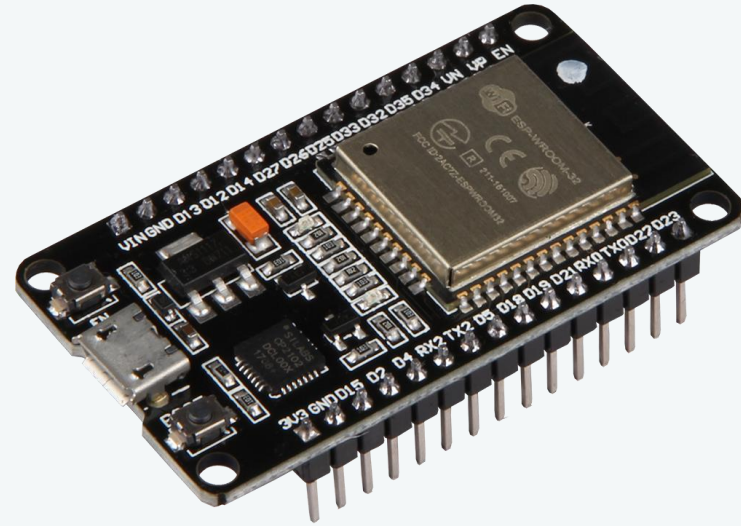
## Integrazione con Hardware

Ipotesi di possibili impieghi in ambito IoT con microcontrollori

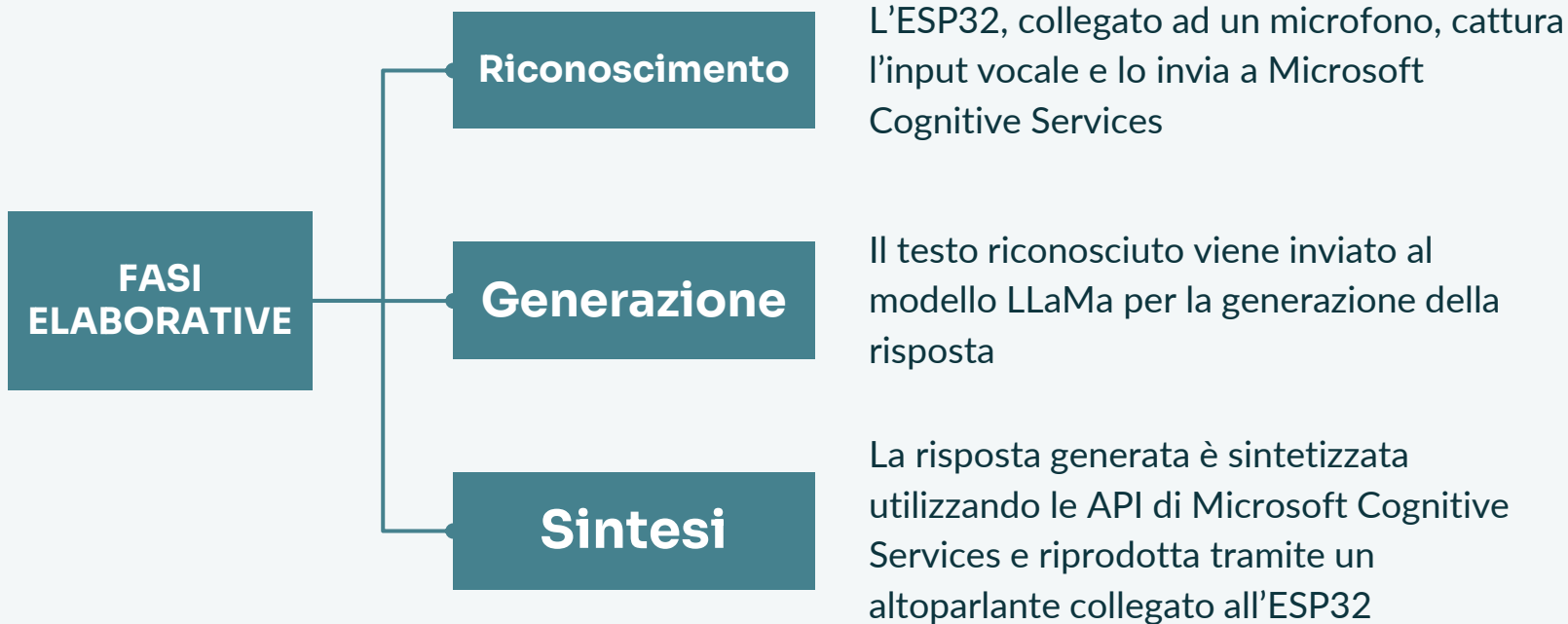




**ESP32: Un  
candidato  
ideale.**



# Architettura del Sistema



# Vantaggi della Soluzione IoT



## Portabilità

Un assistente vocale basato su ESP32 può essere collocato in vari ambienti, come case, uffici o veicoli.



## Scalabilità

Può essere facilmente integrato con altri dispositivi IoT per creare un ecosistema intelligente.



## Efficienza Energetica

L'ESP32 è ottimizzato per operare a basso consumo energetico, rendendolo ideale per applicazioni sempre attive.

# Implementazione



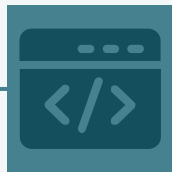
## Integrazione con API

Utilizzare librerie disponibili per ESP32 per interagire con le API di riconoscimento e sintesi vocale.



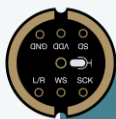
## Configurazione ESP32

Programmare l'ESP32 utilizzando l'IDE Arduino o PlatformIO, configurandolo per inviare dati audio ai servizi cloud.



## IoT e Automazione

Puoi scrivere codice per controllare dispositivi domestici come luci, sensori di temperatura o telecamere di sorveglianza, rendendo l'ESP32 un dispositivo versatile non solo per l'elaborazione del linguaggio naturale, ma anche per l'automazione e il controllo domestico. Integrare queste funzionalità lo trasforma da un semplice modello linguistico in un dispositivo completo e intelligente per la tua casa.



## Microfono

Per questo progetto, si può utilizzare un microfono digitale come il INMP441, che offre un'uscita I2S, facilitando l'integrazione con l'ESP32 per una cattura audio precisa e di alta qualità.



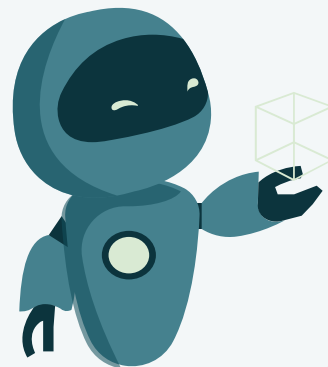
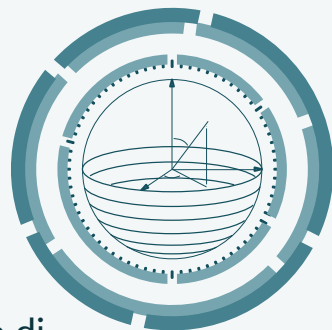
## Altoparlante

Per riprodurre l'audio, si può utilizzare un amplificatore come l'Adafruit I2S 3W Class D Amplifier Breakout, che insieme all'altoparlante Dayton Audio CE38MB-32 da 1.5 fornisce un'uscita audio chiara e potente.



# EXTRAS:

Se sei appassionato alla stampa 3D, potresti realizzare un progetto CAD per racchiudere tutto il tuo hardware in un design alla moda. Grazie alla mia partecipazione al programma Prusa Research, alla formazione fornita dal corso di stampa di Ultimaker e a molteplici rewards da parte del portale Printables di Josef Prusa, posso aiutarti a realizzare qualsiasi idea tu abbia in mente. La parola d'ordine in quello che faccio è **COLLABORAZIONE**.



**PRUSA**  
**RESEARCH**  
by JOSEF PRUSA

 **UltiMaker**

# Grazie per l'attenzione!

Hai altre domande?

[simo\\_d\\_natale45@outlook.it](mailto:simo_d_natale45@outlook.it)

[github.com/TheLastOxygen](https://github.com/TheLastOxygen)

[printables.com/@Oxyg3n\\_234634](https://printables.com/@Oxyg3n_234634)



**Realizza la tua implementazione insieme a me!**

L'idea è pienamente rivisitabile e applicabile in qualunque ambito. Ogni richiesta collaborativa è ben accetta.



Simone Di Natale  
VC Informatica  
ITIS «E.Fermi», Giarre (CT)