

# ***BIT DI PACE (Progetto del gruppo: Cavalli Tommaso, Ouadani Sami, Velluti Mirko, Decolli Andi)***

## **DOCUMENTAZIONE PROGETTO:**

### **Obiettivo del Progetto**

L'obiettivo del progetto è diffondere un linguaggio più "sicuro" in rete. Il programma ascolta l'utente tramite il microfono, trascrive le parole e utilizza un modello di Intelligenza Artificiale in Cloud (tramite l'API di Ollama) per filtrare o riscrivere il contenuto, assicurandosi che il messaggio finale sia educato e pacifico, indipendentemente da come è stato pronunciato originariamente.

#### **1. Configurazione e Inizializzazione**

Il codice prepara gli strumenti necessari:

- Carica un modello linguistico locale (Vosk) per trasformare i suoni in testo senza bisogno di internet.
- Configura i parametri audio (frequenza a 16.000 Hz) per garantire che la qualità della registrazione sia compatibile con il riconoscimento vocale.

#### **2. Acquisizione Audio (Speech-to-Text)**

In questa fase, il programma interagisce con l'hardware:

- Attiva il microfono per un tempo prestabilito (4 secondi).
- Cattura le onde sonore e le trasforma in dati digitali (byte).
- Il motore Vosk analizza questi dati e restituisce una stringa di testo che rappresenta ciò che l'utente ha detto.

#### **3. Elaborazione Intelligente (NLP o Natural Language Processing tramite Ollama)**

Questa è la parte decisionale del codice:

- Il testo ottenuto viene unito a un "Prompt di Sistema" (la variabile `stringHeader`).
- Queste istruzioni obbligano l'IA a comportarsi come un mediatore: se l'utente è stato aggressivo o offensivo, l'IA deve riscrivere la frase in modo gentile.
- Viene effettuata una chiamata API verso il server Ollama, inviando la frase grezza e ricevendo quella "pacificata".

#### 4. Output dei Risultati

Infine, il programma mostra a video il confronto:

- Testo originale: Mostra esattamente cosa ha capito il microfono (errori inclusi).
- Frase pacificata: Mostra la versione rielaborata dall'IA, pronta per essere utilizzata in un contesto formale o sicuro.

#### Requisiti fondamentali:

VERSIONE LOCALE:

- **Hardware:** Microfono funzionante ed un qualsiasi PC.
- **Software:** Python 3.13+
- **Modulo Richiesto:** Vosk 0.3.31+, SoundDevice, Requests (installazione in GUIDA ALL'INSTALLAZIONE)

VERSIONE WEB:

- **Hardware:** Pc

#### Guida all'Installazione per la versione in locale (Setup):

Dopo aver installato Python 3.13+, nel CMD bisogna inserire questa riga di testo:

```
pip install requests sounddevice numpy vosk
```

**Nota:** Su Linux, `sounddevice` potrebbe richiedere anche l'installazione di `libportaudio` tramite i seguenti comandi:

- UBUNTU BASED SYSTEM:
 

```
sudo apt-get update
sudo apt-get install libportaudio2
```

- FEDORA BASED SYSTEM:  
**sudo dnf install portaudio**
- ARCH BASED SYSTEM:  
**sudo pacman -S portaudio**

**Successivamente bisogna solo avviare il codice fornito.**

**Nota: Per la versione Web bisogna solo premere il tasto “Provalo Sul Browser” sul sito web.**

## **Documentazione del Codice: Analisi Riga per Riga**

`import requests` Importa la libreria per effettuare richieste HTTP (usata per comunicare con le API di Ollama).

`import sounddevice as sd` Importa la libreria per gestire l'input audio dal microfono e la riproduzione.

`import numpy as np` Importa NumPy per la gestione dei dati numerici e degli array (fondamentale per processare l'audio).

`import json` Importa la libreria per codificare e decodificare dati in formato JSON.

`from vosk import Model, KaldiRecognizer` Importa le classi necessarie da Vosk per caricare il modello linguistico e riconoscere il parlato.

`def ollama_chat_simple(api_key, message, model="cogito-2.1:671b"):`  
Definisce la funzione per inviare un messaggio al modello LLM locale tramite Ollama.

`url = "https://ollama.com/api/chat"` Definisce l'endpoint API a cui inviare la richiesta (Nota: solitamente per Ollama locale è `http://localhost:11434/api/chat`).

`headers = {` Inizia la definizione del dizionario contenente le intestazioni della richiesta HTTP.

`"Content-Type": "application/json",` Specifica che il corpo della richiesta sarà inviato in formato JSON.

`"Authorization": f"Bearer {api_key}"` Inserisce la chiave API nell'intestazione per l'autenticazione.

`}` Chiude il dizionario delle intestazioni.

`data = {` Inizia la definizione del corpo della richiesta (payload).

`"model": model,` Specifica il nome del modello da utilizzare (es. "cogito-2.1:671b").

```
"messages": [ Inizia la lista dei messaggi per la conversazione.  
{"role": "user", "content": message} Definisce il messaggio dell'utente con il  
relativo contenuto.  
, Chiude la lista dei messaggi.  
  
"temperature": 0.7, Imposta il grado di creatività della risposta (0.7 è un valore  
bilanciato).  
  
"max_tokens": 800, Limita la lunghezza massima della risposta generata.  
  
"stream": False Disabilita lo streaming, chiedendo al server di inviare la risposta  
completa in una volta sola.  
}  
} Chiude il dizionario dei dati della richiesta.  
  
try: Inizia un blocco di gestione errori per intercettare problemi durante la chiamata API.  
  
response = requests.post(url, json=data, headers=headers) Invia la  
richiesta POST all'indirizzo specificato con i dati e le intestazioni definiti.  
  
response.raise_for_status() Solleva un'eccezione se la risposta HTTP indica un  
errore (es. 404 o 500).  
  
result = response.json() Converte il corpo della risposta JSON ricevuta in un  
dizionario Python.  
  
if 'message' in result: Controlla se la chiave 'message' è presente nella risposta  
(formato standard di Ollama Chat).  
  
return result['message'][ 'content' ] Restituisce il testo contenuto nel messaggio  
della risposta.  
  
elif 'response' in result: Controlla se esiste invece la chiave 'response' (usata in  
altri tipi di endpoint Ollama).  
  
return result['response'] Restituisce il contenuto testuale della risposta alternativa.  
  
else: Viene eseguito se nessuno dei formati attesi è presente nella risposta.  
  
return f"Formato risposta non riconosciuto: {json.dumps(result,  
indent=2)[:200]}..." Restituisce un messaggio di errore mostrando una parte della  
risposta ricevuta.  
  
except requests.exceptions.HTTPError as e: Cattura errori specifici legati al  
protocollo HTTP.
```

```
return f"Errore HTTP: {e.response.status_code} - {e.response.text}"
```

Restituisce i dettagli del fallimento della richiesta web.

```
except Exception as e: Cattura qualsiasi altro tipo di errore generico.
```

```
return f"Errore: {str(e)}" Restituisce la descrizione testuale dell'errore avvenuto.
```

```
model = Model("modelli/vosk-model-it-0.22") Inizializza il modello di riconoscimento vocale Vosk puntando alla cartella locale dove risiede.
```

```
samplerate = 16000 Imposta la frequenza di campionamento a 16kHz (standard richiesto da molti modelli Vosk).
```

```
rec = KaldiRecognizer(model, samplerate) Crea l'oggetto ricognitore che utilizzerà il modello e la frequenza impostati.
```

```
rec.SetWords(True) Configura il ricognitore affinché includa informazioni sulle singole parole nel risultato.
```

```
testo_finale = "" Inizializza una stringa vuota che conterrà la trascrizione finale.
```

```
DURATA = 4 Definisce la variabile per la durata della registrazione audio (4 secondi).
```

```
print("Parla per", DURATA, "secondi...") Stampa a video un messaggio per avvisare l'utente di iniziare a parlare.
```

```
audio = sd.rec( Avvia la registrazione audio in modo non bloccante.
```

```
int(DURATA * samplerate), Calcola il numero totale di campioni da registrare (durata * frequenza).
```

```
samplerate=samplerate, Specifica la frequenza di campionamento per la registrazione.
```

```
channels=1, Imposta la registrazione in modalità Mono (singolo canale).
```

```
dtype="int16" Imposta il formato dei dati audio a interi a 16 bit (richiesto da Vosk).
```

```
) Chiude la funzione di registrazione.
```

```
sd.wait() Blocca l'esecuzione del programma finché la registrazione non è completata.
```

```
audio_bytes = audio.tobytes() Converte l'array NumPy contenente l'audio in una sequenza di byte grezzi.
```

```
rec.AcceptWaveform(audio_bytes) Passa i byte audio al ricognitore Vosk per l'elaborazione.
```

```
final = json.loads(rec.FinalResult()) Ottiene il risultato finale della trascrizione e lo converte da stringa JSON a dizionario.
```

```
testo_finale += final.get("text", "") Estrae il testo trascritto e lo aggiunge alla variabile testo_finale.
```

```
API_KEY = "*****" Memorizza la chiave API per l'autenticazione verso il servizio Ollama.
```

```
sistemazioneFrase_prompt = "ti passo una frase captata da un microfono..." Definisce il prompt di sistema per correggere la sintassi della frase (commentato nel codice attivo).
```

```
stringHeader = "ti passo una frase che potrebbe contenere qualcosa di offensivo..." Definisce le istruzioni per l'intelligenza artificiale affinché renda la frase educata e pacifica.
```

```
frase_pacificata = ollama_chat_simple(API_KEY, stringHeader + testo_finale) Invia il prompt e la trascrizione vocale alla funzione API e salva il risultato "ripulito".
```

```
print("\nHai detto:") Stampa un'intestazione per mostrare all'utente cosa è stato trascritto.
```

```
print(testo_finale) Stampa il testo originale riconosciuto dal microfono.
```

```
print("\n\nFrase pacificata:") Stampa un'intestazione per il risultato elaborato dall'IA.
```

```
print(frase_pacificata) Stampa la versione corretta e pacata della frase restituita dal modello.
```

## Documentazione Terminata: Grazie per la Lettura