































TECNICHE E TECNOLOGIE AI

corso oltre formazione

SOMMARIO

- 1)  Tecniche di sviluppo
- 2)  Formula della Similarità del Coseno
- 3)  1 Importazione delle librerie
- 4)  2 Creazione dell'app Flask
- 5)  3 Percorso del file dataset.json
- 6)  4 Caricamento del dataset
- 7)  5 Creazione del vocabolario
- 8)  6 Funzione vectorize(text)
- 9)  7 Funzione cosine_similarity(vec1, vec2)
- 10)  8 Rotta principale /
- 11)  9 Rotta /chat — il cuore del chatbot
- 12)  10 Avvio dell'app
- 13)  Riassunto finale
- 14)  INTERFACCIA WEB DEL CHATBOT (index.html)
- 15)  Struttura generale (HTML)
- 16)  Parte di stile (CSS)
- 17)  Aspetto generale
- 18)  Contenitore della chat
- 19)  Area messaggi
- 20)  Area di input
- 21)  Parte logica (JavaScript)
- 22)  1 Riferimenti al DOM
- 23)  2 Funzione addMessage(text, sender)
- 24)  3 Funzione sendMessage()
- 25)  4 Gestione errori
- 26)  5 Creazione del vocabolario
- 27)  6 Focus automatico
- 28)  Funzionalità Complete
- 29)  Conclusione
- 30)  Utilizzo di ChatGPT per la generazione del codice

◆ Introduzione

In questo progetto abbiamo realizzato un'applicazione di **Intelligenza Artificiale (AI)** utilizzando il linguaggio **Python** e il framework **Flask**.

L'obiettivo è stato quello di creare un **chatbot intelligente**, capace di riconoscere frasi simili e rispondere in modo coerente, utilizzando tecniche di **Machine Learning** semplificate.

La tecnica principale utilizzata è il **calcolo della similarità del coseno**, che misura quanto due frasi (vettori di parole) sono simili tra loro, indipendentemente dalla loro lunghezza o grandezza.

SOMMARIO

Tecniche di sviluppo

Per costruire il chatbot sono stati utilizzati i seguenti strumenti e concetti:

- **Python:** linguaggio di programmazione principale.
 - **Flask:** framework web per creare l'interfaccia e gestire la comunicazione tra server e utente.
 - **JSON:** formato usato per il dataset (domande e risposte predefinite).
 - **Similarità del coseno:** tecnica per confrontare due frasi trasformate in vettori numerici.
 - **Tokenizzazione e vettorizzazione:** trasformazione del testo in numeri per poter applicare calcoli matematici.
-



Formula della Similarità del Coseno

La **similarità del coseno** misura l'angolo tra due vettori e determina quanto essi sono simili, ignorando la loro lunghezza.

del coseno $\text{Similarità del coseno} = \frac{A \cdot B}{|A| \times |B|}$

Dove:

- A e B sono i vettori delle frasi da confrontare
- $A \cdot B$ è il **prodotto scalare** dei due vettori
- $|A|$ e $|B|$ sono le **norme euclidee** dei due vettori

$\text{similarità del coseno} = \frac{A \cdot B}{(|A| * |B|)}$

$\text{similarità del coseno} = \frac{\{A \cdot B\}}{\{|A| \cdot |B|\}}$

$\text{similarità del coseno} = \frac{\{A \cdot B\}}{\{|A| \cdot |B|\}}$

$A \cdot B$ = prodotto scalare dei vettori A e B

$|A| = \sqrt{a_1^2 + a_2^2 + \dots + a_n^2}$

$|B| = \sqrt{b_1^2 + b_2^2 + \dots + b_n^2}$

$\cos(\theta) = \frac{\sum_{i=1}^n (a_i * b_i)}{\sqrt{\sum_{i=1}^n (a_i^2)} * \sqrt{\sum_{i=1}^n (b_i^2)}}$

formula

$\cos(\theta) = \frac{(\sum a_i \cdot b_i)}{(\sqrt{\sum a_i^2} \times \sqrt{\sum b_i^2})}$

prima abbiamo bisogno dei dataset per potere fare funzionare chat boot [dataset.json](#)

esempio

```
[
  {
    "input": "Ciao come stai",
    "output": "Ciao! Sto bene, grazie."
  },
  {
    "input": "Salve come va",
    "output": "Va tutto bene, grazie!"
  }
]
```

```

},
{
  "input": "Buongiorno a te",
  "output": "Buongiorno, come posso aiutarti?"
},
{
  "input": "Come stai oggi",
  "output": "Oggi sto alla grande, grazie!"
}
,
{
  "input": "chi seii",
  "output": "Sono una chat ai programmata!"
}
]

```

SOMMARIO

codice python



1 Importazione delle librerie

```

from flask import Flask, render_template, request, jsonify
import json
import math
import os

```

Flask: è il framework che permette di creare un'applicazione web in Python.

- `render_template`: serve per caricare pagine HTML (come `index.html`).
- `request`: serve per leggere i dati inviati dal browser (come il messaggio dell'utente).
- `jsonify`: serve per inviare una risposta in formato JSON al browser.
- `json`: per leggere file JSON (in questo caso, il dataset con le domande e risposte).
- `math`: per operazioni matematiche (radice quadrata, potenze, ecc.).
- `os`: per gestire i percorsi dei file in modo portabile.

SOMMARIO

app



2 Creazione dell'app Flask

```

app = Flask(__name__)

```

Percorso dataset file

3 Percorso del file dataset.json

```
BASE_DIR = os.path.dirname(os.path.abspath(__file__))  
DATA_PATH = os.path.join(BASE_DIR, 'dataset.json')
```

BASE_DIR: ottiene la cartella dove si trova questo file Python.

- DATA_PATH: crea il percorso completo verso il file `dataset.json` (che contiene le frasi di esempio per il chatbot).

👉 Serve per assicurarsi che il programma trovi il file anche se viene eseguito da un'altra directory.

Caricamento

4 Caricamento del dataset

```
with open(DATA_PATH, 'r', encoding='utf-8') as f:  
    dataset = json.load(f)
```

Apri il file `dataset.json` e lo carica nella variabile `dataset`.

- `json.load(f)` legge il contenuto JSON e lo trasforma in una **lista di dizionari Python**.
- Ogni elemento del dataset è tipicamente fatto così:

```
{ "input": "ciao", "output": "ciao, come posso aiutarti?" }
```

SOMMARIO

Vocabolario

5 Creazione del vocabolario

```
vocabulary = {word for entry in dataset for word in entry['input'].lower().split()}
```

Questa riga costruisce il vocabolario del chatbot, cioè l'elenco di tutte le parole uniche trovate negli "input" del dataset.

Esempio:

```
dataset = [  
    {"input": "ciao", "output": "ciao!"},
```

```
{ "input": "come stai", "output": "bene, grazie!" }  
]
```

Risultato:

```
vocabulary = {"ciao", "come", "stai"}
```

Questo vocabolario servirà per rappresentare le frasi come vettori numerici.

SOMMARIO

Vettore della frase

6 Funzione **vectorize(text)**

```
def vectorize(text):  
    tokens = text.lower().split()  
    return [tokens.count(word) for word in vocabulary]
```

👉 Converti una frase in un **vettore di numeri**.

Ogni numero rappresenta quante volte una parola del vocabolario compare nella frase.

Esempio:

- `vocabulary = ["ciao", "come", "stai"]`
- `text = "ciao come come"`

Allora:

```
vectorize(text) = [1, 2, 0]
```

Significa:

- "ciao" compare 1 volta
 - "come" compare 2 volte
 - "stai" compare 0 volte
-

SOMMARIO

Funzione

7 Funzione **cosine_similarity(vec1, vec2)**

```
def cosine_similarity(vec1, vec2):  
    dot_product = sum(a * b for a, b in zip(vec1, vec2))  
    mag1 = math.sqrt(sum(a * a for a in vec1))  
    mag2 = math.sqrt(sum(b * b for b in vec2))
```

```
return dot_product / (mag1 * mag2) if mag1 and mag2 else 0
```

👉 Calcola la **similarità del coseno** tra due vettori, cioè quanto due frasi sono simili tra loro.

Formula:

$\text{similarita} = \frac{\text{dot_product}}{\|A\| \cdot \|B\|}$

Significato:

- dot_product: prodotto scalare tra i due vettori.
- mag1, mag2: lunghezza (norma) dei due vettori.
- Il risultato è un numero **tra 0 e 1**:
 - 1 = frasi identiche
 - 0 = completamente diverse

SOMMARIO

Rotte principali

8 Rotta principale /

```
@app.route('/')
def index():
    return render_template('index.html')
```

- Quando visiti la pagina principale (<http://localhost:5000/>), Flask mostra il file `index.html`.
- Serve come **interfaccia utente** del chatbot (la pagina web dove l'utente scrive i messaggi).

SOMMARIO

Rotta chat boot

9 Rotta /chat — il cuore del chatbot

```
@app.route('/chat', methods=['POST'])
def chat():
    user_input = request.json.get('message', '')
    user_vec = vectorize(user_input)

    best_sim = 0
    best_resp = "Non ho capito, puoi ripetere?"

    for entry in dataset:
        entry_vec = vectorize(entry['input'])
        sim = cosine_similarity(user_vec, entry_vec)
```

```
if sim > best_sim:
    best_sim = sim
    best_resp = entry['output']

return jsonify({"response": best_resp})
```

Cosa fa passo per passo:

1. **Riceve il messaggio dell'utente** dal browser in formato JSON:

```
{"message": "ciao"}
```

2. **Trasforma il testo in vettore:**

```
user_vec = vectorize(user_input)
```

3. **Inizializza variabili:**

- `best_sim = 0` → tiene traccia della similarità più alta trovata finora
- `best_resp = "Non ho capito..."` → risposta di default se non trova nulla di simile

4. **Ciclo su tutte le frasi del dataset:**

- Converte ogni `entry['input']` in vettore (`entry_vec`)
- Calcola la **similarità del coseno** con l'input dell'utente
- Se questa similarità è maggiore della precedente, aggiorna:
 - `best_sim` (nuovo valore massimo)
 - `best_resp` (la risposta associata)

5. **Alla fine restituisce la risposta migliore** come JSON:

```
{"response": "ciao, come posso aiutarti?"}
```

SOMMARIO

Avvio

**10**

Avvio dell'app

```
if __name__ == '__main__':
    app.run(debug=True)
```

Significa:

- Se il file viene eseguito direttamente (non importato), Flask avvia il server.
 - `debug=True` permette di aggiornare automaticamente il server quando cambi il codice e mostra errori dettagliati.
-

💡 Riassunto finale

Parte del codice	Funzione
Import	Carica le librerie necessarie
Percorsi (BASE_DIR)	Trova e apre <code>dataset.json</code>
Vocabolario	Elenco di tutte le parole uniche
vectorize()	Trasforma il testo in vettore numerico
cosine_similarity()	Calcola la somiglianza tra due frasi
Route /	Mostra la pagina web del chatbot
Route /chat	Riceve il messaggio, calcola la risposta e la invia al browser
app.run()	Avvia il server web Flask

💻 INTERFACCIA WEB DEL CHATBOT (index.html)

◆ Introduzione

Questo file rappresenta la **pagina web** che l'utente vede e usa per interagire con il chatbot creato in Python (Flask).

Contiene **HTML**, **CSS** e **JavaScript**, che insieme permettono di:

- mostrare la chat in modo moderno e reattivo,
- inviare i messaggi al server Flask,
- ricevere e visualizzare la risposta dell'intelligenza artificiale,
- far "parlare" il bot grazie alla **sintesi vocale (text-to-speech)**.

🧱 Struttura generale (HTML)

```
<!DOCTYPE html>
<html lang="it">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Chat AI Moderna</title>
```

- **DOCTYPE** e **<html lang="it">** → definiscono il documento HTML in lingua italiana.
- **meta charset="UTF-8"** → permette di usare caratteri speciali come accenti.
- **meta viewport** → rende la pagina adattabile anche su smartphone.
- **title** → titolo della pagina che compare nella scheda del browser.

🎨 Parte di stile (CSS)

Tutta la sezione **<style>** serve a rendere la chat **grafica, moderna e piacevole**.

Ecco i punti principali:

● Aspetto generale

```
body {  
  background: #121214;  
  font-family: 'Inter', sans-serif;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  height: 100vh;  
}
```

👉 Sfondo scuro, testo moderno, chat centrata al centro dello schermo.

💬 Contenitore della chat

```
.chat-container {  
  width: 420px;  
  height: 600px;  
  background: #1E1E1E;  
  border-radius: 15px;  
  display: flex;  
  flex-direction: column;  
  box-shadow: 0 8px 24px rgba(0,0,0,0.4);  
}
```

👉 Finestra principale della chat: forma rettangolare, bordi arrotondati, ombra e disposizione verticale.



Intestazione

```
.chat-header {  
  background: #2E2E2E;  
  color: white;  
  font-weight: 700;  
  text-align: center;  
}
```

👉 Barra superiore con il titolo “Chat AI Moderna”.

💬 Area messaggi

```
.chat-messages {  
  flex-grow: 1;  
  padding: 20px;  
  overflow-y: auto;  
  display: flex;  
  flex-direction: column;  
  gap: 12px;  
}
```

👉 Zona dove compaiono i messaggi (utente e bot).
Ha **scroll automatico** e spaziatura uniforme tra i messaggi.

Messaggi

```
.message.user { ... }  
.message.bot { ... }
```

- **Messaggi utente:** sfondo azzurro, allineati a destra.
 - **Messaggi bot:** sfondo grigio scuro, allineati a sinistra.
 - Entrambi hanno un effetto di **comparsa animata (fadeIn)**.
-

Area di input

```
.chat-input {  
  background: #2E2E2E;  
  padding: 15px 20px;  
  display: flex;  
  gap: 12px;  
}
```

👉 La barra in basso dove l'utente scrive il messaggio e preme "Invia".

- L'<input> ha stile moderno e sfondo scuro.
 - Il **bottone** cambia colore quando si passa sopra col mouse.
-

Parte logica (JavaScript)

Tutta la parte <script> serve a gestire il comportamento della chat.

1 Riferimenti al DOM

```
const chatMessages = document.getElementById('chatMessages');  
const input = document.getElementById('message');  
const form = document.getElementById('chatForm');
```

Queste variabili servono per accedere agli elementi HTML della chat (finestra messaggi, campo testo e form).

2 Funzione addMessage(text, sender)

```
function addMessage(text, sender) {  
  const div = document.createElement('div');  
  div.classList.add('message', sender);  
  div.textContent = text;  
  chatMessages.appendChild(div);  
  chatMessages.scrollTop = chatMessages.scrollHeight;  
}
```

📌 Crea e mostra un messaggio nella finestra chat.

- sender può essere "user" o "bot", per cambiare lo stile.
 - Aggiorna automaticamente la **scrollbar** in fondo.
-

3 Funzione sendMessage()

```
async function sendMessage() {  
  const msg = input.value.trim();  
  if (!msg) return;  
  addMessage(msg, 'user');  
  input.value = '';  
  input.disabled = true;
```

📌 Quando l'utente invia un messaggio:

1. Lo mostra subito nella chat.
2. Blocca l'input per evitare altri invii.
3. Invia il testo al server Flask con una **richiesta POST**.

```
const response = await fetch('/chat', {  
  method: 'POST',  
  headers: {'Content-Type': 'application/json'},  
  body: JSON.stringify({message: msg})  
});  
const data = await response.json();  
addMessage(data.response, 'bot');  
speak(data.response);
```

✉️ Flask riceve il messaggio, calcola la risposta, e la rimanda in formato JSON.
La chat mostra il messaggio del bot e lo fa “parlare” (funzione **speak**).

4 Gestione errori

```
catch {  
  addMessage('Errore di connessione, riprova.', 'bot');  
}
```

👉 Se il server non risponde o c'è un errore, il bot avvisa l'utente.

5 Sintesi vocale

```
function speak(text) {  
  if ('speechSynthesis' in window) {  
    const utterance = new SpeechSynthesisUtterance(text);  
    window.speechSynthesis.speak(utterance);  
  }  
}
```

👉 Usa la **voce del browser** per leggere la risposta del chatbot.
Funziona su **Chrome, Edge, Safari** ecc.

6 Focus automatico

```
input.focus();
```

👉 Appena la pagina si apre, il cursore è già nel campo di testo, pronto per scrivere.

Funzionalità Complete

Funzione	Descrizione
<code>addMessage()</code>	Mostra messaggi nella finestra chat
<code>sendMessage()</code>	Invia il messaggio al server Flask
<code>speak()</code>	Fa parlare il bot con la voce sintetica
CSS moderno	Grafica chiara, scura e responsiva
Fetch API	Comunicazione tra front-end e back-end
SpeechSynthesis API	Sintesi vocale integrata nel browser

Conclusione

Questa parte HTML/JS rappresenta il **front-end** del progetto, cioè la parte visiva e interattiva. Collegata al **back-end Python (Flask)**, permette di creare un **chatbot AI funzionante, parlante e con interfaccia moderna**.

L'unione delle due parti (Python + HTML/CSS/JS) realizza un **progetto completo di intelligenza artificiale**, che dimostra competenze in:

- **Sviluppo web (Flask + HTML/CSS/JS)**
- **Machine Learning di base (similarità del coseno)**
- **Elaborazione del linguaggio naturale**
- **Integrazione tra front-end e back-end**

Utilizzo di ChatGPT per la generazione del codice

Durante la realizzazione del progetto, abbiamo utilizzato **ChatGPT**, un modello di intelligenza artificiale sviluppato da **OpenAI**, per generare, correggere e migliorare il codice sorgente del chatbot.

Abbiamo fornito a ChatGPT diversi **prompt** (istruzioni testuali) per ottenere:

- esempi di codice Python con **Flask** per creare un server web;
- funzioni per il **calcolo della similarità del coseno**;
- esempi di file **dataset.json** per addestrare il chatbot;
- codice **HTML, CSS e JavaScript** per costruire un'interfaccia utente moderna e funzionale;
- spiegazioni teoriche sul funzionamento delle varie parti del progetto.

L'intelligenza artificiale è stata utilizzata come **strumento di supporto alla programmazione**, consentendoci di:

- risparmiare tempo nella scrittura del codice;
- comprendere meglio la logica delle funzioni;
- migliorare la qualità e la leggibilità del progetto.

Inoltre, ChatGPT è stato impiegato per redigere la **documentazione tecnica**, spiegare il codice passo per passo e creare una relazione chiara e completa.