Corso di ingegneria informatica
A.A. 2023/2024

Interrogazione Intelligente: Dall'OCR al Dialogo Contestualizzato con ChatGPT

Nome dello studente:	*Manuel Frasca*
Matricola	*744HHHINGINFOR*
Data Appello	*Dicembre 2023*
Esame	*Intelligenza Artificiale*

Indice

Premessa / Introduzione

L'avvento dell'elaborazione digitale del testo rappresenta un punto cruciale nel panorama della tecnologia informativa. Il titolo "Interrogazione Intelligente" si configura come un sistema che trasforma il Riconoscimento Ottico dei Caratteri (OCR) da semplice strumento di digitalizzazione testuale in un interlocutore capace di dialogo contestualizzato. Questa trasformazione rappresenta una evoluzione nell'interazione uomo-macchina, avanzando verso la comprensione profonda e l'intelligenza artificiale applicata.

Al nucleo di questo progetto c'è un meccanismo di OCR avanzato che, attraverso l'applicazione di filtri mirati e una raffinata metodologia di preprocessing, crea triple immagini filtrate per affrontare con precisione le variazioni di qualità e complessità visiva. L'unicità di questo progetto risiede nella sua capacità di non solo riconoscere e trascrivere il testo da vari supporti ma di affinarne la qualità mediante un'accurata valutazione basata su algoritmi di intelligenza artificiale, che esaminano lingua e coerenza per determinare la fedeltà del testo estratto.

Successivamente, il testo migliore viene eletto attraverso un processo di selezione quantitativo-qualitativo, permettendo così di procedere verso la fase di dialogo. In questa fase, il testo viene arricchito con domande mirate e inoltrato a ChatGPT, che interpreta, risponde e amplia il contenuto in una dinamica conversazione.

Il risultato è un dialogo contestualizzato dove l’utente ha a disposizione tutti gli strumenti necessari per avere un’immediata interazione con lo strumento più potente di intelligenza artificiale attualmente presente nel mercato.

Svolgimento

Ambiente di Sviluppo e Configurazione

Il progetto di riconoscimento ottico dei caratteri avanzato è stato sviluppato utilizzando Python nella versione 3.8.18, scelta per la sua robustezza e la vasta disponibilità di librerie di supporto.

L'Integrated Development Environment (IDE) adottato è stato Visual Studio Code (VSCode), in dettaglio la versione 1.84.0. Questo ambiente di sviluppo è stato selezionato per le sue avanzate funzionalità di codifica, il suo supporto esteso per il linguaggio Python e la possibilità di integrarsi con una varietà di estensioni che facilitano lo sviluppo di software complesso. VSCode fornisce un'interfaccia utente intuitiva e un ampio insieme di strumenti per il debugging e la gestione del codice, migliorando l'efficienza dello sviluppo e la qualità del prodotto software finale.

L'IDE utilizzato è basato su una serie di componenti:

- **Electron**: 25.9.2
- **ElectronBuildId**: 24603566
- **Chromium**: 114.0.5735.289
- **Node.js**: 18.15.0
- **V8**: 11.4.183.29-electron.0
- **Sistema Operativo**: Linux x64 6.2.0-36-generic

Il sistema operativo di base per lo sviluppo e il testing del software è stato Ubuntu 22.04.3 LTS. Ubuntu è stato scelto per la sua stabilità, sicurezza e supporto esteso. La distribuzione LTS (Long Term Support) garantisce aggiornamenti e supporto per un periodo prolungato, rendendola una scelta affidabile per lo sviluppo di applicazioni.

Configurazione del Sistema Operativo

Il sistema operativo Ubuntu è stato configurato con le seguenti specifiche:

- **Distributor ID**: Ubuntu
- **Description**: Ubuntu 22.04.3 LTS
- **Release**: 22.04
- **Codename**: jammy

Librerie e Framework
Le librerie e i framework sono stati scelti con attenzione per garantire l'efficienza del processo di riconoscimento ottico dei caratteri e il successivo trattamento dei dati. Questi strumenti sono stati fondamentali per l'implementazione delle funzionalità di preprocessing delle immagini, l'esecuzione dell'OCR e la valutazione della qualità del testo estratto. Tutte le librerie sono state selezionate per la loro comprovata affidabilità, ampio supporto della comunità e compatibilità con la versione di Python adottata.

Dettaglio delle Librerie e Framework Utilizzati
Nel contesto del progetto di riconoscimento ottico dei caratteri e l'integrazione con ChatGPT per la creazione di un dialogo contestualizzato, sono state impiegate una serie di librerie Python. Ognuna di queste gioca un ruolo cruciale nelle diverse fasi di elaborazione, dalla cattura dell'input fino alla generazione di risposte pertinenti. Di seguito vengono descritte le librerie principali:

1. **pynput (v0.13.5)**: Una libreria essenziale per monitorare e controllare dispositivi di input come la tastiera e il mouse. In questo progetto, pynput è stata utilizzata per rilevare la pressione di un tasto specifico che innesca la cattura dello schermo o altre azioni di interazione con l'utente.
2. **pyscreenshot (v3.1)**: Questa libreria permette di catturare lo schermo del computer. In combinazione con pynput, ha reso possibile acquisire in tempo reale le immagini sullo schermo che successivamente vengono sottoposte a riconoscimento ottico dei caratteri.
3. **Pillow (PIL Fork) (v10.1.0)**: Il fork attivamente mantenuto della Python Imaging Library (PIL) è stato impiegato per aprire, manipolare e salvare molte differenti formati di file immagine. Nell'ambito del progetto, Pillow è stata utilizzata per la gestione delle immagini estratte, come la conversione in scala di grigi e l'applicazione di filtri necessari a migliorare la qualità dell'OCR.
4. **opencv-python (v4.8.1.78)**: OpenCV è una libreria focalizzata sulla visione artificiale e sul machine learning. È stata utilizzata per trattare le immagini prima della fase di OCR, implementando algoritmi di miglioramento dell'immagine, rilevamento dei bordi e altre trasformazioni necessarie.
5. **pytesseract (v0.3.10)**: Interfaccia Python per Tesseract-OCR, uno dei più famosi motori di OCR open source. pytesseract è stata la scelta principale per convertire le immagini in testo, sfruttando la sua capacità di riconoscere e leggere il testo dalle immagini acquisite.

6. **langdetect (v1.0.9)**: Questa libreria permette di rilevare la lingua di un testo, utilizzata per determinare la lingua del testo estratto dalle immagini prima di procedere con ulteriori elaborazioni o traduzioni se necessario.
7. **pyspellchecker (v0.7.2)**: Utilizzata per identificare e correggere errori ortografici nel testo estratto, migliorando l'affidabilità dei dati inseriti nel sistema di dialogo.
8. **spacy (v3.7.2)**: Una libreria avanzata per il Natural Language Processing (NLP), essenziale per il parsing, la comprensione e l'analisi del testo estratto. Grazie a spacy, è stato possibile analizzare il testo in termini di parti del discorso, entità nominate e altri aspetti strutturali e semantici che contribuiscono a una migliore comprensione del contesto da parte del modello di ChatGPT.
9. **openai (v0.28.1)**: Il client Python per l'interfaccia API di OpenAI. Questa libreria è stata utilizzata per integrare le capacità di conversazione di ChatGPT, permettendo di inviare il testo estratto e ricevere risposte coerenti con il contesto rilevato.
10. **plyer (v2.1.0)**: Una libreria multipiattaforma per la gestione delle funzionalità come le notifiche. Nel progetto, plyer è stato adottato per fornire all'utente feedback visivo e sonoro in risposta a eventi specifici, come la corretta acquisizione di un'immagine o la ricezione di una nuova risposta da ChatGPT.

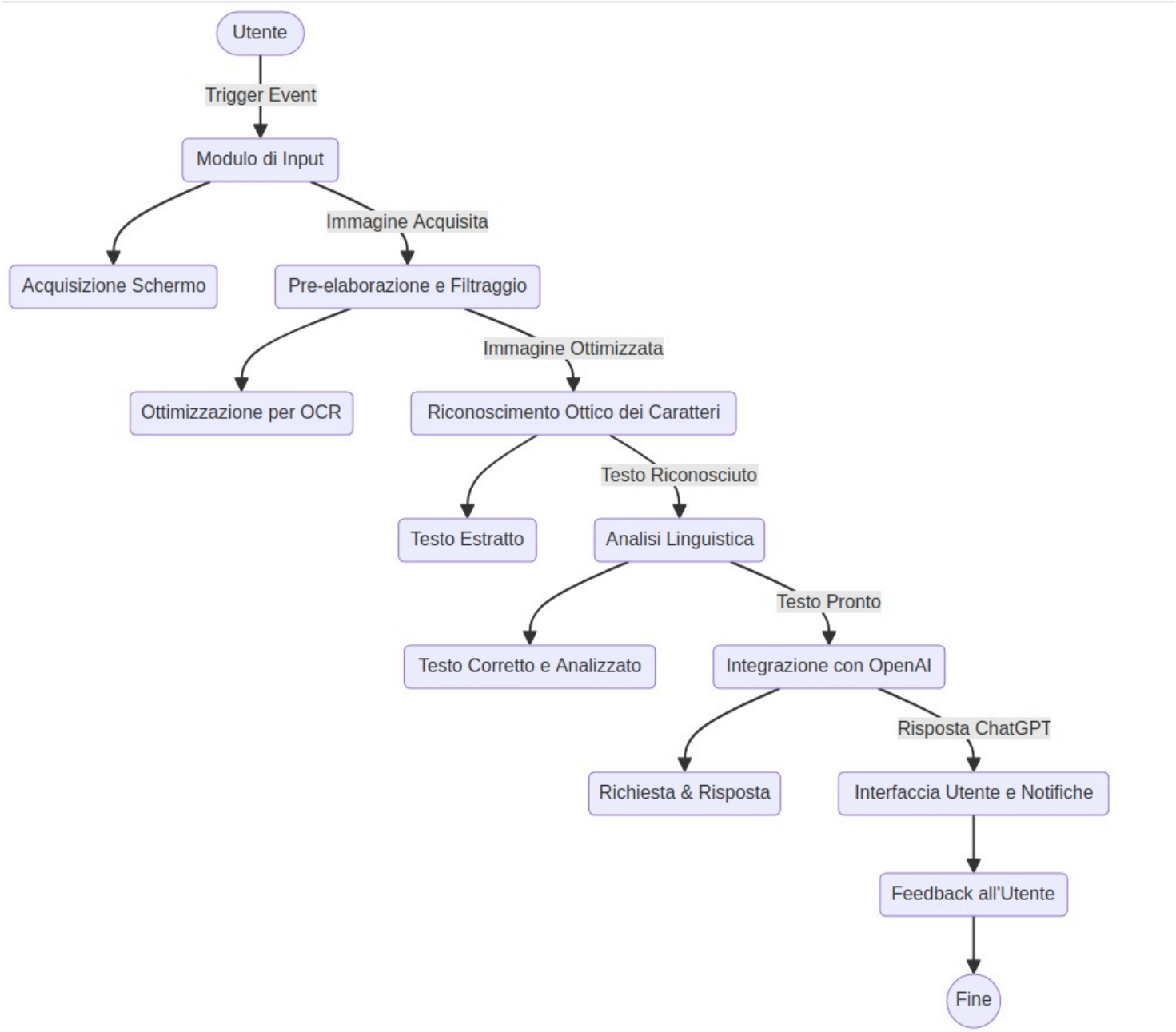
Queste librerie, insieme agli altri moduli installati, formano il nucleo tecnico del progetto, permettendo di elaborare efficacemente i dati da input visivi a testuali e successivamente in un dialogo comprensibile ed efficiente.

Architettura del Sistema
Il sistema è stato progettato per facilitare l'interazione tra l'utente e un'interfaccia di programmazione intelligente fornita da ChatGPT, con l'obiettivo di estrarre testi da immagini (OCR), processarli per la correzione ortografica e l'analisi linguistica, e infine per l'invio di questi a un modello di lingua avanzato per la generazione di risposte contestualizzate.

Descrizione dei Componenti

1. **Modulo di Input (Pynput & PyScreenshot):**
    - Cattura delle immagini dello schermo o input specifici da parte dell'utente.
    - Selezione dell'area dello schermo da processare.
2. **Pre-elaborazione e Filtraggio (OpenCV & Pillow):**
    - Conversione dell'immagine in scala di grigi e ridimensionamento.
    - Applicazione di filtri per migliorare la qualità dell'immagine per l'OCR.
3. **Riconoscimento Ottico dei Caratteri (Pytesseract):**
    - Estrazione del testo dalle immagini pre-elaborate.
    - Riconoscimento e digitalizzazione del contenuto testuale.
4. **Analisi Linguistica (Langdetect & Pyspellchecker):**
    - Determinazione della lingua del testo estratto.
    - Correzione di errori ortografici per garantire l'accuratezza del testo.
5. **Elaborazione del Linguaggio Naturale (Spacy):**
    - Analisi del testo per identificare contesto e struttura.
    - Preparazione del testo per essere utilizzato da modelli di lingua più avanzati.
6. **Integrazione con OpenAI (OpenAI API):**
    - Invio del testo processato a ChatGPT.
    - Ricezione delle risposte basate sul contesto fornito.
7. **Interfaccia Utente e Notifiche (Plyer):**
    - Visualizzazione dei risultati e delle risposte di ChatGPT.
    - Notifiche e feedback all'utente basati sulle interazioni.

Diagramma di Flusso
Per rappresentare visivamente queste interazioni, è utile visualizzare il seguente diagramma di flusso:



Acquisizione e Pre-elaborazione dei Dati

Cattura dello Schermo

Il processo inizia con la cattura dello schermo, un'operazione che permette di acquisire l'immagine contenente il testo da analizzare. Utilizzando la libreria pynput, si monitorano gli eventi della tastiera per attivare la cattura quando l'utente preme un tasto specifico. Subito dopo, entra in gioco pyscreenshot che cattura l'immagine dello schermo, o una porzione di esso, definita dall'utente attraverso una selezione interattiva o parametri preimpostati. Questa flessibilità consente di focalizzare l'OCR su specifiche aree, evitando elementi non necessari che potrebbero compromettere l'accuratezza del riconoscimento del testo.

```
    with keyboard.Listener(on_press=on_press) as listener:
        listener.join()
```

Pre-elaborazione delle Immagini

```
def on_press(key):
    try:
        if key.char == 'c':

            print("Cattura in corso...")
            # Cattura lo schermo
            im = ImageGrab.grab()
            # Salva lo screen
            im.save(screename)
            print("Screen Salvato")
```

Una volta acquisita l'immagine, inizia la fase di pre-elaborazione, che ha l'obiettivo di ottimizzare il materiale grafico per il successivo passaggio di riconoscimento ottico dei caratteri (OCR). La libreria Pillow è impiegata per la conversione dell'immagine in scala di grigi, una trasformazione che riduce la complessità dell'immagine

rimuovendo il colore ma preservando i dettagli necessari per l'identificazione dei caratteri.

Dopo la conversione in scala di grigi, si procede con l'applicazione di filtri di sogliatura attraverso opencv-python. La sogliatura è una tecnica che modifica l'immagine in modo tale che i pixel di interesse (generalmente i pixel che compongono il testo) siano resi più distinguibili dallo sfondo. In termini pratici, viene definito un valore di soglia che determina se un pixel sarà trasformato in nero o bianco, creando così una netta distinzione che facilita il compito dell'algoritmo di OCR.

```
with Image.open(screen) as image:   # Apre l'immagine dal file
    # Converte l'immagine in scala di grigi
    imageBN = image.convert("L")

    # Carica l'immagine utilizzando OpenCV
    img = cv2.imread(screen)
    # Converti l'immagine in scala di grigi
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Applica una sogliatura
    _, thresh_img = cv2.threshold(gray_img, 90, 180, cv2.THRESH_BINARY)
```

- **Apertura dell'Immagine:** La funzione Image.open() del modulo PIL (Pillow) viene utilizzata per aprire l'immagine salvata nel filesystem. Questo è il primo passo per accedere ai dati grafici che verranno poi elaborati.
- **Conversione in Scala di Grigi:** La conversione in scala di grigi è un passo essenziale che riduce la complessità dell'immagine. Questo passaggio è implementato due volte nel codice, una volta con Pillow (image.convert("L")) e una volta con OpenCV (cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)). Anche se sembra ridondante, questo può essere utilizzato per sperimentare e confrontare i risultati della conversione di due differenti librerie.
- **Sogliatura dell'Immagine:** La sogliatura è realizzata attraverso la funzione cv2.threshold(). Il processo trasforma l'immagine in un'immagine binaria dove i pixel possono essere solo neri o bianchi. Questo passaggio migliora il contrasto tra il testo e lo sfondo, facilitando il riconoscimento dei caratteri da parte di Tesseract.

La combinazione di questi passaggi di pre-elaborazione è cruciale per la riuscita del riconoscimento del testo. La conversione in scala di grigi elimina le distrazioni legate ai colori, mentre la sogliatura enfatizza la leggibilità dei caratteri. Questo processo di ottimizzazione è particolarmente importante in ambienti con condizioni di illuminazione variabili o con font e dimensioni di testo diversi, situazioni che possono altrimenti generare errori di riconoscimento da parte dell'OCR.

Attraverso un'attenta regolazione dei parametri di pre-elaborazione, si massimizza la chiarezza del testo nell'immagine risultante, rendendolo il più leggibile possibile per pytesseract, la libreria scelta per eseguire l'OCR. L'accuratezza di pytesseract dipende fortemente dalla qualità dell'immagine in ingresso; di conseguenza, una pre-elaborazione efficace è direttamente proporzionale all'efficacia con cui il testo sarà estratto e interpretato nel passaggio successivo.

Riconoscimento Ottico dei Caratteri (OCR)

Implementazione dell'OCR con Pytesseract

L'implementazione dell'OCR, ossia il Riconoscimento Ottico dei Caratteri, rappresenta il cuore del processo di estrazione del testo dalle immagini elaborate. La libreria pytesseract funge da interfaccia con il motore OCR Tesseract, uno dei più affidabili e precisi algoritmi open-source disponibili per l'OCR.

pytesseract.image_to_string() è il metodo che effettivamente esegue l'OCR. Vengono testate tre versioni dell'immagine: l'originale, quella in scala di grigi gestita da Pillow, e quella sogliata gestita da OpenCV. Questo approccio permette di confrontare e scegliere il risultato migliore.
Il parametro lang='eng' specifica che l'OCR dovrebbe considerare l'inglese come lingua del testo nell'immagine, mentre config='--psm 6 --oem 3' indica la configurazione di Tesseract: --psm 6 stabilisce il Page Segmentation Mode, che influisce su come l'algoritmo interpreta la disposizione del testo nell'immagine; --oem 3 indica il motore OCR Engine Mode da utilizzare, che in questo caso è la modalità di riconoscimento basata sia sul motore legacy che su quello basato su reti neurali LSTM di Tesseract.

OCR.py > perform_ocr_on_image_from_file

```python
import pytesseract
from PIL import Image
import cv2

def perform_ocr_on_image_from_file(screen):
    try:
        with Image.open(screen) as image:   # Apre l'immagine dal file
            # Converte l'immagine in scala di grigi
            imageBN = image.convert("L")

            # Carica l'immagine utilizzando OpenCV
            img = cv2.imread(screen)
            # Converti l'immagine in scala di grigi
            gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            # Applica una sogliatura
            _, thresh_img = cv2.threshold(gray_img, 90, 180, cv2.THRESH_BINARY)

            # pytesseract per effettuare l'OCR sulle immagini pre-processate
            textCV = pytesseract.image_to_string(thresh_img, lang='eng',config='--psm 6 --oem 3')
            text = pytesseract.image_to_string(image, lang='eng',config='--psm 6 --oem 3')
            text1 = pytesseract.image_to_string(imageBN, lang='eng',config='--psm 6 --oem 3')

        return text, text1, textCV
    except Exception as e:
        print(f"Errore: {e}")
        return None
```

Analisi Linguistica e Correzione Ortografica

Garanzia della Correttezza Linguistica e Ortografica

Per assicurare che il testo estratto tramite OCR sia non solo della lingua desiderata ma anche privo di errori ortografici prima del suo processamento tramite la libreria spaCy, è possibile utilizzare strumenti come langdetect per l'identificazione della lingua e pyspellchecker per il controllo ortografico.

Identificazione della Lingua con langdetect

langdetect è una libreria che permette di riconoscere la lingua in cui è scritto un dato testo. Questo può essere particolarmente utile quando si lavora con documenti che potrebbero essere multilingue o quando non si ha a priori la certezza della lingua del documento. Utilizzando langdetect si può automaticamente confermare o determinare la lingua del testo estratto e procedere con il controllo ortografico appropriato.

Nel contesto dell'elaborazione del testo estratto attraverso l'OCR, diventa fondamentale distinguere tra i frammenti di testo in lingua italiana e inglese per poter applicare il controllo ortografico corretto e garantire l'accuratezza del processo di NLP con spaCy. La libreria langdetect offre una soluzione efficace per questo scopo.

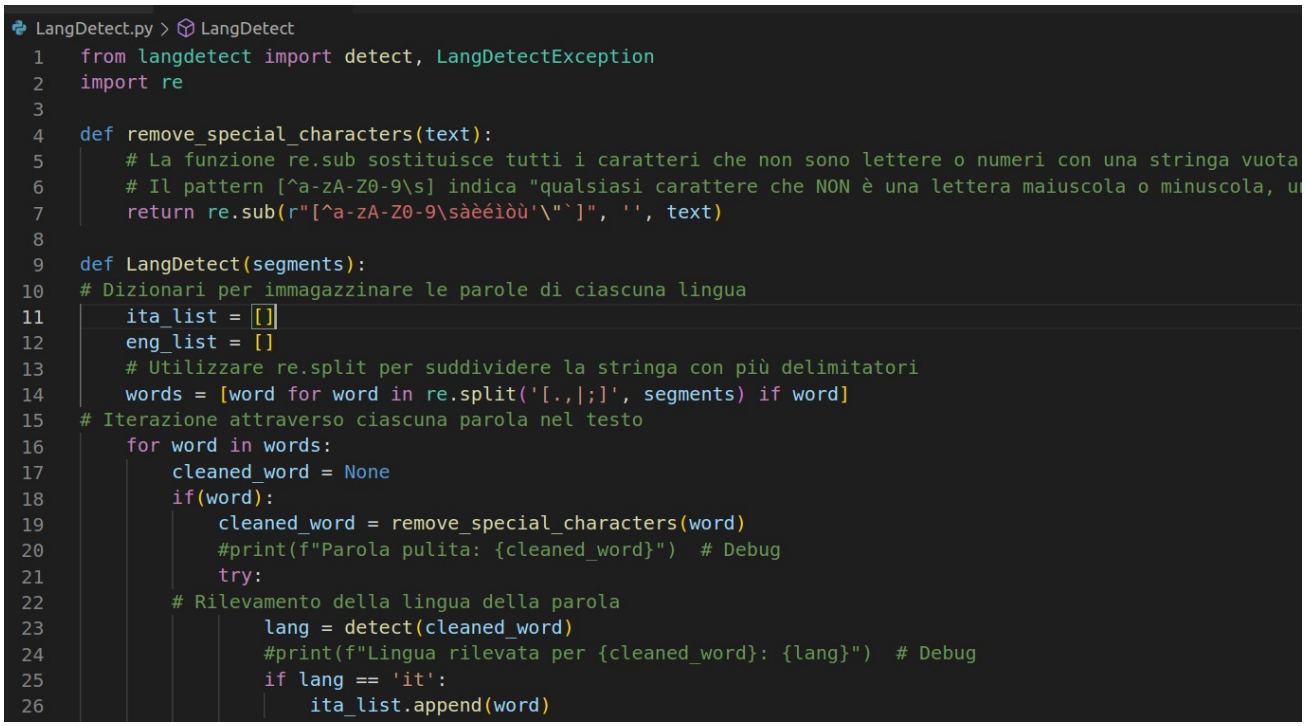
Funzionamento di langdetect

langdetect è una libreria basata sull'algoritmo di rilevamento della lingua Naive Bayes che permette di prevedere la lingua di una determinata porzione di testo. È addestrata su una varietà di lingue e può essere utilizzata per analizzare testi anche brevi, fornendo risultati affidabili.

Implementazione del Rilevamento della Lingua

1. **Pulizia del Testo:** La funzione remove_special_characters() utilizza un'espressione regolare per eliminare dal testo tutti i caratteri speciali che non sono parte del testo scritto regolare, lasciando lettere, numeri e spazi. Questo passaggio è essenziale per ridurre il rumore prima del rilevamento della lingua, migliorando così l'accuratezza di langdetect.
2. **Rilevamento della Lingua per Segmento di Testo:** LangDetect() è la funzione principale che riceve un testo e ritorna due stringhe, una per ogni lingua identificata (italiano e inglese). Utilizza re.split() per suddividere il testo in parole o segmenti basati su delimitatori come punti o virgole. Ogni segmento è poi analizzato individualmente:

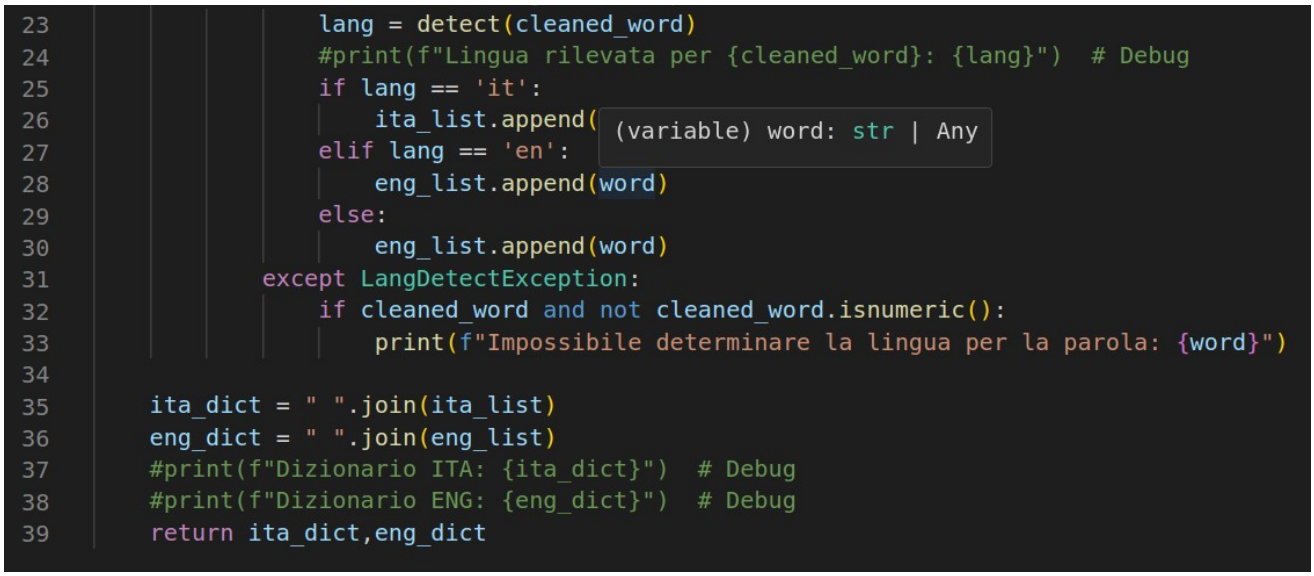
- Se il segmento contiene caratteri, viene prima pulito con remove_special_characters().
- Il segmento pulito viene poi analizzato con detect() per determinare la lingua.
- In base al risultato, il segmento viene aggiunto al dizionario corrispondente, ita_list o eng_list.



```python
from langdetect import detect, LangDetectException
import re

def remove_special_characters(text):
    # La funzione re.sub sostituisce tutti i caratteri che non sono lettere o numeri con una stringa vuota
    # Il pattern [^a-zA-Z0-9\s] indica "qualsiasi carattere che NON è una lettera maiuscola o minuscola, u
    return re.sub(r"[^a-zA-Z0-9\sàèéìòù'\"`]", '', text)

def LangDetect(segments):
# Dizionari per immagazzinare le parole di ciascuna lingua
    ita_list = []
    eng_list = []
    # Utilizzare re.split per suddividere la stringa con più delimitatori
    words = [word for word in re.split('[.,|;]', segments) if word]
# Iterazione attraverso ciascuna parola nel testo
    for word in words:
        cleaned_word = None
        if(word):
            cleaned_word = remove_special_characters(word)
            #print(f"Parola pulita: {cleaned_word}")  # Debug
            try:
        # Rilevamento della lingua della parola
                lang = detect(cleaned_word)
                #print(f"Lingua rilevata per {cleaned_word}: {lang}")  # Debug
                if lang == 'it':
                    ita_list.append(word)
```



```python
                lang = detect(cleaned_word)
                #print(f"Lingua rilevata per {cleaned_word}: {lang}")  # Debug
                if lang == 'it':
                    ita_list.append(
                elif lang == 'en':
                    eng_list.append(word)
                else:
                    eng_list.append(word)
            except LangDetectException:
                if cleaned_word and not cleaned_word.isnumeric():
                    print(f"Impossibile determinare la lingua per la parola: {word}")

    ita_dict = " ".join(ita_list)
    eng_dict = " ".join(eng_list)
    #print(f"Dizionario ITA: {ita_dict}")  # Debug
    #print(f"Dizionario ENG: {eng_dict}")  # Debug
    return ita_dict,eng_dict
```

(variable) word: str | Any

Gestione delle Eccezioni

Una considerazione importante nel rilevamento della lingua è la gestione delle eccezioni, ad esempio, quando detect() non riesce a determinare la lingua di una parola (un evento possibile con parole molto brevi o acronimi). In questi casi, il

codice stampa un messaggio di errore e sceglie di includere la parola nel dizionario delle parole inglesi. La parola successivamente verrà valutata se corretta o meno.

Creazione dei Dizionari Linguistici
Alla fine del processo, le parole vengono raggruppate in base alla lingua rilevata e unite in stringhe separate (ita_dict per l'italiano e eng_dict per l'inglese). Questo permette poi di processare separatamente i segmenti di testo nelle lingue corrispondenti, utilizzando il controllo ortografico adeguato e migliorando la coerenza del testo finale che sarà elaborato da spaCy.

Importanza nella Relazione
Nel contesto della relazione, è importante sottolineare come l'integrazione di langdetect nel flusso di lavoro di elaborazione del testo migliora non solo la precisione del rilevamento linguistico, ma anche la qualità del testo su cui si applicheranno i successivi strumenti di analisi linguistica. Inoltre, la capacità di discernere tra le lingue e gestire le eccezioni aiuta a ridurre significativamente il numero di errori di interpretazione che potrebbero derivare da un controllo ortografico non corretto o da un'analisi NLP inaccurata. Questo aspetto diventa cruciale in documenti bilingue o multilingue, dove la presenza di più lingue può spesso portare a confusione senza una chiara discriminazione linguistica.

Controllo Ortografico con pyspellchecker

pyspellchecker è una libreria Python che fornisce la possibilità di verificare la presenza di errori ortografici e suggerire correzioni. È possibile utilizzarla per una varietà di lingue, garantendo così che il testo estratto sia privo di errori prima di essere analizzato ulteriormente, ad esempio con spaCy. La correttezza ortografica è vitale, poiché gli errori possono distorcere l'analisi semantica e sintattica del testo. Per fare ciò in maniera efficace con testi in italiano, come nel caso esposto, è essenziale caricare un dizionario italiano. Questo è reso possibile caricando una lista di parole italiane da un file e inserendole nel dizionario di SpellChecker.

Implementazione di pyspellchecker

SpellCheck.py > ...

```python
from spellchecker import SpellChecker

# Carica la lista di parole italiane da un file
def load_italian_dictionary(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        italian_words = file.read().splitlines()
    return italian_words


# Inizializza il controllo ortografico con il dizionario italiano
def initialize_spellchecker(italian_words):
    spellchecker = SpellChecker(language=None, case_sensitive=True)
    spellchecker.word_frequency.load_words(italian_words)
    return spellchecker

# Verifica la presenza di errori ortografici e calcola il punteggio
def check_spelling_errors(text, spellchecker):
    # Separare il testo in parole
    words = text.split()
    # Trova le parole che sono errate
    misspelled = spellchecker.unknown(words)
    # Ritorna il numero di errori trovati
    return misspelled

```

Il codice fornito mostra come implementare il controllo ortografico per l'italiano e l'inglese utilizzando pyspellchecker. Il processo è suddiviso in vari passaggi:

1. **Caricamento del Dizionario Italiano:** La funzione load_italian_dictionary() legge da un file un elenco di parole italiane, presumibilmente corrette e complete, creando una lista di parole (italian_words) che servirà come base di riferimento per il controllo ortografico.

```
# Carica spellchecker per la lingua Inglese
spell_en = SpellChecker(language='en')
# Carica il dizionario italiano da un file (devi fornire il percorso al tuo file)
file_path = '660000_parole_italiane.txt'
diz_ita = load_italian_dictionary(file_path)
spell_ita = initialize_spellchecker(diz_ita)
```

2. **Inizializzazione del SpellChecker:** La funzione initialize_spellchecker() crea un'istanza di SpellChecker senza specificare una lingua (language=None), in quanto verrà utilizzato un dizionario personalizzato (in questo caso, quello italiano caricato). Con case_sensitive=True, si assicura che la distinzione tra maiuscole e minuscole sia preservata, fondamentale in lingue come l'italiano dove la capitalizzazione può cambiare il significato delle parole.
3. **Controllo Ortografico Personalizzato:** Dopo aver inizializzato SpellChecker con il dizionario italiano, si procede al controllo ortografico del testo. La funzione check_spelling_errors() divide il testo in parole e utilizza il metodo unknown() per identificare quelle che non corrispondono a nessuna parola nota nel dizionario, restituendo così un insieme di termini potenzialmente errati.

```
# Calcolo degli errori ortografici
misspelled_en = spell_en.unknown(textEN.split())
misspelled_it = check_spelling_errors(textIta.lower(),spell_ita)
```

Applicazione in Differenze Linguistiche

Si procede quindi con un approccio bifocale, utilizzando il controllo ortografico standard di pyspellchecker per l'inglese (spell_en) e quello personalizzato per l'italiano (spell_ita), a seconda della lingua rilevata da langdetect.

Elaborazione del Linguaggio Naturale (NLP)
spaCy gioca un ruolo fondamentale nell'analisi del testo estratto, offrendo robuste funzionalità per il trattamento del linguaggio naturale (NLP) che sono utilizzate per comprendere il contesto e strutturare i dati in maniera tale da poter essere poi inviati a ChatGPT tramite l'API di OpenAI. La funzione check_text_quality() illustra l'uso concreto di spaCy per valutare la qualità di testi in inglese e in italiano, considerando vari aspetti linguistici e stilistici (Calcolo_Qualità.py in Appendice).

Il Ruolo di spaCy nell'Analisi del Testo
spaCy viene utilizzato per eseguire diverse operazioni di NLP essenziali:

- **Tokenizzazione:** Suddivide il testo in token, che possono essere parole, punteggiatura, simboli o spazi. Questo è il primo passo per l'analisi e la comprensione del testo.
- **Etichettatura Morfosintattica (POS tagging):** Assegna a ogni token una categoria grammaticale (sostantivo, verbo, aggettivo, ecc.). Questo aiuta a comprendere la struttura e il ruolo di ogni parola nella frase.
- **Rilevamento delle Frasi (Sentence Detection):** Identifica i confini delle frasi all'interno del testo, permettendo l'analisi delle strutture a livello di frase.
- **Analisi di Dipendenza (Dependency Parsing):** Stabilisce le relazioni di dipendenza tra i token, rivelando la struttura grammaticale del testo e come le parole si combinano per formare significati complessi.

Utilizzo di spaCy per la Comprendere il Contesto e Strutturare i Dati
Il codice in Appendice “Calcolo_Qualità.py” esegue diverse operazioni chiave per valutare la qualità dei testi:

1. **Creazione del Documento:** Crea un oggetto documento spaCy per entrambe le lingue, che contiene informazioni sulla tokenizzazione, etichette grammaticali, e altro.

```
# Caricamento modelli spacy
Inglese = spacy.load("en_core_web_trf") # per la lingua inglese
Ita = spacy.load("it_core_news_lg") # per la lingua italiana
```

2. **Valutazione della Qualità Linguistica:** Utilizza i dati estratti dal parsing per calcolare il punteggio di qualità del testo. Questo si basa sul numero e sulla

tipologia di parole (sostantivi, verbi, aggettivi), che possono indicare la ricchezza e la varietà del testo.

```
# Costanti
NOUN_WEIGHT = 0.45
VERB_WEIGHT = 0.5
ADJ_WEIGHT = 0.4
LEXICAL_DIV_WEIGHT = 0.0 # non utilizzato
AVG_SENT_LEN_WEIGHT = 0.0 # non utilizzato
SPELLING_MISTAKE_WEIGHT = -0.5  # Peso negativo per gli errori ortografici

def check_text_quality(textIta,textEN,return_features=False):
    # Dizionario vuoto da usare come valore di default
    default_features = {}
    docEN = Inglese(textEN)
    docIta = Ita(textIta)

    # Calcolo degli errori ortografici
    misspelled_en = spell_en.unknown(textEN.split())
    misspelled_it = check_spelling_errors(textIta.lower(),spell_ita)
    num_misspelled = (len(misspelled_en) + len(misspelled_it))/2
    #print (f"\n ing: ",misspelled_en," \nita: ",misspelled_it) #debug

        # Inizializzazione dei valori di qualità e delle features
    quality_ita, features_ita = 0, default_features.copy()
    quality_eng, features_eng = 0, default_features.copy()
```

3. **Calcolo della Diversità Lessicale:** Misura la varietà del lessico utilizzato nel testo, che è un indicatore della sua ricchezza semantica e della sua complessità.

```
# Calcola il punteggio di qualità per il testo italiano
ita_tokens_count = sum(1 for token in docIta if not token.is_punct)
if ita_tokens_count>0:
    quality_ita,features_ita = calculate_quality(docIta, num_misspelled)
# Calcola il punteggio di qualità per il testo inglese
eng_tokens_count = sum(1 for token in docEN if not token.is_punct)
if eng_tokens_count>0:
    quality_eng,features_eng = calculate_quality(docEN, num_misspelled)
# Calcola il totale dei token
total_tokens = ita_tokens_count + eng_tokens_count
# Calcola il punteggio di qualità combinato utilizzando una media pesata
if total_tokens > 0:
    combined_quality = ((ita_tokens_count * quality_ita) + (eng_tokens_count * quality_eng))/total_tokens
else:
    combined_quality = 0  # oppure un altro valore di default
if return_features:
    combined_features = {
    feature: features_ita.get(feature, 0) + features_eng.get(feature, 0)
    for feature in features_ita.keys()
    }
    return combined_quality, combined_features
else:
    return combined_quality
```

4. **Considerazione degli Errori Ortografici:** Integra il controllo ortografico nel punteggio di qualità, dove gli errori riducono il punteggio finale.

```python
def calculate_quality(doc, num_misspelled):

    num_sentences = len(list(doc.sents))
    num_tokens = len(doc)
    num_nouns = len([token for token in doc if token.pos_ == "NOUN"])
    num_verbs = len([token for token in doc if token.pos_ == "VERB"])
    num_adjs = len([token for token in doc if token.pos_ == "ADJ"])

    if num_sentences == 0:
        return 0

    avg_sentence_length = num_tokens / num_sentences
    #lexical_diversity = len(set(token.text.lower() for token in doc)) / num_tokens
    lexical_diversity = len(set(token.text.lower() for token in doc)) / (num_tokens + 1e-7)

    # Qui stiamo dando un peso maggiore ai nomi, verbi e aggettivi.
    quality_score = (
        NOUN_WEIGHT * num_nouns +
        VERB_WEIGHT * num_verbs +
        ADJ_WEIGHT * num_adjs +
        LEXICAL_DIV_WEIGHT * lexical_diversity +
        AVG_SENT_LEN_WEIGHT * avg_sentence_length +
        SPELLING_MISTAKE_WEIGHT * num_misspelled
    ) / (num_tokens + 1e-7)
```

```python
    # Qui stiamo dando un peso maggiore ai nomi, verbi e aggettivi.
    quality_score = (
        NOUN_WEIGHT * num_nouns +
        VERB_WEIGHT * num_verbs +
        ADJ_WEIGHT * num_adjs +
        LEXICAL_DIV_WEIGHT * lexical_diversity +
        AVG_SENT_LEN_WEIGHT * avg_sentence_length +
        SPELLING_MISTAKE_WEIGHT * num_misspelled
    ) / (num_tokens + 1e-7)

    features = {
        'num_sentences': num_sentences,
        'num_tokens': num_tokens,
        'num_nouns': num_nouns,
        'num_verbs': num_verbs,
        'num_adjs': num_adjs,
        'avg_sentence_length': avg_sentence_length,
        'lexical_diversity': lexical_diversity,
        'num_misspelled': num_misspelled
    }

    return quality_score,features
```

Preparazione dei Dati per l'Invio a ChatGPT tramite API
Dopo aver effettuato l'analisi con spaCy, il testo è pronto per essere processato ulteriormente o inviato a un modello di linguaggio come ChatGPT. Le features estratte e il punteggio di qualità possono essere utilizzate per:

- **Selezionare il Testo Migliore:** BestText() determina quale dei tre testi ha il punteggio di qualità più alto e, quindi, dovrebbe essere considerato il più appropriato per la consegna a ChatGPT.
- **Formattazione dei Dati:** I dati analizzati possono essere formattati in un payload JSON che include il testo selezionato insieme ad altre meta informazioni utili che ChatGPT potrebbe utilizzare per contestualizzare le risposte.
- **API Request:** Le informazioni possono essere poi incapsulate in una richiesta HTTP POST inviata all'API di OpenAI, la quale richiederà a ChatGPT di generare una risposta o di eseguire un'ulteriore analisi basata sul testo fornito.

Nella fase di progettazione dell'algoritmo di valutazione della qualità del testo, si è prestata particolare attenzione all'efficienza computazionale, soprattutto in considerazione del volume e della complessità dei testi estratti tramite OCR. Pertanto, si è scelto di implementare un modello di esecuzione concorrente mediante l'utilizzo del multithreading, al fine di ottimizzare i tempi di elaborazione.

```
    # Crea un ThreadPoolExecutor per gestire il multithreading
    with ThreadPoolExecutor(max_workers=3) as executor:
        # Sottomette i task al thread pool
        future1 = executor.submit(execute_check_text_quality, ita_dict1, eng_dict1)
        future2 = executor.submit(execute_check_text_quality, ita_dict2, eng_dict2)
        future3 = executor.submit(execute_check_text_quality, ita_dict3, eng_dict3)

        # Recupera i risultati una volta che sono pronti
        quality1, features1 = future1.result()
        quality2, features2 = future2.result()
        quality3, features3 = future3.result()

        # Stampa i punteggi di qualità
    #print(f"Punteggio qualità Text1: {quality1} con features: {features1}")
    #print(f"Punteggio qualità Text2: {quality2} con features: {features2}")
    #print(f"Punteggio qualità Text3: {quality3} con features: {features3}")

    qualità = [quality1,quality2,quality3]
    massimoVal = (max(qualità))
    best_match_index = qualità.index(massimoVal)
    return best_match_index

def execute_check_text_quality(ita_dict, eng_dict):
    return check_text_quality(ita_dict, eng_dict, True)
```

Il codice sfrutta la classe ThreadPoolExecutor del modulo concurrent.futures per creare un pool di thread. Tale scelta si è rivelata efficace per gestire l'analisi parallela di più segmenti di testo. La decisione di impostare il numero massimo di worker a tre deriva dalla necessità di processare simultaneamente tre diversi estratti testuali. L'obiettivo è quello di massimizzare l'uso delle risorse di calcolo disponibili, distribuendo equamente il carico di lavoro tra i vari core del processore.

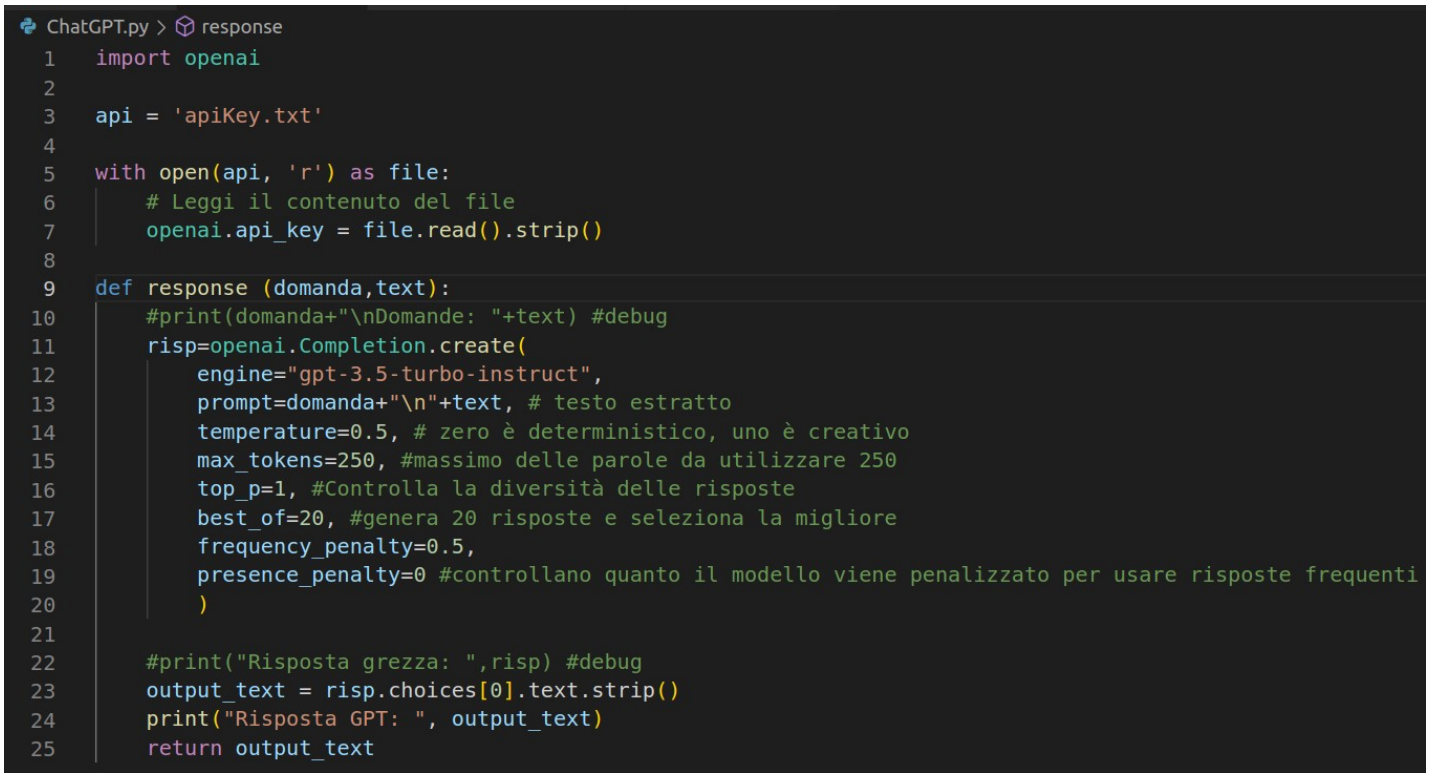
Una volta istanziato l'executor, i task vengono sottomessi attraverso il metodo submit, che accetta come argomento la funzione execute_check_text_quality e i dizionari di testo italiano e inglese corrispondenti ai segmenti estratti dall'OCR. L'uso di Future come oggetto di ritorno per ciascun task sottomesso permette di accedere ai risultati dell'elaborazione non appena disponibili, senza la necessità di un'attesa attiva e quindi inefficiente.

Grazie a questa struttura, ogni segmento di testo viene processato in un thread separato, permettendo un'elaborazione asincrona e indipendente. Ciò garantisce che eventuali complessità o problematiche riscontrate nell'analisi di un segmento non implichino ritardi nell'elaborazione degli altri. Inoltre, questa modalità di esecuzione si dimostra particolarmente vantaggiosa in un contesto di elaborazione di testi OCR, dove il tempo di risposta è un fattore critico e dove l'elaborazione sequenziale standard potrebbe non essere sufficientemente reattiva.

Integrazione con ChatGPT

Nell'ambito del progetto in oggetto, si è proceduto all'integrazione del sistema di elaborazione del testo con l'API di ChatGPT fornita da OpenAI. L'obiettivo principale era quello di sfruttare le capacità avanzate di generazione di testo del modello linguistico per produrre risposte contestualizzate e pertinenti partendo dagli input testuali ottenuti dall'elaborazione OCR.

La sfida principale riscontrata in questa fase è stata l'adattamento degli input per garantire che le risposte generate fossero il più possibile in linea con il contesto specifico delle domande poste. A tale scopo, è stato necessario definire un prompt che includesse la domanda dell'utente e il testo contestuale estratto. L'equilibrio tra questi elementi si è rivelato cruciale: un eccessivo peso dato al testo contestuale poteva risultare in risposte eccessivamente generali, mentre un focus troppo stretto sulla domanda poteva ignorare il contesto essenziale per una risposta informata.



Il modello di risposta utilizzato è il gpt 3.5 turbo, essendo una via di mezzo tra gpt 4 che offre una risposta più lenta ma molto dettagliata e i modelli inferiori che offrono risposte veloci, ma a volte poco precise.

Il parametro temperature è stato impostato a 0.5 per ottenere un compromesso tra determinismo e creatività nelle risposte. Un valore medio consente di mantenere una

certa varietà di output senza sacrificare la coerenza e la pertinenza delle informazioni fornite.

Il max_tokens è stato limitato a 250 per controllare l'estensione delle risposte, rendendole concise e dirette, mentre top_p è stato fissato a 1 per permettere al modello di valutare l'intero spettro di risposte possibili.

Il parametro best_of è stato impostato a 20, funzione essenziale per selezionare la migliore tra una serie di risposte potenzialmente valide, massimizzando così la qualità dell'output. I parametri frequency_penalty e presence_penalty sono stati calibrati per scoraggiare la ripetitività e promuovere la diversificazione del contenuto generato.

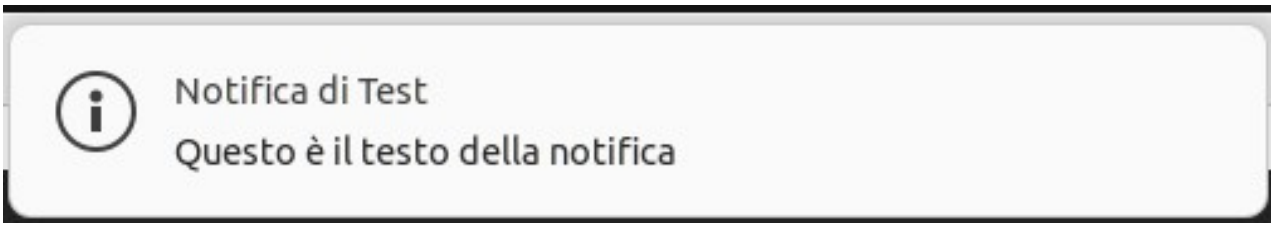
Dopo l'invocazione del metodo Completion.create e la ricezione della risposta, il testo è stato elaborato per rimuovere spazi inutili, garantendo una presentazione pulita e professionale all'utente. La risposta processata, denominata output_text, viene quindi visualizzata in maniera chiara e diretta, consentendo all'utente di percepire immediatamente la pertinenza e l'applicabilità dell'informazione ricevuta.

L'integrazione dell'API di ChatGPT nel sistema si è rivelata un processo iterativo di sperimentazione e affinamento. Le scelte progettuali effettuate hanno permesso di sfruttare in maniera efficace le capacità di GPT-3.5 nel contesto specifico del progetto, migliorando significativamente l'esperienza dell'utente finale attraverso risposte rapide, accurate e contestualmente adeguate.

Interfaccia Utente e Notifiche

Il programma allo stato attuale non dispone interfaccia utente, ma si è utilizzata la libreria ‘plyer’ per inviare la notifica con la risposta fornita dalle API di Chat GPT.

Con questa soluzione l’utente una volta avviato il programma non ha più bisogno di interagire con il terminale.



Notifica.py

```python
from plyer import notification

def Notifica(titolo,messaggio):
    # Mostra la notifica
    notification.notify(
        title=titolo,
        message=messaggio,
        app_icon=None,  # Puoi specificare il percorso ad un'icona .ico (su Windows) o .png (su altri OS)
        timeout=5,  # Tempo dopo il quale la notifica scomparirà in secondi
    )

if __name__ == "__main__":
    # Definisci i parametri della notifica
    titolo = "Notifica di Test"
    messaggio = "Questo è il testo della notifica"
    Notifica(titolo,messaggio)
```

Test e Validazione

I metodi di test sono riportati in appendice e sono commentati con ‘#debug’. Principalmente sono state utilizzate funzioni di stampa per visualizzare che i dati ottenuti dalle funzioni fossero in linea con i risultati attesi.

Risultati e Discussione

Analizzando i risultati ottenuti dall'integrazione del sistema di riconoscimento ottico dei caratteri (OCR) con l'assistente virtuale basato sull'API di ChatGPT. L'accuratezza dell'OCR è stata una componente critica del sistema, e grazie all'implementazione di un processo multithreading che elabora tre immagini filtrate in parallelo, si è ottenuto un rilevante miglioramento della qualità del riconoscimento. La selezione del testo migliore viene effettuata attraverso un algoritmo che valuta e confronta i risultati del riconoscimento, sfruttando metriche come la riconoscibilità dei caratteri e la presenza di errori comuni.

Le prestazioni del sistema sono state monitorate utilizzando la libreria `time` di Python per misurare l'intervallo temporale dall'acquisizione dello screenshot fino alla notifica dell'utente con la risposta generata. I risultati indicano che il tempo medio necessario per completare l'intero ciclo con il mio Hardware è inferiore a due secondi, tempo relativamente breve considerando che le prestazioni del mio ambiente di lavoro sono nella norma. Tale metrica comprende l'acquisizione dell'immagine, l'elaborazione OCR, il riconoscimento della lingua, la valutazione della qualità del testo, la formulazione del prompt per ChatGPT e infine la ricezione e la notifica della risposta all'utente.

In merito alla pertinenza delle risposte fornite da ChatGPT, si è osservata un'elevata corrispondenza tra le query degli utenti e le soluzioni fornite. Il sistema, infatti, non solo genera risposte pertinenti in un breve lasso di tempo ma garantisce anche un'interazione fluida e naturale con l'utente. Questo risultato è stato raggiunto impostando accuratamente i parametri di generazione del testo, come precedentemente descritto.

Nonostante gli esiti positivi, il sistema presenta alcune limitazioni. La dipendenza dalla qualità iniziale delle immagini per l'OCR e la necessità di un equilibrio delicato tra i parametri di generazione di testo sono sfide non trascurabili. Inoltre, la variazione delle condizioni ambientali e le differenze nei layout possono influenzare l'accuratezza dell'OCR, mentre la latenza nella comunicazione con l'API di OpenAI potrebbe variare a seconda della banda disponibile e del carico del server.

Per i lavori futuri, si prospetta l'adozione di un sistema di machine learning per migliorare l'autonomia del processo di selezione del testo OCR più affidabile, possibilmente attraverso un feedback continuo basato sulle interazioni degli utenti. Inoltre, si potrebbe esplorare l'utilizzo di un modello linguistico più avanzato o specificamente addestrato per il contesto di utilizzo, non appena disponibile, per migliorare ulteriormente la pertinenza e la personalizzazione delle risposte di

ChatGPT. In termini di infrastruttura, la considerazione di server più performanti o l'implementazione di un sistema di caching potrebbero essere strategie valide per ridurre la latenza e garantire tempi di risposta ancora più rapidi.

In conclusione, i risultati ottenuti finora sono promettenti e pongono una solida base per lo sviluppo e il miglioramento continuo del sistema. Le prestazioni attuali forniscono un'esperienza utente efficiente e soddisfacente, e gli sviluppi futuri si concentreranno su ottimizzazioni mirate e sull'espansione delle capacità del sistema.

Riferimento bibliografici, Sitografia

Biondo Giuseppe - Sacchi Enrico, “Manuale di elettronica e telecomunicazioni”, HOEPLI, 2015.
Michael Margolis, “Arduino. Progetti e soluzioni”, Tecniche Nuove, 2013.
Massimo Banzi - Michael Shiloh, “Arduino La guida ufficiale”, Tecniche Nuove, 2015.
J. Seitz, T. Arnold, “Python per Hacker Tecniche offensive Black Hat”, Edizioni LSWR, 2021
Stefano Novelli, “HackLOG Volume 1 Anonimato”
Stefano Novelli, “HackLOG Volume 2 Web Hacking”
Paolo Aliverti, “Elettronica per maker”, Edizioni LSWR, 2015.
Paolo Aliverti, “Il manuale di Arduino”, Edizioni LSWR, 2016.
Paolo Aliverti, “Arduino. Trucchi e segreti. 120 idee per risolvere ogni problema”, Edizioni LSWR, 2018.

https://pypi.org/project/plyer/
https://www.arduino.cc/en/Guide/ArduinoEthernetShield
https://www.raspberrypi.org/documentation/accessories/camera.html
https://picamera.readthedocs.io/en/release-1.13/recipes2.html#web-streaming
https://flows.nodered.org/node/node-red-dashboard
https://it.wikipedia.org/wiki/Arduino_(hardware)
https://it.wikipedia.org/wiki/Arduino_IDE
https://pypi.org/project/openai/
https://pypi.org/project/pyspellchecker/
https://pypi.org/project/spacy/
https://pypi.org/project/langdetect/
https://pypi.org/project/pytesseract/
https://pypi.org/project/opencv-python/
https://pypi.org/project/Pillow/
https://pypi.org/project/pyscreenshot/
https://pypi.org/project/pynput/
https://www.python.org/downloads/release/python-380/
https://platform.openai.com/

Appendice

Requirements.txt

```
aiohttp                   3.8.6
aiosignal                 1.3.1
annotated-types           0.6.0
asgiref                   3.7.2
async-timeout             4.0.3
attrs                     23.1.0
backports.zoneinfo        0.2.1
blis                      0.7.11
catalogue                 2.0.10
    certifi                   2023.7.22
    charset-normalizer        3.3.1
    click                     8.1.7
    cloudpathlib              0.16.0
    confection                0.1.3
    curated-tokenizers        0.0.8
    curated-transformers      0.1.1
    cymem                     2.0.8
    Django                    4.2.6
    EasyProcess               1.1
    en-core-web-trf           3.7.2
    entrypoint2               1.1
    filelock                  3.13.1
    frozenlist                1.4.0
    fsspec                    2023.10.0
    idna                      3.4
    image                     1.5.33
    it-core-news-lg           3.7.0
    jeepney                   0.8.0
    Jinja2                    3.1.2
    joblib                    1.3.2
    keyboard                  0.13.5
    langcodes                 3.3.0
    langdetect                1.0.9
    MarkupSafe                2.1.3
    MouseInfo                 0.1.3
    mpmath                    1.3.0
    mss                       9.0.1
    multidict                 6.0.4
    murmurhash                1.0.10
    networkx                  3.1
    numpy                     1.24.4
    nvidia-cublas-cu12        12.1.3.1
    nvidia-cuda-cupti-cu12    12.1.105
```

```
nvidia-cuda-nvrtc-cu12     12.1.105
nvidia-cuda-runtime-cu12   12.1.105
nvidia-cudnn-cu12          8.9.2.26
nvidia-cufft-cu12          11.0.2.54
nvidia-curand-cu12         10.3.2.106
nvidia-cusolver-cu12       11.4.5.107
nvidia-cusparse-cu12       12.1.0.106
nvidia-nccl-cu12           2.18.1
nvidia-nvjitlink-cu12      12.3.52
nvidia-nvtx-cu12           12.1.105
openai                     0.28.1
opencv-python              4.8.1.78
packaging                  23.2
Pillow                     10.1.0
pip                        23.3.1
plyer                      2.1.0
preshed                    3.0.9
PyAutoGUI                  0.9.54
pydantic                   2.4.2
pydantic_core              2.10.1
PyGetWindow                0.0.9
PyMsgBox                   1.0.9
pynput                     1.6.8
pyperclip                  1.8.2
PyRect                     0.2.0
pyscreenshot               3.1
PyScreeze                  0.1.29
pyspellchecker             0.7.2
pytesseract                0.3.10
python-xlib                0.33
python3-xlib               0.15
pytweening                 1.0.7
regex                      2023.10.3
requests                   2.31.0
scikit-learn               1.3.2
scipy                      1.10.1
setuptools                 68.2.2
six                        1.16.0
smart-open                 6.4.0
spacy                      3.7.2
spacy-curated-transformers 0.2.0
spacy-legacy               3.0.12
spacy-loggers              1.0.5
sqlparse                   0.4.4
srsly                      2.4.8
sympy                      1.12
thinc                      8.2.1
threadpoolctl              3.2.0
torch                      2.1.0
tqdm                       4.66.1
```

```
triton                  2.1.0
typer                   0.9.0
typing_extensions       4.8.0
urllib3                 2.0.7
wasabi                  1.1.2
weasel                  0.3.3
wheel                   0.41.3
yarl                    1.9.2
```

main.py

```
import pyscreenshot as ImageGrab
from pynput import keyboard
import time
from OCR import perform_ocr_on_image_from_file,perform_ocr_on_image_from_file_Reduce, write_text_to_file
from Detect_ import BestText
from Notifica import Notifica
from ChatGPT import response

screename = 'screenshot.png'
    filename ='output.txt'

    def inizializzazione():
        domanda = input("Inserisci quello che vuoi chiedere: ")
        OCR_avanzato = False # su True imposta l'ocr avanzato
        return domanda,OCR_avanzato

    def menu():
        print("Premi il tasto 'C' per catturare lo schermo.")
        print("Premi il tasto 'Q' per uscire.")

    def on_press(key):
        try:
            if key.char == 'c':

                print("Cattura in corso...")
                # Cattura lo schermo
                im = ImageGrab.grab()
                # Salva lo screen
                im.save(screename)
                print("Screen Salvato")

                # Punto di partenza: registra il tempo attuale
                start_time = time.time()

                print("Esecuzione OCR su un file immagine in corso...")
                if OCR_avanzato:
                    # Esegui OCR sull'immagine letta dal file
                    text,text1,textCV = perform_ocr_on_image_from_file(screename)
                    # Esegue valutazione miglior testo
                    IndiceBT = BestText(text,text1,textCV)
                else:
                    text =
```

```
    perform_ocr_on_image_from_file_Reduce(screename)
43.                     IndiceBT = None
44.
45.                 print("Sto inviando a chatGPT...")
46.                 if IndiceBT is not None:
47.                     texts = [text, text1, textCV]
48.                     print(f"Testo con indice {IndiceBT}:
    {texts[IndiceBT]}")
49.                     try:
50.                         risposta = response(domanda,texts[IndiceBT])
51.                     except Exception as err:
52.                         print(f"Errore con ChatGPT: {err}")
53.                 else:
54.                     try:
55.                         risposta = response(domanda,text)
56.                     except Exception as err:
57.                         print(f"Errore con ChatGPT: {err}")
58.
59.                 Notifica("Risposta GPT", risposta)
60.                 #write_text_to_file(risposta,filename)
61.                 print("Operazione Completata!")
62.
63.                 # Punto di arrivo: registra il tempo attuale
64.                 end_time = time.time()
65.                 # Calcolo del tempo trascorso
66.                 elapsed_time = end_time - start_time
67.                 print(f"L'elaborazione ha impiegato {elapsed_time}
    secondi.")
68.
69.                 menu()
70.
71.             elif key.char == 'q':
72.                 print("Uscita in corso...")
73.                 exit(0)
74.         except AttributeError:
75.             pass  # Gestisce i casi in cui il tasto non ha un
    carattere associato
76.
77.
78.     if __name__ == "__main__":
79.
80.         domanda, OCR_avanzato = inizializzazione()
81.
82.         menu()
83.
84.         with keyboard.Listener(on_press=on_press) as listener:
85.           listener.join()
```

OCR.py

```
1. import pytesseract
2. from PIL import Image
3. import cv2
4.
5. def perform_ocr_on_image_from_file(screen):
6.     try:
7.         with Image.open(screen) as image:   # Apre l'immagine dal file
8.             # Converte l'immagine in scala di grigi
9.             imageBN = image.convert("L")
10.
11.                 # Carica l'immagine utilizzando OpenCV
```

```
                    img = cv2.imread(screen)
                    # Converti l'immagine in scala di grigi
                    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    # Applica una sogliatura
                    _, thresh_img = cv2.threshold(gray_img, 90, 180, cv2.THRESH_BINARY)

                    # pytesseract per effettuare l'OCR sulle immagini pre-processate
                    textCV = pytesseract.image_to_string(thresh_img, lang='eng',config='--psm 6 --oem 3')
                    text = pytesseract.image_to_string(image, lang='eng',config='--psm 6 --oem 3')
                    text1 = pytesseract.image_to_string(imageBN, lang='eng',config='--psm 6 --oem 3')

                return text, text1, textCV
            except Exception as e:
                print(f"Errore: {e}")
                return None


        def perform_ocr_on_image_from_file_Reduce(screen):
            try:
                with Image.open(screen) as image:  # Apre l'immagine dal file

                    # Converte l'immagine in scala di grigi
                    imageBN = image.convert("L")

                    # pytesseract per effettuare l'OCR sull' immagine pre-processata
                    text = pytesseract.image_to_string(imageBN, lang='eng',config='--psm 6 --oem 3')

                #print(f"Testo estratto: {text}")  # Debug
                return text
            except Exception as e:
                print(f"Errore nell'apertura del file immagine: {e}")
                return None


        def write_text_to_file(text, filename):
            try:
                with open(filename, 'w') as f:
                    f.write(text)
                    f.write('\n')
                    f.flush()
                print(f"Testo scritto con successo nel file {filename}.")
            except Exception as e:
                print(f"Errore nella scrittura del file: {e}")

```

Detect_.py

```
from LangDetect import LangDetect
from Calcolo_Qualità import check_text_quality
from concurrent.futures import ThreadPoolExecutor

def BestText(text1,text2,text3):

```

```
    ita_dict1,eng_dict1 = LangDetect(text1)
    ita_dict2,eng_dict2 = LangDetect(text2)
    ita_dict3,eng_dict3 = LangDetect(text3)

          # Crea un ThreadPoolExecutor per gestire il multithreading
          with ThreadPoolExecutor(max_workers=3) as executor:
              # Sottomette i task al thread pool
              future1 = executor.submit(execute_check_text_quality,
   ita_dict1, eng_dict1)
              future2 = executor.submit(execute_check_text_quality,
   ita_dict2, eng_dict2)
              future3 = executor.submit(execute_check_text_quality,
   ita_dict3, eng_dict3)

              # Recupera i risultati una volta che sono pronti
              quality1, features1 = future1.result()
              quality2, features2 = future2.result()
              quality3, features3 = future3.result()

              # Stampa i punteggi di qualità
          #print(f"Punteggio qualità Text1: {quality1} con features:
   {features1}")
          #print(f"Punteggio qualità Text2: {quality2} con features:
   {features2}")
          #print(f"Punteggio qualità Text3: {quality3} con features:
   {features3}")

          qualità = [quality1,quality2,quality3]
          massimoVal = (max(qualità))
          best_match_index = qualità.index(massimoVal)
          return best_match_index

      def execute_check_text_quality(ita_dict, eng_dict):
          return check_text_quality(ita_dict, eng_dict, True)


      if __name__ == "__main__":
          text1 = "Esempio di testo uno, the quality it isn't water, tre
   tigri contro tre tigri. the moa+n is biger"
          text2 = "Esempio di testo duea, the quality it isnt water, tre
   tigri contro tre tigri. the moon is bipg"
          text3 = "Esempio di testo tare, the quality it isnt water, tre
   tigri contro tre tigri. the moon is big"
          Index_best_text=BestText(text1,text2,text3)
          if Index_best_text is not None:
                      texts = [text1, text2, text3]
                      print(f"Testo con indice {Index_best_text}:
   {texts[Index_best_text]}")
          else:
              print("Errore indice funzione BestText ")

```

Calcolo_Qualità.py

```
from spellchecker import SpellChecker
from SpellCheck import check_spelling_errors, load_italian_dictionary,
   initialize_spellchecker
import spacy
import logging

# Inizializzazione del logger
```

```
logging.basicConfig(level=logging.INFO)

# Caricamento modelli spacy
      Inglese = spacy.load("en_core_web_trf") # per la lingua inglese
      Ita = spacy.load("it_core_news_lg") # per la lingua italiana

      # Carica spellchecker per la lingua Inglese
      spell_en = SpellChecker(language='en')
      # Carica il dizionario italiano da un file (devi fornire il percorso al tuo file)
      file_path = '660000_parole_italiane.txt'
      diz_ita = load_italian_dictionary(file_path)
      spell_ita = initialize_spellchecker(diz_ita)

      # Costanti
      NOUN_WEIGHT = 0.45
      VERB_WEIGHT = 0.5
      ADJ_WEIGHT = 0.4
      LEXICAL_DIV_WEIGHT = 0.0 # non utilizzato
      AVG_SENT_LEN_WEIGHT = 0.0 # non utilizzato
      SPELLING_MISTAKE_WEIGHT = -0.5  # Peso negativo per gli errori ortografici

      def check_text_quality(textIta,textEN,return_features=False):
          # Dizionario vuoto da usare come valore di default
          default_features = {}
          docEN = Inglese(textEN)
          docIta = Ita(textIta)

          # Calcolo degli errori ortografici
          misspelled_en = spell_en.unknown(textEN.split())
          misspelled_it = check_spelling_errors(textIta.lower(),spell_ita)
          num_misspelled = (len(misspelled_en) + len(misspelled_it))/2
          #print (f"\n ing: ",misspelled_en," \nita: ",misspelled_it) #debug

              # Inizializzazione dei valori di qualità e delle features
          quality_ita, features_ita = 0, default_features.copy()
          quality_eng, features_eng = 0, default_features.copy()

          try:
              # Calcola il punteggio di qualità per il testo italiano
              ita_tokens_count = sum(1 for token in docIta if not token.is_punct)
              if ita_tokens_count>0:
                  quality_ita,features_ita = calculate_quality(docIta, num_misspelled)
              # Calcola il punteggio di qualità per il testo inglese
              eng_tokens_count = sum(1 for token in docEN if not token.is_punct)
              if eng_tokens_count>0:
                  quality_eng,features_eng = calculate_quality(docEN, num_misspelled)
              # Calcola il totale dei token
              total_tokens = ita_tokens_count + eng_tokens_count
              # Calcola il punteggio di qualità combinato utilizzando una media pesata
              if total_tokens > 0:
                  combined_quality = ((ita_tokens_count * quality_ita) + (eng_tokens_count * quality_eng)) / total_tokens
              else:
```

```
                    combined_quality = 0  # oppure un altro valore di default
                if return_features:
                    combined_features = {
                    feature: features_ita.get(feature, 0) + features_eng.get(feature, 0)
                    for feature in features_ita.keys()
                    }
                    return combined_quality, combined_features
                else:
                    return combined_quality
            except Exception as e:
                logging.error(f"Errore nel calcolo della qualità del testo: {e}")
                return 0, default_features # oppure un altro valore di default


        def calculate_quality(doc, num_misspelled):

            num_sentences = len(list(doc.sents))
            num_tokens = len(doc)
            num_nouns = len([token for token in doc if token.pos_ == "NOUN"])
            num_verbs = len([token for token in doc if token.pos_ == "VERB"])
            num_adjs = len([token for token in doc if token.pos_ == "ADJ"])


            if num_sentences == 0:
                return 0

            avg_sentence_length = num_tokens / num_sentences
            #lexical_diversity = len(set(token.text.lower() for token in doc)) / num_tokens
            lexical_diversity = len(set(token.text.lower() for token in doc)) / (num_tokens + 1e-7)


            # Qui stiamo dando un peso maggiore ai nomi, verbi e aggettivi.
            # Inoltre, consideriamo favorevolmente i testi con una maggiore diversità lessicale e una lunghezza media della frase ragionevole.
            quality_score = (
                NOUN_WEIGHT * num_nouns +
                VERB_WEIGHT * num_verbs +
                ADJ_WEIGHT * num_adjs +
                LEXICAL_DIV_WEIGHT * lexical_diversity +
                AVG_SENT_LEN_WEIGHT * avg_sentence_length +
                SPELLING_MISTAKE_WEIGHT * num_misspelled
            ) / (num_tokens + 1e-7)

            features = {
                'num_sentences': num_sentences,
                'num_tokens': num_tokens,
                'num_nouns': num_nouns,
                'num_verbs': num_verbs,
                'num_adjs': num_adjs,
                'avg_sentence_length': avg_sentence_length,
                'lexical_diversity': lexical_diversity,
```

```
            'num_misspelled': num_misspelled
        }

        return quality_score,features
```

LangDetect.py

```
from langdetect import detect, LangDetectException
import re

def remove_special_characters(text):
    # La funzione re.sub sostituisce tutti i caratteri che non sono lettere o numeri con una stringa vuota.
    # Il pattern [^a-zA-Z0-9\s] indica "qualsiasi carattere che NON è una lettera maiuscola o minuscola, un numero o uno spazio bianco".
    return re.sub(r"[^a-zA-Z0-9\sàèéìòù'\"`]", '', text)

def LangDetect(segments):
      # Dizionari per immagazzinare le parole di ciascuna lingua
          ita_list = []
          eng_list = []
          # Utilizzare re.split per suddividere la stringa con più delimitatori
          words = [word for word in re.split('[.,|;]', segments) if word]
      # Iterazione attraverso ciascuna parola nel testo
          for word in words:
              cleaned_word = None
              if(word):
                  cleaned_word = remove_special_characters(word)
                  #print(f"Parola pulita: {cleaned_word}")  # Debug
                  try:
              # Rilevamento della lingua della parola
                        lang = detect(cleaned_word)
                        #print(f"Lingua rilevata per {cleaned_word}: {lang}")  # Debug
                        if lang == 'it':
                            ita_list.append(word)
                        elif lang == 'en':
                            eng_list.append(word)
                        else:
                            eng_list.append(word)
                    except LangDetectException:
                        if cleaned_word and not cleaned_word.isnumeric():
                            print(f"Impossibile determinare la lingua per la parola: {word}")

          ita_dict = " ".join(ita_list)
          eng_dict = " ".join(eng_list)
          #print(f"Dizionario ITA: {ita_dict}")  # Debug
          #print(f"Dizionario ENG: {eng_dict}")  # Debug
          return ita_dict,eng_dict
```

SpellCheck.py

```
from spellchecker import SpellChecker

# Carica la lista di parole italiane da un file
```

```
def load_italian_dictionary(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        italian_words = file.read().splitlines()
    return italian_words


    # Inizializza il controllo ortografico con il dizionario italiano
    def initialize_spellchecker(italian_words):
        spellchecker = SpellChecker(language=None, case_sensitive=True)
        spellchecker.word_frequency.load_words(italian_words)
        return spellchecker

    # Verifica la presenza di errori ortografici e calcola il punteggio
    def check_spelling_errors(text, spellchecker):
        # Separare il testo in parole
        words = text.split()
        # Trova le parole che sono errate
        misspelled = spellchecker.unknown(words)
        # Ritorna il numero di errori trovati
        return misspelled
```

ChatGPT.py

```
import openai

api = 'apiKey.txt'

with open(api, 'r') as file:
    # Leggi il contenuto del file
    openai.api_key = file.read().strip()

def response (domanda,text):
        #print(domanda+"\nDomande: "+text) #debug
        risp=openai.Completion.create(
            engine="gpt-3.5-turbo-instruct",
            prompt=domanda+"\n"+text, # testo estratto
            temperature=0.5, # zero è deterministico, uno è creativo
            max_tokens=250, #massimo delle parole da utilizzare 250
            top_p=1, #Controlla la diversità delle risposte
            best_of=20, #genera 20 risposte e seleziona la migliore
            frequency_penalty=0.5,
            presence_penalty=0 #controllano quanto il modello viene penalizzato per usare risposte frequenti
            )

        #print("Risposta grezza: ",risp) #debug
        output_text = risp.choices[0].text.strip()
        print("Risposta GPT: ", output_text)
        return output_text
```

Notifica.py

```
from plyer import notification

def Notifica(titolo,messaggio):
    # Mostra la notifica
```

```
    notification.notify(
        title=titolo,
        message=messaggio,
        app_icon=None,  # Puoi specificare il percorso ad un'icona .ico (su Windows) o .png (su altri OS)
        timeout=5,  # Tempo dopo il quale la notifica scomparirà in secondi
         )

    if __name__ == "__main__":
        # Definisci i parametri della notifica
        titolo = "Notifica di Test"
        messaggio = "Questo è il testo della notifica"
        Notifica(titolo,messaggio)
```