

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica



Tesi di Laurea Magistrale

Sviluppo di un'estensione VS Code AI-based per migliorare lo sviluppo web accessibile

Relatori

Prof. Riccardo COPPOLA

Dott. Tommaso FULCINI

Candidato

Elisa CALÌ

Aprile 2025

Sommario

La tesi affronta il tema dell'accessibilità digitale e l'impiego di Large Language Model (LLM) per supportare gli sviluppatori nella creazione di prodotti digitali accessibili. Dopo un'introduzione al contesto normativo e tecnologico, il lavoro approfondisce il concetto di accessibilità web, le linee guida WCAG e le tecniche di testing, esaminando anche ricerche che utilizzano gli LLM per identificare e correggere errori di accessibilità.

Successivamente, vengono analizzate tre possibili architetture per sviluppare un'estensione AI-based per Visual Studio Code, insieme alle tecniche di prompt engineering per ottimizzare il comportamento degli LLM. Vengono proposti ed implementati due use case, FixWithAI e CheckAndFixWithAI, aventi lo scopo di correggere e verificare errori di accessibilità presenti nel codice sorgente di un prodotto digitale. Per ognuno di essi viene riportata una descrizione dettagliata degli aspetti tecnici.

Infine, vengono illustrati i processi di validazione dell'estensione, analizzando i risultati ottenuti nei test per i due casi d'uso. Si evidenziano promettenti risultati per FixWithAI, mentre CheckAndFixWithAI richiede ulteriori sviluppi. La tesi si conclude con una discussione sui limiti dell'estensione e prospettive per futuri miglioramenti.

È inoltre presentato un articolo scientifico, frutto del lavoro di tesi, accettato per la pubblicazione alla ICSE 2025 - 47th International Conference on Software Engineering.

Ringraziamenti

Ringrazio il Prof. Riccardo Coppola e il Dott. Tommaso Fulcini per il supporto datomi in questa fase conclusiva del mio percorso accademico e l'azienda, di cui non posso citare il nome, per avermi fornito una repository su cui validare il lavoro di tesi.

Indice

Elenco delle tabelle	7
Elenco delle figure	8
Glossario	12
Acronimi	14
1 Introduzione	15
2 Background	17
2.1 LLM	17
2.1.1 Reti neurali	17
2.1.2 Addestramento ed ottimizzazione degli LLM	18
2.1.3 Vantaggi e limitazioni	19
2.2 Accessibilità	20
2.2.1 WCAG 2.2	21
2.2.2 Legge Stanca	22
2.2.3 Accessibilità, Usabilità ed Inclusività	22
2.2.4 Testare l'accessibilità	22
2.3 Lavori correlati	23
3 Design	25
3.1 Analisi delle possibili architetture	25
3.1.1 Soluzione 1 - Estensione del linter ESLint	25
3.1.2 Soluzione 2 - Realizzazione di un'estensione per VS Code	26
3.1.3 Soluzione 3 - Creazione di un'estensione per VS Code e utilizzo di ESLint	27
3.2 Contesto applicativo	27
3.3 Design Framework	28
3.3.1 Prompt engineering	28
4 Implementazione	37
4.1 Architettura	37
4.2 Use Case e Requisiti	38
4.2.1 Use Case 1 - FixWithAI	38
4.2.2 Use Case 2 - CheckAndFixWithAI	41
4.3 Prototipo	45
4.3.1 Implementazione use case CheckAndFixWithAI	45

4.3.2	Implementazione use case FixWithAI	49
5	Validazione	52
5.1	Criteri di classificazione dei risultati	52
5.1.1	Use case FixWithAI	53
5.1.2	Use case CheckAndFixWithAI	55
5.2	Risultati	60
5.2.1	FixWithAI	60
5.2.2	CheckAndFixWithAI	64
5.3	Analisi	65
5.3.1	FixWithAI	67
5.3.2	CheckAndFixWithAI	68
6	Conclusioni	73
6.1	Limiti	73
6.2	Future Works	74
A	Prompt utilizzati	75
A.1	Use case FixWithAI	75
A.1.1	System prompt	75
A.1.2	User prompt - correzione degli errori segnalati da ESLint	75
A.2	Use case CheckAndFixWithAI	76
A.2.1	Primo prototipo	76
A.2.2	Secondo prototipo	79
	Bibliografia	104

Elenco delle tabelle

5.1	Tabella rappresentante i criteri di successo suddividi per livelli di conformità relativi allo use case FixWithAI	56
5.1	Tabella rappresentante i criteri di successo suddividi per livelli di conformità relativi allo use case FixWithAI	57
5.2	Tabella rappresentante i criteri di successo suddividi per livelli di conformità relativi allo use case CheckAndFixWithAI	61
5.2	Tabella rappresentante i criteri di successo suddividi per livelli di conformità relativi allo use case CheckAndFixWithAI	62
5.2	Tabella rappresentante i criteri di successo suddividi per livelli di conformità relativi allo use case CheckAndFixWithAI	63
5.2	Tabella rappresentante i criteri di successo suddividi per livelli di conformità relativi allo use case CheckAndFixWithAI	64
5.3	Tabella riportante le percentuali di valutazione per ogni criterio dello use case FixWithAI	67
5.4	Tabella riportante le percentuali di valutazione per ogni criterio dello use case CheckAndFixWithAI	69
5.4	Tabella riportante le percentuali di valutazione per ogni criterio dello use case CheckAndFixWithAI	70

Elenco delle figure

2.1	Struttura di una rete neurale	18
2.2	Esempi di disabilità permanenti, temporanee e situazionali per i sensi del tatto, dell'udito, della vista e della parola	20
3.1	Esempio di prompt con l'utilizzo di un'istruzione, dei dati in input e dell'indicatore dell'output	29
3.2	Esempio di prompt che definisce due step per rispondere agli input dell'utente. Nella prima fase il modello dovrà riassumere il testo fornito dall'utente, inserendo il prefisso "Riassunto:", mentre nella seconda fase dovrà tradurre il riassunto in spagnolo e restituire la traduzione antepoendo la frase "Traduzione:"	30
3.3	Esempio di prompt che definisce il template per l'output da ricevere	30
3.4	Esempio di prompt dove viene indicato al modello di non chiedere informazioni personali o interessi, utilizzando frasi negative invece che la tecnica del rinforzo positivo	30
3.5	Esempio di risposta al prompt di figura 3.4 dove l'LLM chiede le informazioni personali che gli era stato detto di non chiedere	31
3.6	Esempio di prompt con la tecnica del rinforzo positivo ; al modello viene detto cosa fare. Nel caso in cui non debba fare qualcosa, questo viene veicolato con la forma <i>astenersi dal chiedere, evitare di chiedere</i> invece che con una negazione	31
3.7	Esempio di risposta al prompt di figura 3.6. L'utente chiede di ricevere dei suggerimenti in base ai suoi gusti personali. Poiché il modello non può chiedere queste informazioni, risponde che non può fornire risposte in linea con la domanda fatta	31
3.8	Esempio di prompt che inserisce dei delimitatori per separare le istruzioni dal contesto	32
3.9	Esempio di prompt che assegna un ruolo al modello. In questo caso il modello sa di essere un esperto di comunicazione e marketing con diversi anni di esperienza in alcuni campi del suo lavoro	32
3.10	Esempio di prompt che fornisce degli articoli dove il modello potrà trovare le risposte	32
3.11	Esempio di priming prompt	33
3.12	Esempio di prompt Zero-Shot . L'LLM è in grado di classificare un testo come <i>neutro, negativo o positivo</i> senza che gli vengano forniti degli esempi	33
3.13	Esempio di prompt e output della tecnica prompt Few Shot	33
3.14	Esempio della tecnica prompt Chain-of-Thought (CoT) . Vengono forniti degli esempi di domande e risposte, così che il modello impara il tipo di ragionamento che deve porre in essere per poter formulare l'output	34

3.15	Esempio di errore di risposta di un LLM quando gli viene sottoposto un semplice problema matematico e non viene usata la tecnica del Chain-of-Thought	34
3.16	Esempio di prompt che utilizza la tecnica dello Zero-Shot Chain-of-Thought attraverso la quale il modello riesce a formulare la risposta corretta	35
3.17	Esempio di prompt e output dove non viene applicata la tecnica di self-consistency	35
3.18	Esempio della tecnica di self-consistency	36
4.1	Architettura dell'estensione	38
4.2	Diagramma degli use case	39
4.3	BPMN dello use case FixWithAI	40
4.4	BPMN dello use case CheckAndFixWithAI - primo prototipo	42
4.5	BPMN dello use case CheckAndFixWithAI - secondo prototipo	43
4.6	Sequenza delle interrogazioni all'LLM per i due use case	44
4.7	File package.json dell'applicazione dove viene definito il comando <i>checkAndFixWithAi</i>	45
4.8	Comando presente nel menù contestuale solo se lo sviluppatore si trova su un file Javascript o TypeScript e ha evidenziato un blocco di codice	46
4.9	Comando presente nel command palette solo se lo sviluppatore si trova su un file Javascript o TypeScript e ha evidenziato un blocco di codice	46
4.10	Interrogazione dell'LLM per lo use case CheckAndFixWithAI - primo prototipo. Vengono effettuate due chiamate in cascata, la prima per identificare la presenza di errori di accessibilità, la seconda per correggerli. In ultimo viene generato un file txt contenente il codice analizzato e le risposte delle due interrogazioni effettuate all'LLM	47
4.11	Interrogazione dell'LLM per lo use case CheckAndFixWithAI - secondo prototipo. Vengono effettuate tre chiamate in cascata, la prima per identificare la presenza di errori di accessibilità. Essa restituisce una lista di errori identificati; la seconda converte la lista in un JSON; la terza corregge gli errori ricevuti in input. Successivamente l'LLM viene rimosso dalla memoria. In ultimo viene generato un file txt contenente il codice analizzato e le risposte delle tre interrogazioni effettuate all'LLM	48
4.12	Codice per creare il comando <i>fixAccessibilityIssue</i>	49
4.13	Menù di correzione rapida generato da eslint che presenta la voce "Get fix suggestion from AI"	50
4.14	Interrogazione dell'LLM per lo use case FixWithAI. La funzione riceve le segnalazioni fatte da ESLint e il codice che genera l'errore e li passa, sotto forma di JSON, all'LLM che restituisce un array con le dovute correzioni	50
4.15	Funzione che formatta la risposta dell'LLM che verrà mostrata allo sviluppatore all'interno del file dove si ha il codice contenente la segnalazione. La risposta contiene la descrizione della problematica, la descrizione delle possibili risoluzioni, nonché il codice corretto	51
5.1	Processo decisionale per la classificazione dei test. Se almeno un criterio è "Non corretto" o "Non valutabile" il test viene classificato come non corretto. Se almeno l'80% dei criteri è "Corretto" o "Parzialmente corretto", il test viene considerato corretto. Se meno dell'80% dei criteri è corretto o parzialmente corretto, il test è classificato come parzialmente corretto	53
5.2	Grafico risultati preliminari FixWithAI su 25 test. 64% dei test è "Corretto", il 20% è "Parzialmente corretto", il 16% è "Non corretto"	61
5.3	Grafico risultati definitivi FixWithAI su 120 test. 68,33% dei test è "Corretto", il 1,67% è "Parzialmente corretto", il 30% è "Non corretto"	64

5.4	Grafico risultati preliminari primo prototipo CheckAndFixWithAI su 25 test. 0% dei test è "Corretto", il 8% è "Parzialmente corretto", il 92% è "Non corretto" . . .	65
5.5	Grafico risultati preliminari secondo prototipo CheckAndFixWithAI su 25 test. 24% dei test è "Corretto", il 16% è "Parzialmente corretto", il 60% è "Non corretto"	66
5.6	Grafico risultati definitivi secondo prototipo CheckAndFixWithAI su 120 test. 27,5% dei test è "Corretto", il 11,67% è "Parzialmente corretto", il 60,83% è "Non corretto"	66
5.7	Grafico distribuzione numero criteri errati per test - FixWithAI	68
5.8	Grafico risultati definitivi secondo prototipo CheckAndFixWithAI su 120 test senza considerare il criterio "Integrità codice". 23.33% dei test è "Corretto", il 28.33% è "Parzialmente corretto", il 48.33% è "Non corretto"	71
5.9	Grafico distribuzione numero criteri errati per test - CheckAndFixWithAI	71

Glossario

Apprendimento auto-supervisionato Vedi apprendimento non supervisionato

Apprendimento non supervisionato (anche **auto-supervisionato**) I dati di addestramento che vengono forniti al modello non sono stati precedentemente etichettati. E' il modello stesso a comprendere le relazioni tra i dati ed il loro significato

ARIA (Accessible Rich Internet Applications) Insieme di ruoli e attributi da inserire nel codice HTML che definiscono modi per cui le pagine e le applicazioni web risultino più accessibili a persone con disabilità

Blind prompting Approccio alla creazione di prompt basato sul try and error, senza l'utilizzo di tecniche di prompt engineering

Code analysis Processo di analisi del codice sorgente, byte code o binary code per verificarne la qualità, l'affidabilità e la sicurezza

Code Llama LLM basato su Llama 2 che permette di generare e analizzare il codice

Code obfuscation Processo per il quale viene generato un programma funzionalmente equivalente a quello di partenza ma più complesso da comprendere, al fine di proteggere la proprietà intellettuale del software o nascondere comportamenti dannosi

CodeGemma Collezione di modelli leggeri e potenti rilasciati da Google e che possono eseguire diversi task legati al codice, tra cui completare del codice, generarlo, seguire istruzioni, comprendere il linguaggio naturale

Context window Numero massimo di token che un modello è in grado di analizzare durante l'elaborazione di un input

Dati esterni Dati forniti all'LLM che non appartengono al set di training iniziale

ESLint Linter per codice Javascript e JSX, analizza staticamente il codice e segnala eventuali problematiche

Fine-tuning Tecnica di apprendimento supervisionato per specializzare un modello LLM su task specifici

Javascript Linguaggio di programmazione usato sia per sviluppare pagine web che da ambienti non-browser come Node.js

JSX Estensione sintattica per Javascript che permette di scrivere del codice simile all'HTML all'interno di file Javascript

Lint Strumento di analisi statica del codice che analizza il codice sorgente in cerca di errori

Llama 2 Collezione di LLM open source rilasciati da Meta

Llama 3 Collezione di LLM open source rilasciati da Meta

Narrator Lettore di schermo per dispositivi Windows

NVDA NonVisual Desktop Access - screen reader gratuito e open-source per Microsoft Windows

Prompt Nel contesto dell'intelligenza generativa, input/istruzioni che l'utente fornisce all'LLM. Definisce l'informazione che si vuole avere, il formato e lo stile

Prompt engineering Disciplina che studia tecniche per ottimizzare prompt e utilizzare al meglio le capacità degli LLM

PromptMixer Tool open source che permette di confrontare diversi llm e la loro risposta alla modifica del prompt in input. Permette inoltre di creare un versioning dei prompt così da poter tornare indietro agilmente

React Libreria Javascript per lo sviluppo di applicazioni web

React Native Libreria Javascript per lo sviluppo di applicazioni native

Rete neurale Modello di machine learning ispirato alle reti neurali dei cervelli animali

Rinforzo positivo Tecnica di prompt engineering attraverso cui viene indicato all'LLM cosa può effettuare. Se si indicano delle limitazioni nell'attività del modello, queste vengono veicolate tramite espressione del tipo *evitare di fare qualcosa*

Sviluppatore Front-End Sviluppatore che si occupa della realizzazione della parte visibile ed interattiva di una pagina web

Transformer Architettura di rete neurale basata sull'attenzione che è in grado di modificare un input ricevuto

TypeScript Linguaggio di programmazione sviluppato da Microsoft. Estende Javascript inserendo i tipi al linguaggio

Visual Studio Code (anche VS Code) Editor per sviluppo di codice

VoiceOver Lettore di schermo per dispositivi Apple

Vue.js Framework Javascript per lo sviluppo di pagine web

Acronimi

ADA

Americans with Disabilities Act

AGID

Agenzia per l'Italia digitale

APG

ARIA Authoring Practices Guide

BPMN

Business Process Model and Notation

CMS

Content Management System

LLM

Large Language Model

OMS

Organizzazione Mondiale della Sanità

RAG

Retrieval-Augmented Generation

W3C

World Wide Web Consortium

WAI

Web Accessibility Initiative

WCAG

Web Content Accessibility Guidelines

Capitolo 1

Introduzione

Negli ultimi anni, a causa anche alla pandemia, sempre più servizi vengono fruiti online da un numero crescente di persone. L'estensione della Legge Stanca anche ai soggetti privati[1] attraverso il decreto-legge n. 76/2020, ha dato un impulso significativo verso la creazione di siti web e applicazioni mobile accessibili.

L'OMS, nella classificazione del funzionamento, della disabilità e della salute[24], pone l'accento sul gap tra le caratteristiche individuali e il modo in cui l'individuo interagisce con la società. Per contro, per accessibilità dei sistemi informatici si intende la capacità di questi di erogare servizi e rendere fruibili le informazioni anche a coloro che necessitano di tecnologie assistive per poter interagire con i contenuti digitali.

Sebbene vi sia una crescente consapevolezza da parte delle aziende sull'importanza di creare contenuti digitali accessibili, il processo di realizzazione e verifica di siti e applicazioni accessibili rimane complesso e dispendioso anche in termini di tempo. Gli attuali test automatici riescono a coprire solo una parte di tutti i test necessari, rendendo quindi fondamentale un largo impiego di test manuali che possono essere effettuati solo post sviluppo del sito/applicazione mobile. Ciò comporta, come si evince dal monitoraggio dell'AGID[2, 3], un numero elevato di errori di accessibilità presenti sui siti online. Diventa, quindi, essenziale anticipare il testing del prodotto digitale già durante la fase di sviluppo, così da ridurre il numero di errori di accessibilità sin dalle prime fasi.

Recentemente sempre più modelli LLM vengono utilizzati in diversi ambiti, anche nella generazione di codice. Con le crescenti potenzialità di comprensione e di generazione, gli LLM possono supportare lo sviluppatore durante il suo lavoro.

Lo scopo di questo elaborato di tesi è la valutazione degli LLM nella generazione e valutazione di codice accessibile e la realizzazione di un'estensione per VS Code AI-based, con l'obiettivo di assistere lo sviluppatore durante tutte le fasi di creazione del prodotto digitale.

Il resto della tesi sarà così strutturata:

- all'interno del capitolo 2 verrà introdotta la definizione di LLM, esplorando alcune tecniche di addestramento ed ottimizzazione dei modelli. Successivamente si esplorerà il concetto di accessibilità, con particolare attenzione all'accessibilità web. Verranno analizzate le linee guida WCAG e le metodologie utilizzate per testare l'accessibilità di un prodotto digitale. In ultimo verranno citati dei lavori di ricerca che utilizzano gli LLM per identificare e correggere errori di accessibilità;
- nel terzo capitolo verranno analizzate tre possibili architetture per realizzare l'estensione per VS Code, evidenziandone vantaggi e svantaggi. Verrà descritto il contesto applicativo in cui

l'applicazione dovrà operare e saranno illustrate numerose tecniche di prompt engineering per ottimizzare il comportamento di un LLM;

- nel quarto capitolo verranno descritti i due use case su cui si baserà l'estensione: FixWithAI e CheckAndFixWithAI. Il primo si pone come obiettivo l'utilizzo dell'AI solo per la correzione degli errori di accessibilità, demandando a un linter l'identificazione degli errori di accessibilità. Il secondo use case, invece, utilizzerà un LLM sia per l'identificazione degli errori di accessibilità che per la loro correzione. Verranno, in fine, riportati i diversi aspetti implementativi suddivisi per use case;
- all'interno del capitolo 5 saranno illustrati i processi di classificazione e di validazione dell'estensione; per ogni use case verranno analizzati i diversi criteri di successo suddivisi nelle categorie "non corretto", "parzialmente corretto" e "corretto". Saranno presentati i risultati dei test condotti per ogni use case e si analizzeranno i diversi risultati. Essi evidenziano risultati promettenti per lo use case FixWithAI, mentre lo use case CheckAndFixWithAI richiede ulteriori miglioramenti, non inclusi all'interno di questa trattazione. Tuttavia, anche quest'ultimo mostra segnali incoraggianti;
- nell'ultimo capitolo saranno discussi i limiti dell'estensione sviluppata e dell'analisi condotta, riportando possibili sviluppi futuri.

In ultimo, si cita che il lavoro svolto ha portato alla redazione di un paper scientifico, intitolato *A Prototype VS Code Extension to Improve Web Accessible Development*, accettato per la pubblicazione all'interno di *The 2nd IDE Workshop*¹ che si terrà il 3 Maggio ad Ottawa nell'ambito degli *ICSE 2025 - 47th International Conference on Software Engineering*².

¹<https://ide-workshop.github.io/>.

²<https://conf.researchr.org/track/icse-2025/icse-2025-workshops>.

Capitolo 2

Background

All'interno di questo capitolo si daranno le nozioni base sugli LLM (Large Language Model) e sull'accessibilità. Si esaminerà l'origine, lo sviluppo e le caratteristiche degli LLM; verranno introdotti i concetti di accessibilità e disabilità, le linee guida da seguire per sviluppare prodotti accessibili, le leggi italiane che regolano tale materia, il rapporto tra accessibilità, usabilità ed inclusività, nonché le pratiche esistenti per verificare l'accessibilità di un prodotto; in ultimo, verranno citate le ricerche esistenti che si pongono come obiettivo quello di utilizzare gli LLM per migliorare lo sviluppo di prodotti web accessibili nonché per la valutazione di correttezza di un dato codice.

2.1 LLM

Un LLM (Large Language Model) è un modello di intelligenza artificiale, in particolare delle reti neurali, di grandi dimensioni e preaddestrato su un ampio numero di informazioni. E' in grado di comprendere e generare testi, audio, immagini e file di varia natura[5].

L'addestramento di questi modelli è definito **apprendimento non supervisionato** o **auto-supervisionato**, ovvero i dati di addestramento che vengono forniti al modello non sono stati precedentemente etichettati; è quindi possibile utilizzare quantità di dati enormi in modo agevole. Il modello, analizzando le correlazioni all'interno dei dati di addestramento, è in grado di trovare le relazioni più probabili tra le parole e di comprendere il loro significato in base al contesto, ricavando quindi delle generalizzazioni da applicare a svariati contesti[7].

2.1.1 Reti neurali

Una rete neurale¹ è costituita da layers di neuroni. Ogni neurone è collegato a tutti quelli dello strato precedente e successivo e ogni strato elabora le informazioni ricevute dagli strati precedenti e passa l'output agli strati successivi (figura 2.1).

L'introduzione dei transformers ha creato le condizioni per cui le reti neurali si sviluppassero sempre più.

¹Modelli ispirati alle reti neurali dei cervelli degli animali.

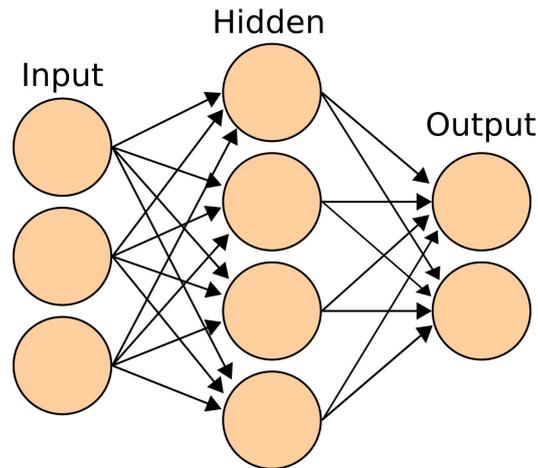


Figura 2.1: Struttura di una rete neurale

Transformers

I transformers sono in grado di ricevere in input un dato e restituirlo in output modificato. Utilizzano un **meccanismo di attenzione**[29] che permette al modello di assegnare ad ogni elemento ricevuto in input un valore di "importanza" rispetto agli altri elementi.

Un transformer è costituito da un encoder e un decoder[29]:

- **encoder:** prende in input un dato e ne produce una rappresentazione pesata indicante l'attenzione delle parole in input;
- **decoder:** prende l'output dell'encoder e genera un output che tiene conto dell'intero contesto dell'input.

2.1.2 Addestramento ed ottimizzazione degli LLM

L'addestramento di un LLM è una fase cruciale in quanto determina la qualità di comprensione e generazione di linguaggio naturale del modello stesso. L'addestramento è costituito da due fasi: il pre-addestramento ed il fine-tuning. Esistono anche delle tecniche di ottimizzazione degli LLM che permettono di migliorare le risposte generate dal modello. Tra queste troviamo il fine-tuning stesso e il RAG.

Pre-addestramento

E' la prima fase dell'addestramento di un LLM dove vengono forniti al modello un insieme vasto di dati testuali proveniente da diverse fonti, come ad esempio libri o internet, di modo che il modello inizi a riconoscere la struttura del linguaggio e le relazioni semantiche tra le parole in modo non supervisionato.

In questa fase, quindi, il modello acquisisce capacità generali di comprensione del linguaggio che gli permettono di effettuare una vasta gamma di compiti non specializzati[28].

Fine-tuning

E' una tecnica di apprendimento supervisionato dove il modello viene addestrato su task specifici. In questa fase è possibile effettuare un fine-tuning su tutta la struttura dell'LLM, andando a modificare tutti i pesi dei parametri del modello, oppure solo su alcuni di essi.

Questa fase di addestramento può durare molto, anche alcuni mesi, in quanto è necessario avere un insieme, anche vasto, di dati strutturati specifici sullo use case su cui si vuole trainare il modello e che descrivano il risultato atteso; i dati quindi devono essere rielaborati prima di poterli fornire all'LLM[14].

RAG

RAG sta per **Retrieval-Augmented Generation** ed è un metodo avente come obiettivo quello di potenziare le capacità degli LLM migliorandone l'output.

Esso si basa sul fornire al modello delle fonti autorevoli, dette **dati esterni**, su cui l'LLM non è stato trainato e da cui può ricavare le informazioni necessarie per svolgere un dato problema senza doverlo riaddestrare[6]. Questa tecnica fa sì che le risposte generate dall'LLM siano più precise e pertinenti alla domanda posta dall'utente, nonché riduce la possibilità che vengano generate delle risposte con informazioni inesatte.

2.1.3 Vantaggi e limitazioni

Gli LLM vengono sempre più usati in moltissimi ambiti lavorativi poiché permettono di migliorare ed efficientare numerosi processi. Tuttavia presentano sia dei vantaggi che delle limitazioni[20].

Vantaggi

- **versatilità:** poiché vengono addestrati con moltissimi dati testuali di diversa natura, possono essere utilizzati in diversi casi d'uso, anche senza dover effettuare modifiche al modello base. Inoltre, con il fine-tuning o il RAG è possibile ampliare ed adattare le capacità di un LLM pre-trainato;
- **efficienza:** permettono di velocizzare processi complessi e ripetitivi; è possibile far analizzare grandi volumi di dati per identificare informazioni rilevanti.

Limitazioni

- **elevato consumo di risorse computazionali:** per poter addestrare un modello ed effettuare un fine-tuning su un LLM sono necessarie notevoli risorse computazionali con un dispendio rilevante di energia, costi ed impatto ambientale. Questo fa sì che non tutte le organizzazioni possano avere accesso a questa tecnologia, soprattutto se si volesse addestrare un LLM senza utilizzarne uno pre-addestrato;
- **bias:** essendo addestrati su documenti pubblici di varia natura, i bias contenuti in questi documenti si ripercuotono sulle risposte che questi modelli forniscono[22];
- **output dipendente dalla qualità dei dati di training:** per poter avere buone risposte da parte del modello, è necessario avere dei buoni dati di training. Poiché però gli LLM vengono allenati su dati non necessariamente verificati o non di buona qualità, l'output può non essere ottimale. Nel caso di generazione di codice, ad esempio, se tra i dati di training si hanno dei pezzi di codice formalmente errati o che contengono degli errori, il modello potrebbe non essere in grado di identificarli e quindi riproporli nel codice generato[22];

- **allucinazioni**: è possibile che il modello generi risposte che non sono basate ne sui dati di addestramento ne sul contesto, di fatto inventando la risposta;
- **black box**: non sempre è possibile comprendere quale sia il processo decisionale che ha portato un dato modello a fornire un determinato output;
- **leak di informazioni private**: è stato dimostrato che gli LLM possano fornire informazioni private fornitegli in un precedente utilizzo, sia agli sviluppatori che gestiscono il modello, che ad utenti terzi[22].

2.2 Accessibilità

Per **accessibilità** si intende *la capacità di un dispositivo, servizio o risorsa di essere fruibile con facilità da una qualsiasi tipologia d'utente* [18]. Non si rivolge quindi a un gruppo definito di utenti con particolari necessità di interazione e a dispositivi rivolti ad utenti portatori di disabilità, ma a un gruppo ampio, non ben delimitato, di utenti e alla facilità di interazioni che questi hanno con un qualsiasi dispositivo o risorsa.

Accanto al concetto di accessibilità si ha quello di disabilità. Esistono diverse tipologie di **disabilità** in quanto questa è il gap tra le caratteristiche dell'individuo e il modo che questo ha di interagire con la società. In particolar modo si possono avere disabilità (figura 2.2):

- **permanenti**, ad esempio una persona affetta da mutismo;
- **temporanee**, durature nel tempo ma non permanenti, come ad esempio quando si ha un braccio ingessato;
- **situazionali**, che dipendono dalla situazione in cui l'utente si trova, ad esempio se si ha un bambino in braccio non si può interagire con la tastiera del pc con due mani.

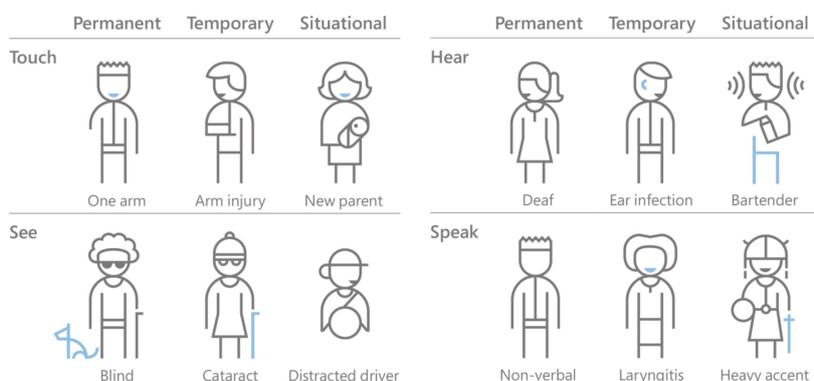


Figura 2.2: Esempi di disabilità permanenti, temporanee e situazionali per i sensi del tatto, dell'udito, della vista e della parola

L'accessibilità non è un tema recente. Sin dal 1997 il W3C (World Wide Web Consortium) lanciò l'iniziativa **WAI (Web Accessibility Initiative)** avente come scopo lo sviluppo di standard e materiali di supporto per sviluppare un web più accessibile[12]. Tra le guide redatte nell'ambito del WAI, le **WCAG (Web Content Accessibility Guidelines)** sono lo standard

di fatto dell'accessibilità web. L'ultima versione contrassegnata come *recomendation* è la 2.2 ed attualmente il consorzio sta lavorando alla definizione della versione 3.0 che diventerà lo standard di accessibilità nel prossimo futuro.

2.2.1 WCAG 2.2

Le WCAG 2.2 sono state pubblicate il 5 ottobre 2023 e si pongono l'obiettivo di rendere il web più accessibile a una grande varietà di disabilità, consci del fatto che, soprattutto nel caso in cui si hanno più patologie, l'accessibilità in toto non può essere garantita[13].

Sono composte da 4 principi che rappresentano le basi portanti dell'accessibilità. Se anche solo uno di questi venisse violato, un utente con disabilità non sarebbe in grado di utilizzare il web:

- **Percepibile:** le informazioni e i componenti dell'interfaccia utente devono essere percepibili da parte dell'utente, ovvero l'utente deve essere a conoscenza che nella pagina è presente una data informazione o componente dell'interfaccia. Questa conoscenza può avvenire tramite uno dei suoi sensi;
- **Operabile:** l'interfaccia utente e i suoi componenti devono avere delle funzionalità che l'utente può effettuare;
- **Comprensibile:** l'utente deve comprendere sia le informazioni che le funzionalità dell'interfaccia utente;
- **Robusto:** i contenuti devono essere accessibili da diversi user agents, incluse le tecnologie assistive.

Ad ogni principio vengono associate delle **linee guida**, 13 in totale, che rappresentano gli obiettivi principali su cui lo sviluppatore, lo UX e lo UI devono lavorare per rendere i contenuti digitali più accessibili agli utenti con disabilità. Un aspetto fondamentale delle linee guida è quello di assicurarsi che il contenuto sia accessibile a più persone possibili e che possa essere rappresentato in diverse forme così che gli utenti possano accedervi attraverso diverse abilità sensoriali, fisiche e cognitive[9].

Sotto ogni linea guida si hanno uno o più **criteri di successo** che descrivono, nel dettaglio, cosa deve essere raggiunto per poter essere considerato conforme al criterio. Non si riferiscono a una tecnologia specifica, così da poter essere adattate a contesti differenti. Ogni criterio di successo è associato a un **livello di conformità**. I livelli sono:

- A
- AA
- AAA

Il livello A è il livello base; non soddisfare uno di questi criteri può rendere totalmente inaccessibile il contenuto digitale a un utente con disabilità. Il livello AAA, invece, è il più complicato da raggiungere e spesso necessita di un redesign del contenuto digitale.

In ultimo, per ogni linea guida e criterio di successo, si hanno delle **tecniche sufficienti e consigliate** che rappresentano delle modalità consolidate per soddisfare un dato criterio di successo o per migliorare l'accessibilità.

2.2.2 Legge Stanca

Un aspetto importante che spinge le aziende a considerare l'accessibilità all'interno dei loro touchpoint digitali è quello normativo. In Italia, la legge di riferimento è la Legge Stanca, pubblicata in Gazzetta Ufficiale il 17 gennaio 2004.

Inizialmente l'obiettivo della legge era quello di garantire alle persone con disabilità di accedere ai servizi informatici e telematici delle sole pubbliche amministrazioni. Il Decreto Legge 76/2020 ha esteso alcuni obblighi anche ai soggetti privati che offrono servizi al pubblico attraverso siti web o applicazioni mobile aventi un fatturato medio, negli ultimi tre anni di attività, superiore a cinquecento milioni di euro[4]. Proprio quest'ultimo aggiornamento ha dato l'impulso definitivo per cui, sempre più aziende, hanno iniziato a considerare l'accessibilità all'interno dei loro servizi digitali.

La legge si rifà alle WCAG 2.1 e pone, come livello minimo per considerare un sito conforme alla legge, il rispetto del livello di conformità AA.

2.2.3 Accessibilità, Usabilità ed Inclusività

Il concetto di accessibilità si sposa anche con quello di usabilità ed inclusività, seppur con delle differenze. Mentre l'**accessibilità**, ed in particolar modo quella web, si pone l'obiettivo di permettere alle persone con disabilità di *percepire, comprendere, navigare ed interagire* con i siti web, l'**usabilità** si concentra sul progettare prodotti efficaci, efficienti e soddisfacenti. Include quindi degli aspetti di *user experience design* che possono avere impatti su tutti gli utenti, non solo quelli con disabilità. L'**inclusività**, invece, si occupa di diversità, promuove l'uguaglianza ed il maggior coinvolgimento possibile. Si parla quindi di *universal design*.

E' bene ricordare però che migliorare l'accessibilità significa migliorare o risolvere alcuni problemi di usabilità e di inclusività e viceversa. Quindi, aggregare questi tre concetti durante la progettazione e sviluppo di una risorsa digitale contribuisce a rendere il web un posto più accessibile, usabile ed inclusivo per tutti[8].

2.2.4 Testare l'accessibilità

Testare l'accessibilità è fondamentale per creare prodotti fruibili da tutti. Al contempo, però, è anche la fase più critica in quanto rendere un prodotto accessibile a più persone possibili necessita test su diversi dispositivi e con diverse tipologie di utenti.

Esistono diverse metodologie di testing:

- test manuali da parte di accessibility experts;
- test automatici;
- test con utenti.

Durante i test manuali effettuati da esperti è utile utilizzare diversi tool, che possono automatizzare parte del test, o documenti di supporto, tra cui:

- checklist WCAG - ne esistono diverse online, tra cui una ufficiale delle WCAG 2.0;
- estensioni del browser che segnalano alcune problematiche, come color contrast, l'assenza di testi alternativi per le immagini, per i link o i pulsanti;
- Lighthouse - strumento di google integrato negli strumenti per sviluppatori che, tra le varie funzioni, può fornire report sull'accessibilità di una pagina web;

- verifica della responsività del sito web fino ad almeno il 200% dello zoom;
- navigazione da tastiera senza l'utilizzo di un lettore di schermo;
- screen reader quali NVDA, VoiceOver, Narrator.

Un test approfondito però richiederebbe di dover testare il sistema con diverse tecnologie assistive oltre allo screen reader². E' inoltre fondamentale effettuare dei test con gli utenti con diverse disabilità e con diversi livelli di gravità della disabilità. E' proprio questa tipologia di test che permette di verificare, veramente, se un sito sia accessibile oppure no.

Effettuare delle expert review o dei test con gli utenti necessita però di tanto tempo, che spesso all'interno dei progetti non si ha, ma soprattutto queste tipologie di test sono possibili solo in una fase di sviluppo avanzata o addirittura post release del prodotto. Ciò comporta un elevato costo, in termini di tempo e monetario, per poter effettuare delle modifiche e sistemare le problematiche di accessibilità. Riuscire a identificare più problematiche possibili già nelle prime fasi di sviluppo del prodotto permetterebbe di creare dei prodotti più accessibili sin dalla prima release e a costo minore.

2.3 Lavori correlati

Esistono diversi studi che provano ad analizzare le capacità degli LLM di comprendere ed identificare errori all'interno del codice, tra cui anche l'identificazione di problematiche di accessibilità nelle pagine web.

In particolare si cita la ricerca «Turning manual web accessibility success criteria into automatic: an LLM-based approach». Essa aveva l'obiettivo di valutare se un LLM fosse in grado di valutare la correttezza di alcuni criteri delle WCAG³ che normalmente richiedono una valutazione manuale poiché necessitano la comprensione del contesto che i tool di analisi automatici spesso non hanno. In questo paper i test condotti con gli LLM vengono effettuati anche con noti tool di analisi automatici⁴ ed i risultati messi a confronto. Alcuni tool automatici, infatti, hanno segnalato come passati test che, con un'analisi che prenda in considerazione il contesto dell'applicazione, un expert review avrebbe segnalato come non passato. Ad esempio ad una immagine rappresentante il logo di W3C è stato fornito un alt text che riportava come descrizione *a black dog*; i tool di analisi automatici hanno segnalato il test come passato, in quanto verificano solo che il tag img abbia un alt text e non che il testo inserito sia significativo per l'immagine che descrive. Il test con LLM, invece, ha portato risultati migliori in quanto si è utilizzato un primo LLM che fosse in grado di descrivere l'immagine e un secondo LLM che verificasse quanto la descrizione fornita dal primo LLM e quella presente nell'alt text fossero differenti.

Da questa ricerca, seppur limitata su 3 principi delle WCAG, si evince che l'utilizzo di LLM può automatizzare alcuni test che ad oggi devono essere svolti manualmente in quanto gli attuali test automatici non forniscono risultati affidabili.

Un'ulteriore ricerca delle università *University of California, Davis* e *Temple University*, intitolata «Large Language Models for Code Analysis: Do LLMs Really Do Their Job?», ha analizzato le capacità di *code analysis* dei più diffusi LLM. In particolare, lo studio ha previsto

²Display braille, software di ingrandimento dello schermo, dispositivi di puntamento diversi dal mouse tradizionale.

³1.1.1 Non-text Content, 2.4.4 Link Purpose (In Context), 3.1.2 Language of Parts.

⁴A11y, Pa11y, Mauve++, AChecker, AccessMonitor, Lighthouse.

l'analisi sia di codice offuscato⁵ che non offuscato. I risultati di questa ricerca evidenziano diverse criticità riscontrate nelle capacità degli LLM attuali di analisi del codice, soprattutto se l'analisi viene effettuata su un codice che ha subito un processo di offuscamento.

In fine si menziona lo studio «Understanding HTML with Large Language Models» che si è concentrato sulle seguenti capacità degli LLM: classificazione semantica degli elementi HTML, generazione di descrizioni per input HTML e navigazione web autonoma. Da questo studio emerge l'importanza dell'architettura del modello, in quanto alcuni modelli performano meglio di altri, e di alcune limitazioni degli LLM nel momento in cui è stato condotto lo studio, ovvero una *context window* limitata che ha un impatto negativo sulle capacità di comprensione del modello.

⁵Code obfuscation.

Capitolo 3

Design

In questo capitolo analizzeremo tre possibili soluzioni per realizzare uno strumento che rilevi gli errori di accessibilità nel codice e, tramite un LLM, suggerisca allo sviluppatore una possibile soluzione. Si ha quindi l'obiettivo di formulare la soluzione più adatta per integrare gli LLM nella risoluzione di problemi di accessibilità web, andando a delineare i vantaggi e svantaggi di ogni soluzione.

Verrà inoltre delineato il contesto applicativo dello strumento, definendo le caratteristiche del codice che verrà analizzato e le tipologie di correzioni che l'LLM dovrà fornire.

3.1 Analisi delle possibili architetture

Esistono tre strade percorribili per sviluppare una soluzione che aiuti lo sviluppatore nella scrittura di codice accessibile.

In particolare le possibili architetture sono:

1. estensione del linter ESLint inserendo la funzionalità di risoluzione della problematica segnalata tramite LLM;
2. realizzazione di un'estensione di VS Code che si occupi di analizzare il codice alla ricerca di errori di accessibilità e, interrogando un LLM, suggerisca una possibile soluzione;
3. una soluzione ibrida tra le due elencate sopra: creare un'estensione per VS Code che utilizzi ESLint per identificare gli errori di accessibilità e un LLM per fornire una possibile soluzione.

3.1.1 Soluzione 1 - Estensione del linter ESLint

ESLint è un linter per il linguaggio Javascript utilizzato per analizzare staticamente il codice sorgente. Il suo principale obiettivo è quello di identificare parti di codice problematici o che non rispettino lo stile di codifica prestabilito. Esso si basa sulla definizione di regole che il codice deve seguire. Se si avesse bisogno di regole custom per un singolo progetto, è possibile definirne di nuove. Quando viene trovata la violazione di una regola, essa viene segnalata riportando riga e colonna che hanno generato l'errore e una breve descrizione della problematica.

ESLint non è un linter specifico per verificare l'accessibilità. Si potrebbero utilizzare altri linter specifici¹, ma ESLint, poiché è più generico², malleabile e adattabile, è verosimilmente più utilizzato rispetto a linter specifici³. Inoltre esistono diversi plugin ben mantenuti che estendono le funzionalità di ESLint nell'analisi delle regole di accessibilità⁴.

I vantaggi nell'estendere ESLint quindi sono:

- la presenza di una vasta community;
- diversi plugin esistenti e aggiornati regolarmente;
- l'esistenza di diversi plugin per l'accessibilità specifici per diverse librerie Javascript;
- la possibilità di inserire regole custom specifiche per un dato progetto;
- il suo largo utilizzo da parte degli sviluppatori.

Al contempo però, essendo un progetto ben strutturato, ha un'architettura ben definita e difficile da modificare. In particolare, per poter interrogare un LLM per fornire una possibile soluzione all'errore segnalato, bisognerebbe definire una nuova regola per ogni principio di accessibilità da verificare e, per ognuna di essa, interrogare l'LLM. Questo si tradurrebbe in un rallentamento delle prestazioni, in quanto per mostrare le segnalazioni prima bisognerebbe aspettare che le varie interrogazioni poste all'LLM ricevano una risposta.

Utilizzare ESLint per verificare se esiste un problema di accessibilità pone un'ulteriore problematica: poiché è focalizzato sulla **verifica sintattica e di stile del codice**, le problematiche legate a una **verifica semantica** non verrebbero evidenziate.

Consideriamo ad esempio una immagine. Il tag `img`⁵ prevede l'inserimento dell'attributo **alt** che specifichi il testo alternativo che deve essere presentato alle persone che interagiscono con la pagina web attraverso una tecnologia assistiva o a quelle che, per motivi di connessione, non riescono a scaricare correttamente l'immagine. Se l'immagine ha il solo scopo decorativo e non fornisce alcun significato aggiuntivo all'interno della pagina web, l'alt deve essere posto pari a stringa vuota (`alt=""`)[11], in alternativa deve spiegare il significato semantico dell'immagine all'interno del contesto della pagina web[10]. In questi casi ESLint si limiterà a segnalare l'assenza dell'attributo **alt** e non potrà alcun tipo di verifica sul testo inserito come descrizione, che quindi dovrà essere verificato tramite un test manuale.

3.1.2 Soluzione 2 - Realizzazione di un'estensione per VS Code

VS Code è un editor di codice sorgente sviluppato da Microsoft. Supporta diversi linguaggi ed è caratterizzato da un'architettura estendibile grazie all'utilizzo di estensioni sviluppate dalla community[23].

Sviluppare un'estensione per l'editor permette una migliore user experience per lo sviluppatore. Infatti, tramite estensione, è possibile:

¹Deque offre diverse soluzioni per integrare l'accessibilità sin dalle prime fasi di sviluppo, tra cui uno tra i migliori linter per verificare l'accessibilità del codice.

²Permette un'analisi generale di Javascript.

³Se si considerano solo i download delle estensioni per VS Code, ESLint ha oltre 36 milioni e 750 mila download contro i quasi 465 mila dell'estensione di axe Accessibility Linter.

⁴`eslint-plugin-jsx-a11y`, `eslint-plugin-react-native-a11y`, `eslint-plugin-vuejs-accessibility` sono plugin di ESLint che implementano dei check sulle regole di accessibilità rispettivamente per JSX, React Native e Vue.js.

⁵Tag HTML per inserire una immagine in una pagina web.

- comunicare in tempo reale la presenza di errori rilevati analizzando il codice staticamente;
- fornire la risoluzione del problema tramite LLM solo se lo sviluppatore lo richiede; i tempi di segnalazione degli errori quindi non subiscono impatti;
- integrare la funzionalità all'interno del workflow dello sviluppatore permettendo anche la interoperabilità con altre estensioni o configurazioni custom;
- inserire una voce di menu all'interno dell'editor che permetta allo sviluppatore di interrogare un LLM per verificare se il codice selezionato ha degli errori di accessibilità ed, eventualmente, ricevere supporto nella correzione di questi.

Al contempo però, per poter segnalare gli errori di accessibilità del codice senza interrogare un LLM ogni volta che un utente modifichi o salvi un file e non affidandosi a librerie già rodiate che segnalino, anche in parte, gli errori di accessibilità, è necessario creare delle funzioni che rilevino tali errori.

La difficoltà di questa soluzione sta proprio nella realizzazione dell'analisi del codice e nella identificazione degli errori di accessibilità. In particolare, dovendosi basare sui principi e le soluzioni tecniche suggerite nelle WCAG, bisognerebbe creare un linter che vada ad analizzare ogni linea guida, e quindi ogni criterio di successo, la cui correttezza possa essere verificata staticamente analizzando il codice sorgente.

Si potrebbe, inoltre, pensare che la creazione di un'estensione solo per VS Code limiti l'ambito di utilizzo della stessa. Sebbene questo sia vero, poiché l'estensione in questione ha l'obiettivo di aiutare gli sviluppatori nello sviluppo di codice accessibile per il web, VS Code è uno degli editor di testo più utilizzato dagli sviluppatori front-end, di conseguenza la limitazione è accettabile.

3.1.3 Soluzione 3 - Creazione di un'estensione per VS Code e utilizzo di ESLint

Nei due paragrafi precedenti sono stati analizzati i pro ed i contro delle due soluzioni proposte. La soluzione che viene illustrata in questo paragrafo combina i vantaggi delle altre due sopra citate.

In particolare, si potrebbe:

- creare un'estensione per VS Code così da permettere una migliore interazione dello sviluppatore con lo strumento, nonché di migliorare il numero di interrogazioni con l'LLM in quanto verrebbero sottoposte solo le problematiche che lo sviluppatore vuole risolvere;
- utilizzare ESLint come linter per identificare gli errori di accessibilità così da utilizzare uno strumento accreditato all'interno della community degli sviluppatori.

L'estensione quindi dovrebbe identificare quando ESLint segnala un errore legato all'accessibilità e, in quel caso, permettere allo sviluppatore di interrogare un LLM per ricevere supporto nella risoluzione della problematica. Inoltre, a richiesta dello sviluppatore, si potrà interrogare un LLM che, datogli in pasto un pezzo di codice, restituirà i potenziali errori di accessibilità e le possibili soluzioni.

3.2 Contesto applicativo

Dall'analisi effettuata nel paragrafo 3.1 la soluzione architetturale migliore risulta essere la terza descritta del paragrafo 3.1.3, ovvero la realizzazione di un'estensione che utilizzi ESLint per

analizzare staticamente il codice e un LLM per analizzare il codice e ricevere dei suggerimenti di correzione.

L'estensione quindi dovrà funzionare per i file scritti in Javascript e TypeScript, nonché riconoscere il Javascript utilizzato dentro un progetto React, anche quando questo utilizza TypeScript.

Nei progetti dove è configurato ESLint⁶, l'estensione dovrà riconoscere se la segnalazione del linter faccia riferimento a una problematica di accessibilità. Se così fosse, lo sviluppatore potrà decidere di ricevere una proposta di correzione tramite l'interrogazione di un LLM.

In tutti i progetti, anche dove non sia configurato ESLint, lo sviluppatore potrà scegliere delle righe di codice da sottoporre all'analisi di un LLM che indicherà la presenza o meno di errori di accessibilità e, se necessario, fornirà una possibile correzione. Questa tipologia di analisi non si limiterà alla sola verifica sintattica del codice, ma anche, ove possibile, a una verifica semantica. Inoltre, nel caso di elementi che prevedono una interazione attraverso la tastiera⁷, verrà verificato che tutte le tipologie di interazione che l'utente si aspetta siano rispettate.

3.3 Design Framework

L'estensione dovrà risultare facile da utilizzare e ben integrata all'interno del workflow di uno sviluppatore. Per questo motivo verranno attenzionate le prestazioni, soprattutto nell'interrogazione dell'LLM, in quanto questo potrà richiedere del tempo per effettuare l'analisi richiesta e restituire una soluzione appropriata.

Quando si interroga un LLM, la formulazione di un buon prompt è essenziale per ricevere risposte precise, pertinenti e coerenti. Le tecniche volte ad ottimizzare i prompt ricadono sotto il nome di **prompt engineering**. Nel seguente paragrafo ne verranno analizzate alcune tra le più rilevanti.

3.3.1 Prompt engineering

Il prompt engineering è una disciplina avente come obiettivo quello di sviluppare ed ottimizzare prompt per utilizzare al meglio le capacità degli LLM[15].

Un prompt è l'input che l'utente fornisce all'intelligenza generativa per ricevere in output le informazioni richieste. Esso può essere composto da diversi elementi, il cui utilizzo dipende dal compito da svolgere (figura 3.1):

- **Istruzione:** definisce il compito o l'istruzione che il modello dovrà eseguire;
- **Contesto:** esplica il contesto del compito da svolgere, così da indirizzare il modello a restituire risposte migliori;

⁶Si suggerisce di utilizzare il plugin `eslint-plugin-jsx-a11y` che presenta delle regole specifiche per l'accessibilità per JSX.

⁷Il WAI ha redatto il APG (ARIA Authoring Practices Guide), un documento contenente degli esempi e guide su come utilizzare al meglio gli ARIA (Accessible Rich Internet Applications). Vi è una sezione contenente dei pattern di programmazione dove si ha la descrizione di ogni elemento, gli attributi che questo deve possedere e l'interazione da tastiera che l'utente si aspetta di avere. Ad esempio, al seguente link <https://www.w3.org/WAI/ARIA/apg/patterns/accordion/> è possibile trovare la descrizione del pattern di un *accordion*. In questo si ha la descrizione delle diverse interazioni da tastiera - i pulsanti *enter* e *space* aprono o chiudono l'accordion, il *tab* sposta il focus sul prossimo elemento interattivo, ecc. - nonché gli stati e ruoli che l'elemento dovrà possedere.

- **Dati in input:** domanda per la quale si richiede che l'LLM generi una risposta;
- **Indicatore dell'output:** indicazione sul tipo o il formato dell'output che l'LLM dovrà restituire.

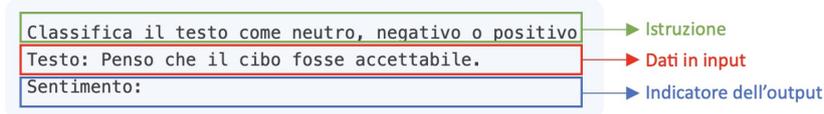


Figura 3.1: Esempio di prompt con l'utilizzo di un'istruzione, dei dati in input e dell'indicatore dell'output

La generazione di un prompt è un processo iterativo dove i risultati migliori si trovano dopo diverse prove. A volte però si può ricadere nel *blind prompting*, ovvero nella generazione di prompt tramite tentativi casuali privi di una strategia precisa. Per questo motivo esistono delle best practice e delle tecniche che, utilizzate in modo incrementale, possono aiutare a generare prompt migliori. In questo paragrafo ne analizzeremo alcune.

Best practice per un prompt efficace

Come già detto, quando si scrive un prompt è possibile utilizzare alcune accortezze per massimizzare le capacità dell'LLM che si sta utilizzando. In tutti i casi, l'importante è partire da un prompt **semplice**.

Le tecniche più consigliate sono:[17, 25, 26]

- l'utilizzo di **istruzioni specifiche**: più si danno informazioni precise e dettagliate sull'informazione che si vuole avere in output, più il modello risponderà con le informazioni richieste. Per questo è possibile utilizzare dei comandi come *scrivi*, *classifica*, *riassumi*. E' importante però non includere dettagli non necessari e mantenere un prompt relativamente breve;
- è importante **evitare l'imprecisione**. Si potrebbe pensare che un prompt del tipo

Spiega il concetto di prompt engineering. Mantieni la spiegazione breve, solo poche frasi, e non essere troppo descrittivo.

sia un buon prompt. Sebbene si potrebbero avere dei buoni output, esso presenta alcune problematiche: non è chiaro quante frasi usare e con quale stile, quindi i risultati potrebbero variare da esecuzione ad esecuzione. Un prompt migliore potrebbe essere

Usa 2-3 frasi per spiegare il concetto di prompt engineering a uno studente di scuola superiore.

- se l'elaborazione del risultato dovesse richiedere diverse fasi, è bene **specificare i vari step** da prevedere nell'elaborazione dell'output (figura 3.2);

```

Utilizza le seguenti istruzioni passo-passo per rispondere agli input
dell'utente.
Fase 1 - L'utente fornirà un testo tra virgolette tripla. Riassumete il testo in
una frase con il prefisso "Riassunto: ".
Fase 2 - Traducete il riassunto della fase 1 in spagnolo, con un prefisso che
dice "Traduzione": ".

```

Figura 3.2: Esempio di prompt che definisce due step per rispondere agli input dell'utente. Nella prima fase il modello dovrà riassumere il testo fornito dall'utente, inserendo il prefisso "Riassunto:", mentre nella seconda fase dovrà tradurre il riassunto in spagnolo e restituire la traduzione anteponendo la frase "Traduzione:"

- è possibile definire il **formato**, lo **stile**, la **formattazione** e i **contenuti** che si vorrebbero avere nella risposta. In questi casi è necessario fornire dei template o degli esempi di risposta (figura 3.3);

```

Estrarre le entità importanti menzionate nel testo sottostante. Estrarre prima
tutti i nomi delle aziende, poi tutti i nomi delle persone, quindi estrarre gli
argomenti specifici che si adattano al contenuto e infine estrarre i temi
generali.

Formato desiderato:
Nomi di aziende: <elenco_separato_da_nomi_di_aziende>
Nomi di persone: -||-
Argomenti specifici: -||-
Temi generali: -||-

Testo: {testo}

```

Figura 3.3: Esempio di prompt che definisce il template per l'output da ricevere

- utilizzare il **rinforzo positivo**, ovvero esprimere cosa il modello può fare, invece di dire cosa non può fare. In questo modo si ha una maggiore specificità del prompt e vengono inseriti i dettagli che poi porteranno a una risposta migliore. Ad esempio, nel prompt di figura 3.4 è stato detto al modello di non chiedere interessi o informazioni personali utilizzando delle frasi contenenti delle negazioni.

```

Di seguito viene presentato un agente che consiglia dei film a un cliente. NON
CHIEDERE INTERESSI. NON CHIEDERE INFORMAZIONI PERSONALI.
Cliente: Per favore, raccomandami un film basato sui miei interessi.
Agente:

```

Figura 3.4: Esempio di prompt dove viene indicato al modello di non chiedere informazioni personali o interessi, utilizzando frasi negative invece che la tecnica del **rinforzo positivo**

Un possibile output può essere quello rappresentato in figura 3.5, dove il modello domanda proprio le informazioni che gli era stato detto di non chiedere.

Certo, posso consigliarle un film in base ai suoi interessi. Che tipo di film le piacerebbe vedere? Preferisce l'azione, la commedia, il romanticismo o altro?

Figura 3.5: Esempio di risposta al prompt di figura 3.4 dove l'LLM chiede le informazioni personali che gli era stato detto di non chiedere

Per far sì che il modello non vada a chiedere all'utente delle informazioni non richieste, un prompt migliore potrebbe essere quello rappresentato in figura 3.6, dove all'LLM viene comunicato che non può chiedere le preferenze degli utenti o le loro informazioni personali attraverso le sintassi *astenersi dal chiedere* oppure *evitare di chiedere*.

Il seguente è un agente che raccomanda film a un cliente. L'agente ha il compito di consigliare un film tra quelli di maggiore tendenza a livello globale. Deve astenersi dal chiedere agli utenti le loro preferenze ed evitare di chiedere informazioni personali. Se l'agente non ha un film da raccomandare, deve rispondere "Spiacente, non è stato possibile trovare un film da raccomandare oggi".
 Cliente: Per favore, mi raccomandi un film basato sui miei interessi.
 Agente:

Figura 3.6: Esempio di prompt con la tecnica del **rinforzo positivo**; al modello viene detto cosa fare. Nel caso in cui non debba fare qualcosa, questo viene veicolato con la forma *astenersi dal chiedere*, *evitare di chiedere* invece che con una negazione

Nell'input del prompt di figura 3.6 l'utente chiede al modello di suggerirgli dei film in base ai gusti personali dell'utente stesso. Poiché però l'LLM è stato istruito affinché non chieda tali informazioni attraverso la tecnica del rinforzo positivo, esso risponde comunicando che non riesce a fornire tale risposta, ma comunque suggerire dei film di tendenza (figura 3.7);

Mi dispiace, non ho informazioni sui tuoi interessi. Tuttavia, ecco un elenco dei film di tendenza a livello globale in questo momento: [elenco dei film]. Spero che troviate qualcosa di vostro gradimento!

Figura 3.7: Esempio di risposta al prompt di figura 3.6. L'utente chiede di ricevere dei suggerimenti in base ai suoi gusti personali. Poiché il modello non può chiedere queste informazioni, risponde che non può fornire risposte in linea con la domanda fatta

- utilizzare dei **separatori** come `###` o `"""` per delimitare le istruzioni dal contesto (figura 3.8);

Riassumi il testo in una lista con i punti più importanti.

```
Testo: """"
{testo di input}
""""
```

Figura 3.8: Esempio di prompt che inserisce dei delimitatori per separare le istruzioni dal contesto

- assegnare al modello un **ruolo**; questo permetterà di determinare meglio l'area di competenza che dovrà utilizzare per fornire una risposta che sarà più approfondita e mirata (figura 3.9);

Sei un esperto di comunicazione e marketing con anni di esperienza nella creazione di strategie di branding, campagne pubblicitarie e gestione dei social media. Sei chiamato a consigliare un'azienda su come migliorare la sua presenza online e raggiungere nuovi clienti.

Figura 3.9: Esempio di prompt che assegna un ruolo al modello. In questo caso il modello sa di essere un esperto di comunicazione e marketing con diversi anni di esperienza in alcuni campi del suo lavoro

- fornire al modello dei **testi di riferimento** dove potrà trovare le informazioni richieste. Questo permette di minimizzare le allucinazioni (figura 3.10);

Usa gli articoli forniti delimitati da tre virgolette per rispondere alle domande. Se la risposta non può essere trovata negli articoli, scrivi "Non posso trovare la risposta."

```
""""articolo1""""
""""articolo2""""
```

Domanda: <inserisci la domanda qui>

Figura 3.10: Esempio di prompt che fornisce degli articoli dove il modello potrà trovare le risposte

Tecniche di prompting

Oltre alle best practice citate nel paragrafo 3.3.1, esistono delle vere e proprie **tecniche di prompting** che permettono di realizzare compiti più complessi e di migliorare l'affidabilità e le prestazioni degli LLM.

Tra queste alcune sono:

- **priming prompts**: al modello viene fornito un input iniziale che dia il contesto di azione senza richiedere un'azione all'LLM. In figura 3.11 si ha un esempio di priming prompts dove, prima di porre una richiesta al modello, viene indicato il contesto dell'utente: esso è un designer UX con più di 15 anni di esperienza, attualmente sta progettando una interfaccia di login di un sito web dove gli utenti sono degli giovani adulti attenti alla moda e sta

lavorando sui messaggi di errore da mostrare nel login. Solo dopo questo preambolo verrà effettuata la vera richiesta all'LLM;

Sono un designer UX senior con 15 anni di esperienza con sede in Canada. Lavoro su un sito di e-commerce che vende occhiali da sole e da vista.

Attualmente sto progettando la schermata di login, dove. I nostri clienti sono giovani adulti attenti alla moda che apprezzano la qualità, l'autenticità e l'espressione di sé.

Sto lavorando a un messaggio di errore che gli utenti vedranno se tentano di accedere con credenziali (e-mail e password) che non corrispondono a quelle presenti nel nostro sistema.

Figura 3.11: Esempio di **priming prompt**

- **prompt Zero-Shot:** gli LLM sono addestrati su grandi quantità di dati. Per questo motivo esistono già delle istruzioni che riescono ad eseguire senza che nel prompt vengano forniti degli esempi. Questa tipologia di interazione è detta *Zero-Shot*. Ad esempio, il *sentiment* è qualcosa che un LLM riesce già a classificare. In figura 3.12 si ha un esempio di prompt Zero-Shot dove l'LLM è in grado di classificare un testo in base al sentiment;

Classificare il testo in neutro, negativo o positivo.
Testo: Penso che le vacanze siano ok.
Sentimento:

Figura 3.12: Esempio di **prompt Zero-Shot**. L'LLM è in grado di classificare un testo come *neutro*, *negativo* o *positivo* senza che gli vengano forniti degli esempi

- **prompt Few-Shot:** sebbene gli LLM siano in grado di rispondere in modo corretto a diversi compiti attraverso la tecnica dello *Zero-Shot*, quando si richiede di effettuare compiti più complessi potrebbe essere necessario fornire delle dimostrazioni all'interno del prompt. Questo permette al modello di apprendere dal contesto e di rispondere in maniera più pertinente (figura 3.13);

Prompt:

Il "whatpu" è un piccolo animale peloso originario della Tanzania. Un esempio di frase che utilizza la parola whatpu è:
Eravamo in viaggio in Africa e abbiamo visto questi simpaticissimi whatpu.
Fare un "farduddle" significa saltare su e giù molto velocemente.
Un esempio di frase che usa la parola farduddle è:

Output:

Quando abbiamo vinto la partita, abbiamo iniziato a fare farduddle tutti insieme.

Figura 3.13: Esempio di prompt e output della tecnica **prompt Few Shot**

- **prompt Chain-of-Thought (CoT)**: permette di ottenere capacità di ragionamento complesse attraverso fasi di ragionamento intermedie e può essere combinato con il *prompt Few-Shot* in compiti complessi che richiedono un ragionamento prima di rispondere. Nell'esempio di figura 3.14 nel prompt di input vengono forniti degli esempi di domanda e risposta che indicano al modello come dovrà ragionare per formulare lui la risposta richiesta. Questa proprietà si manifesta nei modelli linguistici sufficientemente grandi[16, 30];

```

Prompt:
La somma dei numeri dispari di questo gruppo corrisponde ad un numero pari: 4, 8, 9,
15, 12, 2, 1.
R: Sommando tutti i numeri dispari (9, 15, 1) si ottiene 25. La risposta è Falsa.
La somma dei numeri dispari di questo gruppo è un numero pari: 17, 10, 19, 4, 8, 12,
24.
R: La somma di tutti i numeri dispari (17, 19) risulta 36. La risposta è Corretta.
La somma dei numeri dispari di questo gruppo è un numero pari: 16, 11, 14, 4, 8,
13, 24.
R: La somma di tutti i numeri dispari (11, 13) risulta 24. La risposta è Corretta.
La somma dei numeri dispari di questo gruppo è un numero pari: 17, 9, 10, 12, 13,
4, 2.
R: La somma di tutti i numeri dispari (17, 9, 13) risulta 39. La risposta è Falsa.
I numeri dispari di questo gruppo si sommano in un numero pari.: 15, 32, 5, 13, 82,
7, 1.
R:

Output:
La somma di numeri dispari (15, 5, 13, 7, 1) risulta 41. La risposta è Falsa.

```

Figura 3.14: Esempio della tecnica **prompt Chain-of-Thought (CoT)**. Vengono forniti degli esempi di domande e risposte, così che il modello impara il tipo di ragionamento che deve porre in essere per poter formulare l'output

- **prompt Zero-Shot CoT**: è la variante *Zero-Shot* del *Chain-of-Thought*⁸. Consiste nell'inserire la frase "Pensiamo passo dopo passo". Essa infatti sembra innescare un meccanismo di ragionamento all'interno dell'LLM. Proviamo a sottoporre a un LLM un problema semplice di matematica. Ad esempio supponiamo di comprare 10 mele, 2 mele le diamo a un vicino e 2 al riparatore. Poi ne compriamo altre 5 e ne mangiamo 1. Chiediamo al modello quante mele ci sono rimaste e questo ci risponde 11, che è una risposta errata (figura 3.15).

```

Prompt:
Sono andato al mercato e ho comprato 10 mele. Ho dato 2 mele al vicino e 2 al
riparatore. Poi sono andato a comprare altre 5 mele e ne ho mangiata una. Quante
mele mi sono rimaste?

Output:
11 mele

```

Figura 3.15: Esempio di errore di risposta di un LLM quando gli viene sottoposto un semplice problema matematico e non viene usata la tecnica del **Chain-of-Thought**

⁸Formulata da Takeshi Kojima nel 2022.

Se invece allo stesso prompt viene inserita la frase *Pensiamo passo dopo passo*, l'LLM riesce a formulare la risposta corretta, ovvero 10 mele (figura 3.16)[16];

```
Prompt:
Sono andato al mercato e ho comprato 10 mele. Ho dato 2 mele al vicino e 2
all'operaio. Poi sono andato a comprare altre 5 mele e ne ho mangiata una. Quante
mele mi sono rimaste?
Pensiamo passo dopo passo.

Output:
Innanzitutto, si è partiti da 10 mele.
Hai dato via 2 mele al vicino e 2 all'operaio, quindi ti sono rimaste 6 mele.
Poi hai comprato altre 5 mele, per cui ora ne hai 11.
Infine, hai mangiato 1 mela, quindi ti sono rimaste 10 mele.
```

Figura 3.16: Esempio di prompt che utilizza la tecnica dello **Zero-Shot Chain-of-Thought** attraverso la quale il modello riesce a formulare la risposta corretta

- **Self-Consistency:** l'idea di base è quella di fornire al modello percorsi di ragionamento multipli tramite esempi all'interno del prompt - quindi utilizzando il *prompt Few-Shot* e, attraverso le generazioni, selezionare la risposta più coerente. Proviamo, ad esempio, a chiedere ad un LLM quanti anni ha una persona x, sapendo che una seconda persona y adesso ha 70 anni e, quando y aveva 6 anni, x aveva la metà degli anni di y. Il modello risponde *35*, che è errato (figura 3.17).

```
Prompt:
Quando avevo 6 anni mia sorella aveva la metà dei miei anni.
Ora ho 70 anni, quanti ne ha mia sorella?

Output:
35
```

Figura 3.17: Esempio di prompt e output dove non viene applicata la tecnica di **self-consistency**

Se invece forniamo una serie di esempi di ragionamento (figura 3.18) e proviamo a chiedere per tre volte il risultato all'LLM, questo per 2 volte risponde *67*, mentre una volta dà come risposta *35*. Di conseguenza, siccome *67* è la risposta che è stata data per più volte, è quella che viene selezionata come risposta finale;

- **prompt chaining:** quando si devono realizzare compiti complessi potrebbe essere necessario scrivere un prompt molto lungo e dettagliato; questo potrebbe mettere in difficoltà un LLM. In questi casi è quindi consigliabile utilizzare la tecnica del **prompt chaining**. Essa consiste nel dividere i compiti che il modello dovrà effettuare in sottocompiti. Per ogni sottocompito viene dato al modello un prompt il cui output verrà usato come input per un altro prompt. Di fatto si crea una catena di operazioni che l'LLM deve effettuare.

Nella realizzazione del prompt che verrà utilizzato per analizzare il codice alla ricerca di problematiche di accessibilità e formularne una risoluzione - nel caso in cui lo sviluppatore decida di analizzare delle righe di codice generiche - e di quello utilizzato per suggerire una correzione a una problematica segnalata da ESLint verranno utilizzate le tecniche sopra descritte, ove

D: Nel boschetto ci sono 15 alberi. Gli operai del boschetto planteranno gli alberi oggi. Quando avranno finito, ci saranno 21 alberi. Quanti alberi hanno piantato oggi gli operai del boschetto?

R: Iniziamo con 15 alberi. Successivamente abbiamo 21 alberi. La differenza deve essere il numero di alberi che hanno piantato. Quindi, devono aver piantato $21 - 15 = 6$ alberi. La risposta è 6.

D: Se ci sono 3 auto nel parcheggio e ne arrivano altre 2, quante auto ci sono nel parcheggio?

R: Ci sono già 3 macchine nel parcheggio. Ne arrivano altre 2. Ora ci sono $3 + 2 = 5$ auto. La risposta è 5.

D: Leah aveva 32 cioccolatini e sua sorella 42. Se ne hanno mangiati 35, quanti pezzi sono rimasti in totale?

R: Leah aveva 32 cioccolatini e la sorella di Leah ne aveva 42. Ciò significa che in origine c'erano $32 + 42 = 74$ cioccolatini. 35 sono stati mangiati. Quindi in totale hanno ancora $74 - 35 = 39$ cioccolatini. La risposta è 39.

D: Jason aveva 20 lecca-lecca. Ha dato a Denny alcuni lecca-lecca. Ora Jason ha 12 lecca-lecca. Quanti lecca-lecca Jason ha dato a Denny?

R: Jason aveva 20 lecca-lecca. Poiché ora ne ha solo 12, deve aver dato il resto a Denny. Il numero di lecca-lecca che ha dato a Denny deve essere $20 - 12 = 8$ lecca-lecca. La risposta è 8.

D: Shawn ha cinque giocattoli. Per Natale ha ricevuto due giocattoli ciascuno da mamma e papà. Quanti giocattoli ha ora?

R: Ha 5 giocattoli. Ne ha ricevuti 2 dalla mamma, quindi ha $5 + 2 = 7$ giocattoli. Poi ne ha ricevuti altri 2 dal papà, quindi in totale ha $7 + 2 = 9$ giocattoli. La risposta è 9.

D: Nella sala server c'erano nove computer. Ogni giorno, da lunedì a giovedì, sono stati installati altri cinque computer. Quanti computer ci sono ora nella sala server?

R: Ci sono 4 giorni da lunedì a giovedì. Ogni giorno sono stati aggiunti 5 computer. Ciò significa che in totale sono stati aggiunti $4 * 5 = 20$ computer. All'inizio c'erano 9 computer, quindi ora ci sono $9 + 20 = 29$ computer. La risposta è 29.

D: Michael aveva 58 palline da golf. Martedì ha perso 23 palline da golf. Mercoledì ha perso altre 2 palline. Quante palline da golf alla fine di mercoledì?

R: Michael aveva inizialmente 58 palline. Martedì ne ha perse 23, quindi ha $58 - 23 = 35$ palline. Il mercoledì ne ha perse altre 2, quindi ora ha $35 - 2 = 33$ palline. La risposta è 33.

D: Olivia ha 23 dollari e ha comprato cinque bagel a 3 dollari l'uno. Quanti soldi le rimangono?

R: Ha comprato 5 bagel a 3 dollari l'uno. Ciò significa che ha speso 15 dollari. Le rimangono 8 dollari.

D: Quando avevo 6 anni mia sorella aveva la metà dei miei anni. Ora che ho 70 anni, quanti anni ha mia sorella?

R:

Figura 3.18: Esempio della tecnica di **self-consistency**

necessario. Si farà attenzione a fornire al modello delle istruzioni precise, specificando diverse fasi di elaborazione; verrà specificato il ruolo che il modello dovrà assumere, nonché il formato di risposta che ci si aspetta. Le tecniche di prompt engineering di cui verosimilmente si avrà bisogno sono:

- **prompt chaining:** per la funzionalità di analisi del codice fornito dallo sviluppatore e l'elenco di possibili soluzioni agli errori trovati sarà necessaria una prima fase dove LLM identifichi gli errori e generi, secondo un preciso formato, delle descrizioni delle problematiche rilevate e una seconda fase dove vengano elaborate delle soluzioni. Sebbene sia possibile definire entrambe le fasi in un unico prompt, poiché le elaborazioni che il modello dovrebbe fare sono complesse, dividere queste fasi in più prompt potrebbe risultare ottimale;
- **prompt Few-Shot:** sia nella fase di identificazione delle problematiche che in quella di risoluzione delle stesse, fornire alcuni esempi di errori di accessibilità e di possibili risoluzioni potrà essere di aiuto al modello nella generazione della risposta.

Capitolo 4

Implementazione

Nel seguente capitolo si illustrerà il processo di implementazione del plugin per VS Code. Come discusso nel paragrafo 3.1, l'architettura che verrà realizzata utilizzerà sia ESLint per l'analisi statica del codice che un LLM per analizzare un dato codice e correggere eventuali errori di accessibilità.

Verranno risolti due diversi use case: il primo si pone l'obiettivo di segnalare in tempo reale allo sviluppatore eventuali errori di accessibilità analizzando il codice staticamente tramite ESLint e, su richiesta, di avere una risoluzione della problematica tramite l'LLM; il secondo permette allo sviluppatore di analizzare un dato codice alla ricerca di eventuali errori di accessibilità non segnalati¹ e di ricevere delle soluzioni elaborate da un LLM.

A seguito di un iniziale test dell'estensione, i cui risultati saranno descritti nel capitolo 5, è emersa la necessità di modificare parte della struttura dello use case che utilizza l'LLM per individuare e correggere gli errori di accessibilità. Questa revisione si è resa necessaria poiché i test iniziali non hanno prodotto un tasso di successo soddisfacente². All'interno di questo capitolo verranno riportate entrambe le versioni dello use case.

4.1 Architettura

L'architettura dell'estensione è composta da **ESLint** per identificare staticamente gli errori di accessibilità e da un **LLM** per l'analisi e la risoluzione degli errori di accessibilità. Ciò comporta che, per utilizzare le funzionalità di ESLint, esso dovrà essere configurato nel progetto front-end³.

Il plugin supporta l'analisi di codice scritto in Javascript, Javascript per React (JSX), TypeScript e TypeScript per React.

In figura 4.1 si riporta l'architettura dell'applicazione. In essa si ha anche il *System storage* in quanto lo sviluppatore potrà salvare nel proprio computer un file contenente il risultato generato dall'estensione.

¹L'analisi potrà essere effettuata senza l'obbligo di avere ESLint configurato nel progetto.

²Su 25 test, il 92% sono falliti.

³Si suggerisce di utilizzare il plugin *eslint-plugin-jsx-a11y* specifico per la rilevazione di errori di accessibilità nelle applicazioni React.

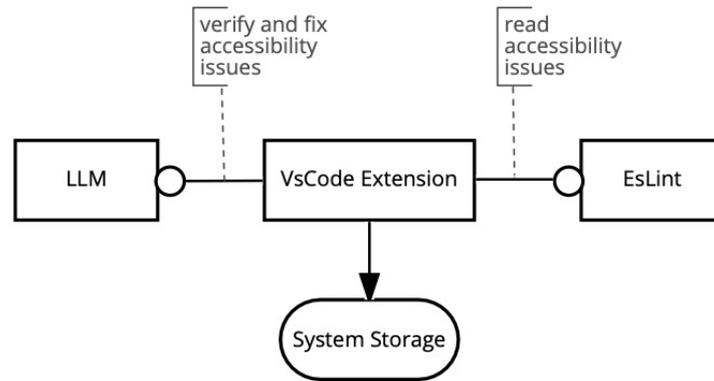


Figura 4.1: Architettura dell'estensione

4.2 Use Case e Requisiti

I casi d'uso definiscono un *attore* che interagisce con il *sistema*, come l'attore può interagire con il sistema e il comportamento del sistema stesso, ovvero come questo reagisce alla interazione dell'attore. Per ogni caso d'uso si hanno le precondizioni⁴, la sequenza degli eventi di cui è composto il caso d'uso e le postcondizioni⁵.

Per lo scopo di questo elaborato sono stati identificati due use case, denominati **FixWithAI** e **CheckAndFixWithAI**, che rappresentano due modalità attraverso le quali gli sviluppatori potranno identificare e risolvere problemi di accessibilità nel codice. **FixWithAI** si basa sull'identificazione automatica degli errori di accessibilità tramite ESLint e la loro risoluzione grazie all'LLM. Attraverso **CheckAndFixWithAI**, invece, lo sviluppatore potrà arbitrariamente verificare un blocco di codice e ricevere suggerimenti su eventuali errori di accessibilità.

In figura 4.2 si ha la rappresentazione degli use case dell'estensione.

4.2.1 Use Case 1 - FixWithAI

- Attori: Sviluppatore;
- Precondizioni: L'LLM scelto deve essere disponibile sulla macchina e avviato; ESLint deve essere installato nel progetto e configurato per identificare gli errori di accessibilità; l'estensione per VS Code deve essere installata e attiva;
- Sequenza degli eventi:
 1. Lo sviluppatore apre un file Javascript dove sono presenti degli errori di accessibilità;
 2. ESLint segnala gli errori di accessibilità rilevati staticamente;
 3. Lo sviluppatore fa l'hover su una segnalazione di ESLint, apre le correzioni rapide e seleziona l'opzione "Get fix suggestion from AI";

⁴Condizioni che devono essere vere per far sì che il caso d'uso possa essere eseguito.

⁵Condizioni che devono essere vere dopo l'esecuzione del caso d'uso.

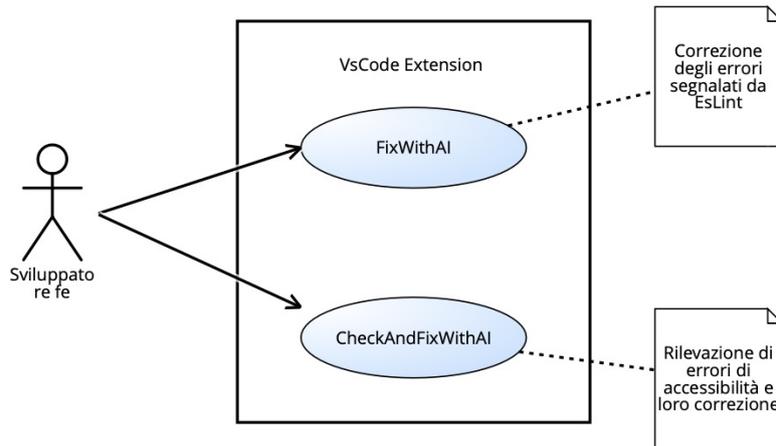


Figura 4.2: Diagramma degli use case

4. Il plugin ricava da ESLint la descrizione dell'errore di accessibilità e il codice che lo genera, interroga l'LLM per generare delle risoluzioni, restituisce la risposta allo sviluppatore:
 - (a) genera un file txt riportante la risposta dell'LLM. Le correzioni riportate sono sotto forma di JSON;
 - (b) se il JSON restituito dall'LLM è valido, in append al codice segnalato da ESLint viene restituita la risoluzione con il formato

```

Se numero di errori > 1 && Errore {n}
Problematica:
descrizione della problematica riscontrata
Risoluzione:
descrizione delle risoluzioni che si possono perseguire
Codice fixato:
codice con una delle fix descritte nella risoluzione
Se numero di errori > 1 && !ultimo errore nella lista && -----
    
```

- Postcondizioni: Lo sviluppatore può
 - visionare le correzioni nel file generato;
 - visionare le correzioni nel contesto del codice.

In figura 4.3 si ha la rappresentazione del BPMN (Business Process Model and Notation) dello use case.

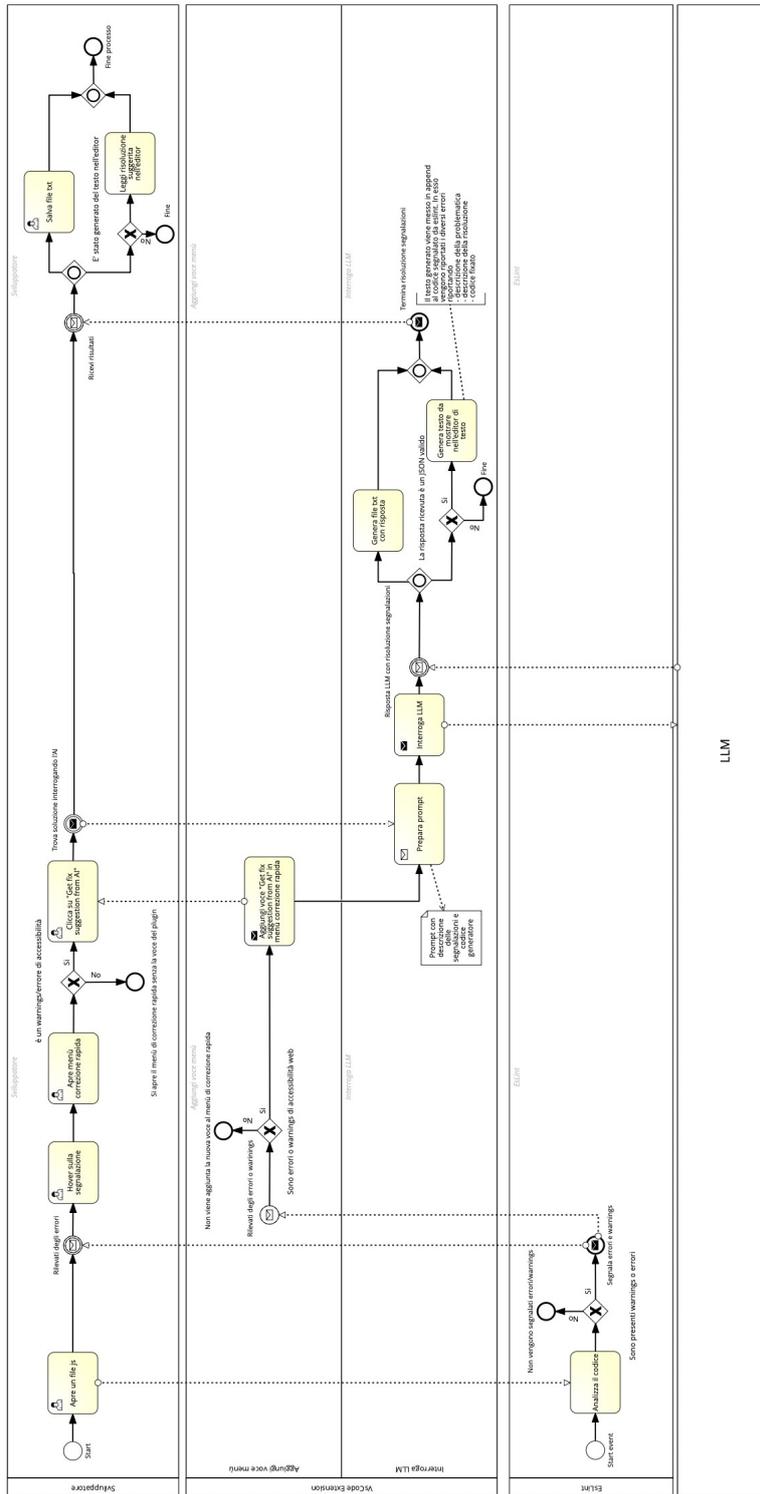


Figura 4.3: BPMN dello use case FixWithAI

4.2.2 Use Case 2 - CheckAndFixWithAI

Prototipo 1

Questo prototipo rappresenta la versione iniziale dell'estensione, successivamente modificata.

- Attori: Sviluppatore;
- Precondizioni: L'LLM scelto deve essere disponibile sulla macchina e avviato; l'estensione per VS Code deve essere installata e attiva;
- Sequenza degli eventi:
 1. Lo sviluppatore apre un file Javascript e seleziona un blocco di codice;
 - (a) tramite tasto destro, accede al menù rapido dove seleziona l'opzione "Check and fix accessibility issues with AI";
 - (b) dal command pallet accede al menù dei comandi, cerca e seleziona l'opzione "Check and fix accessibility issues with AI";
 2. Il plugin effettua due interrogazioni all'LLM:
 - (a) la prima interrogazione riceve il blocco di codice selezionato dallo sviluppatore e rileva gli errori di accessibilità presenti, che restituisce tramite JSON;
 - (b) la seconda interrogazione riceve in input il JSON generato dalla prima interrogazione e ne genera uno nuovo completo di fix per gli errori di accessibilità riscontrati;

Il sistema restituisce un file txt contenente il codice analizzato, la risposta alla prima interrogazione contenente solo gli errori riscontrati e la risposta alla seconda interrogazione contenente la risoluzione alle problematiche riscontrate;
- Postcondizioni: Lo sviluppatore può visionare gli errori e le correzioni nel file generato.

In figura 4.4 si ha la rappresentazione del BPMN (Business Process Model and Notation) dello use case.

Prototipo 2

Questo prototipo è la versione definitiva utilizzata per soddisfare lo use case **CheckAnd-FixWithAI**. In grassetto verranno riportati gli step che differiscono dal primo prototipo 4.2.2.

- Attori: Sviluppatore;
- Precondizioni: L'LLM scelto deve essere disponibile sulla macchina e avviato; l'estensione per VS Code deve essere installata e attiva;
- Sequenza degli eventi:

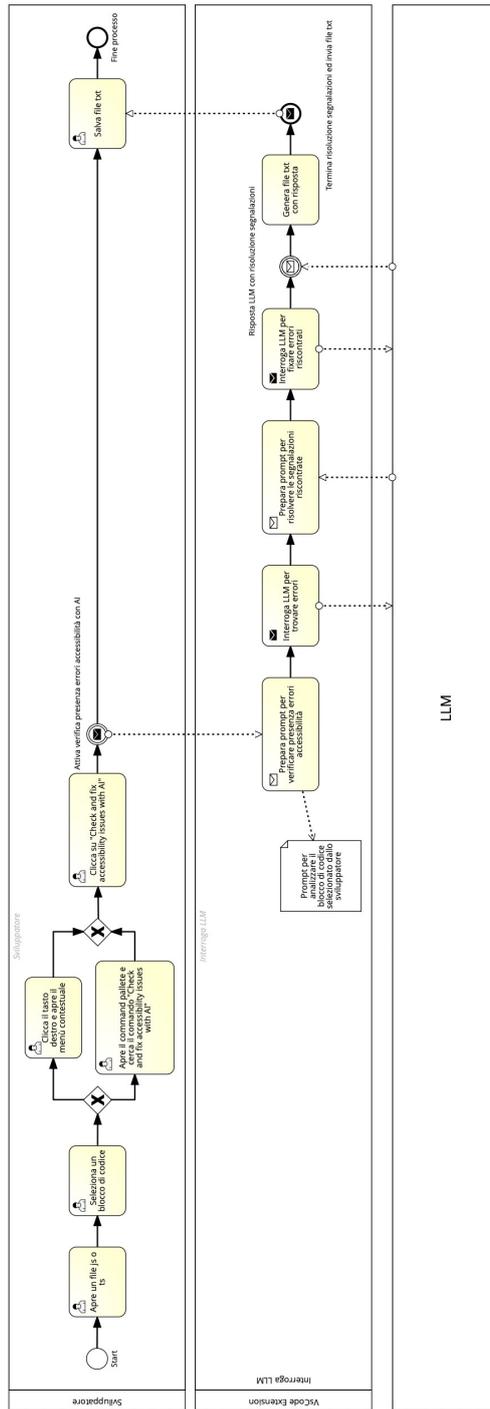


Figura 4.4: BPMN dello use case CheckAndFixWithAI - primo prototipo

1. Lo sviluppatore apre un file Javascript e seleziona un blocco di codice;
 - (a) tramite tasto destro, accede al menù rapido dove seleziona l'opzione "Check and fix accessibility issues with AI";
 - (b) dal command pallett accede al menù dei comandi, cerca e seleziona l'opzione "Check and fix accessibility issues with AI";
2. Il plugin effettua **tre** interrogazioni all'LLM:
 - (a) la prima interrogazione riceve il blocco di codice selezionato dallo sviluppatore e rileva gli errori di accessibilità presenti, che restituisce tramite **una lista con il seguente formato:**
 Errore {n}
 Descrizione: breve descrizione della problematica riscontrata
 Codice generatore: codice che genera l'errore di accessibilità
 Criterio: criterio WCAG violato
 - (b) la **seconda** interrogazione riceve in input la lista generata dalla prima interrogazione e la converte in un JSON;
 - (c) la **terza** interrogazione riceve in input il JSON generato dalla **seconda** interrogazione e ne genera uno nuovo completo di fix per gli errori di accessibilità riscontrati;

Il sistema restituisce un file txt contenente il codice analizzato, la risposta alla prima interrogazione contenente solo gli errori riscontrati **sotto forma di lista**, la **risposta della seconda interrogazione riportante la lista convertita in JSON** e la risposta alla **terza** interrogazione contenente la risoluzione alle problematiche riscontrate;

- Postcondizioni: Lo sviluppatore può visionare gli errori e le correzioni nel file generato.

In figura 4.5 si ha la rappresentazione del BPMN (Business Process Model and Notation) dello use case.

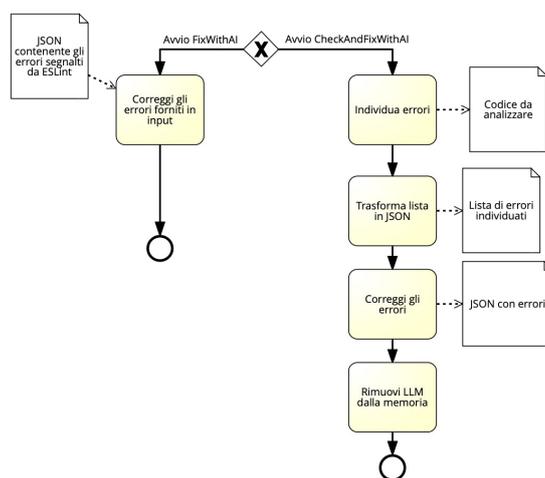


Figura 4.6: Sequenza delle interrogazioni all'LLM per i due use case

4.3 Prototipo

Per la realizzazione dell'estensione di VS Code si è dapprima provveduto a identificare l'LLM e i prompt da utilizzare. Per questo scopo si è deciso di utilizzare PromptMixer, un'applicazione che permette di confrontare l'output di diversi LLM, sia open source che non, al variare dei prompt⁶. Si è quindi provveduto a confrontare gli LLM Llama 2, Llama 3, Code Llama. E' stato inoltre considerato anche il modello CodeGemma. Il suo confronto con gli altri 3 LLM è stato effettuato manualmente passando le varie versioni dei prompt, in quanto PromptMixer non lo supporta.

Da questa analisi l'LLM che ha generato risultati migliori in termini di identificazione di errori di accessibilità e di risoluzioni di essi è stato Code Llama. CodeGemma ha generato risultati comparabili a Code Llama; per lo scopo di questo elaborato, si è preferito utilizzare Code Llama.

I prompt utilizzati nei due use case sono riportati in appendice A. Per lo use case CheckAndFixWithAI vengono riportati i prompt sia del primo prototipo che del secondo.

In figura 4.6 è riportata la sequenza delle interrogazioni all'LLM che avviene quando si sceglie di eseguire FixWithAI o CheckAndFixWithAI.

4.3.1 Implementazione use case CheckAndFixWithAI

All'interno del package.json del plugin è stato definito il comando *checkAndFixWithAi*, che avvia lo use case *CheckAndFixWithAI* 4.2.2. La sua attivazione, sia nel commandPalette che nel editor/context, è stata limitata ai file scritti in Javascript e TypeScript e solo nel caso in cui ci fosse una selezione (figura 4.7).

```
main : ../extension.js ,
"contributes": {
  "commands": [
    {
      "command": "accessibility-with-ai.checkAndFixWithAi",
      "title": "Check and fix accessibility issues with AI"
    }
  ],
  "menus": {
    "commandPalette": [
      {
        "command": "accessibility-with-ai.checkAndFixWithAi",
        "when": "(editorLangId == javascript || editorLangId == typescript) && editorHasSelection"
      }
    ],
    "editor/context": [
      {
        "command": "accessibility-with-ai.checkAndFixWithAi",
        "when": "(editorLangId == javascript || editorLangId == typescript) && editorHasSelection"
      }
    ]
  }
},
```

Figura 4.7: File package.json dell'applicazione dove viene definito il comando *checkAndFixWithAi*

Questo tipo di definizione fa sì che il comando venga mostrato nel menù contestuale (figura 4.8) e nel command palette (figura 4.9) solo se lo sviluppatore si trova in un file Javascript o TypeScript e solo se ha evidenziato un blocco di codice.

⁶Permette di creare delle versioni di prompt, così da poter tornare a una versione precedente senza difficoltà.

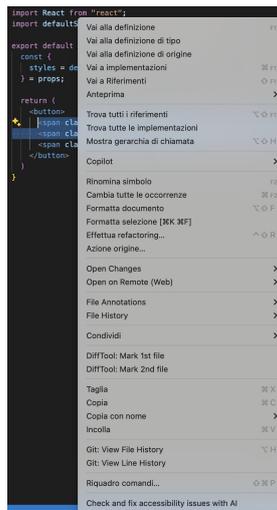


Figura 4.8: Comando presente nel menù contestuale solo se lo sviluppatore si trova su un file Javascript o TypeScript e ha evidenziato un blocco di codice

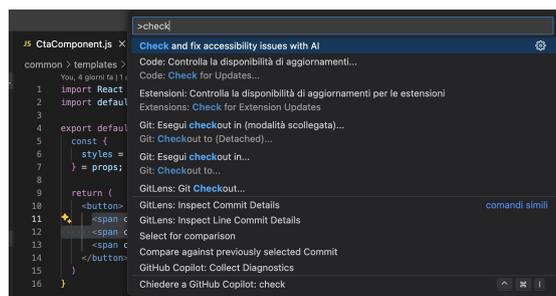


Figura 4.9: Comando presente nel command palette solo se lo sviluppatore si trova su un file Javascript o TypeScript e ha evidenziato un blocco di codice

Quando lo sviluppatore avvia il comando, nella prima versione dell'estensione veniva richiamata la funzione di figura 4.10. Essa effettua due chiamate in cascata all'LLM, la prima per analizzare il codice ed identificare eventuali errori di accessibilità, la seconda per correggere gli errori identificati dalla prima interrogazione. Quando entrambe le interrogazioni sono terminate, viene creato un file di testo contenente il codice analizzato e le risposte di entrambe le interrogazioni.

Nel secondo prototipo la funzione è stata modificata come mostrato in figura 4.11. In particolare la prima interrogazione restituisce una lista di errori. Essa poi viene convertita in JSON tramite una interrogazione intermedia. Successivamente il JSON contenente gli errori viene fornito in input a una terza interrogazione che analizza gli errori di accessibilità e fornisce la loro risoluzione. In ultimo è stata aggiunta una chiamata che permette di rimuovere l'LLM dalla memoria. All'interno delle modifiche apportate al secondo prototipo vi è anche la riscrittura dei prompt utilizzati nell'interrogazione per identificare gli errori di accessibilità e per la loro correzione. In particolare, il prompt per l'identificazione degli errori di accessibilità è stato arricchito di 4 esempi di codice-lista degli errori identificati, così da aiutare l'LLM ad identificare diverse tipologie di errori; anche il prompt per la correzione degli errori di accessibilità è stato

```
const checkAndFixWithAi = vscode.commands.registerCommand('accessibility-with-ai.checkAndFixWithAi', function () {
  const editor = vscode.window.activeTextEditor;
  if (editor) {
    const selection = editor.selection;
    const text = editor.document.getText(selection);
    const getResponse = async () => {
      vscode.window.withProgress({
        location: vscode.ProgressLocation.Notification,
        title: "Chiamata al LLM in corso...",
        cancellable: false
      }, async (progress) => {
        progress.report({ increment: 0 });
        try {
          const response = await ollama.default.chat({
            model: model,
            messages: [
              {
                role: "system",
                content: SYSTEM_PROMPT,
                // You, 4 settimane fa • adding prompts for fix with ai version 1
              },
              {
                role: "user",
                content: PROMPT_1_FIND_AND_FIX.replace("{{ codice }}", text)
              }
            ],
            stream: false,
          })

          const response2 = await ollama.default.chat({
            model: model,
            messages: [
              {
                role: "system",
                content: SYSTEM_PROMPT
              },
              {
                role: "user",
                content: PROMPT_2_FIND_AND_FIX.replace("{{ json }}", response.message.content)
              }
            ],
            stream: false,
          })

          const llmResponse = `
          \ncode:
          \n${text}
          \n-----
          \nerrors:
          \n${response.message.content}
          \n-----
          \nfix:
          ${response2.message.content}
          `;
          const doc = await vscode.workspace.openTextDocument({ content: llmResponse });
          await vscode.window.showTextDocument(doc, { preview: false });

        } catch (error) {
          vscode.window.showErrorMessage('Errore nella chiamata al LLM: ${error.message}');
        }
      });
    }
  }
  getResponse();
})
```

Figura 4.10: Interrogazione dell'LLM per lo use case CheckAndFixWithAI - primo prototipo. Vengono effettuate due chiamate in cascata, la prima per identificare la presenza di errori di accessibilità, la seconda per correggerli. In ultimo viene generato un file txt contenente il codice analizzato e le risposte delle due interrogazioni effettuate all'LLM

arricchito con 4 esempi di JSON contenente gli errori di accessibilità-JSON contenente la soluzione agli errori forniti.

```

const checkAndFixWithAi = vscode.commands.registerCommand('accessibility-with-ai.checkAndFixWithAi', function () {
  const editor = vscode.window.activeTextEditor;
  if (editor) {
    const selection = editor.selection;
    const text = editor.document.getText(selection);
    const getResponse = async () => {
      vscode.window.withProgress({
        location: vscode.ProgressLocation.Notification,
        title: "Chiamata al LLM in corso...",
        cancellable: false
      }, async (progress) => {
        progress.report({ increment: 0 });
        try {
          const response = await ollama.default.chat({
            model: model,
            messages: [
              {
                role: "system",
                content: SYSTEM_PROMPT,
              },
              {
                role: "user",
                content: prompt1_ves2.replace("{{ codice }}", text)
              }
            ],
            stream: false,
            keep_alive: 0
          })

          const response1_json = await ollama.default.chat({
            model: model,
            messages: [
              {
                role: "system",
                content: SYSTEM_PROMPT_JSON,
              },
              {
                role: "user",
                content: prompt2_ves2.replace("{{ lista }}", response.message.content)
              }
            ],
            stream: false,
            keep_alive: 0
          })

          const response2 = await ollama.default.chat({
            model: model,
            messages: [
              {
                role: "system",
                content: SYSTEM_PROMPT
              },
              {
                role: "user",
                content: prompt8.replace("{{ json }}", response1_json.message.content)
              }
            ],
            stream: false,
            keep_alive: 0
          })

          const resetModel1 = await ollama.default.chat({
            model: model,
            messages: [],
            stream: false,
            keep_alive: 0
          })

          const llmResponse = `
          \ncode:      You, 3 mesi fa • update prompts
          \n${text}
          \n-----
          \nerrors:
          \n${response.message.content}
          \n-----
          \njson errors:
          \n${response1_json.message.content}
          \n-----
          \nfix:
          \n${response2.message.content}
          `;
          const doc = await vscode.workspace.openTextDocument({ content: llmResponse });
          await vscode.window.showTextDocument(doc, { preview: false });

          catch (error) {
            vscode.window.showErrorMessage(`Errore nella chiamata al LLM: ${error.message}`);
          }
        }
      });
    };
  }
});

```

Figura 4.11: Interrogazione dell'LLM per lo use case CheckAndFixWithAI - secondo prototipo. Vengono effettuate tre chiamate in cascata, la prima per identificare la presenza di errori di accessibilità. Essa restituisce una lista di errori identificati; la seconda converte la lista in un JSON; la terza corregge gli errori ricevuti in input. Successivamente l'LLM viene rimosso dalla memoria. In ultimo viene generato un file txt contenente il codice analizzato e le risposte delle tre interrogazioni effettuate all'LLM

4.3.2 Implementazione use case FixWithAI

Per lo use case *FixWithAI* 4.2.1, poichè la voce di menù deve comparire nel menù di correzioni rapide solo quando la segnalazione di ESLint riguarda un problema di accessibilità, il comando *fixAccessibilityIssue* è stato inserito nei diversi menù di correzione rapida dinamicamente.

Nei file scritti in Javascript o TypeScript (React oppure no) si prendono tutte le segnalazioni⁷ effettuate da ESLint. Se le descrizioni di queste includono almeno una tra le parole⁸ identificate come rappresentative di segnalazione di accessibilità o è inclusa la stringa `jsx-a11y`⁹, allora il comando *fixAccessibilityIssue* viene aggiunto al menù delle correzioni rapide. Il codice è riportato in figura 4.12.

```
const checkErrors = vscode.languages.registerCodeActionsProvider(
  // enable the provider in file with the following language ids
  ['javascript', 'javascriptreact', 'typescript', 'typescriptreact'], {
  provideCodeActions(document, range, context, token) {
    const codeActions = [];

    // Find all the diagnostics related to accessibility
    const accessibilityDiagnostics = context.diagnostics.filter(diagnostic => {
      if (diagnostic.source === 'eslint') {
        if (diagnostic.code.value.includes('jsx-a11y')) {
          return true;
        }
        if (accessibilityKeywords.some(keyword => diagnostic.message.toLowerCase().includes(keyword.toLowerCase()))) {
          return true;
        }
      }
      return false;
    });

    if (accessibilityDiagnostics.length > 0) {
      // Create a code action to fix the accessibility issue. This will be displayed in the menu to fix errors that eslint finds
      const fixAccessibilityAction = new vscode.CodeAction('Get fix suggestion from AI', vscode.CodeActionKind.QuickFix);
      fixAccessibilityAction.command = {
        title: 'Get fix suggestion from AI',
        command: 'accessibility-with-ai.fixAccessibilityIssue',
        arguments: [document, accessibilityDiagnostics]
      };

      codeActions.push(fixAccessibilityAction);
    }

    return codeActions;
  }
});
```

Figura 4.12: Codice per creare il comando *fixAccessibilityIssue*

La nuova voce di menù viene mostrata all'interno del menù di correzione rapida generato da ESLint (figura 4.13).

⁷Sia errori che warnings.

⁸aria, role, contrast, focus, keyboard, accessibility, a11y, alt.

⁹Stringa che rappresenta l'estensione di ESLint per la verifica degli errori di accessibilità di codice scritto in Javascript e con la sintassi di React.

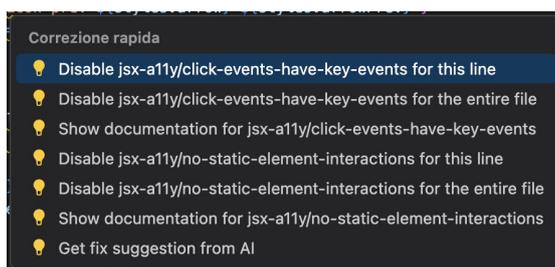


Figura 4.13: Menù di correzione rapida generato da eslint che presenta la voce "Get fix suggestion from AI"

Cliccando sulla voce "Get fix suggestion from AI" viene chiamata la funzione *fixAccessibilityIssue* (figura 4.14). Essa riceve gli errori e segnalazioni inerenti alla data istanza¹⁰, nonché il codice che ha creato la segnalazione che vengono passati, sotto forma di JSON, all'LLM che restituisce le dovute correzioni, sempre sotto forma di JSON.

```
const fixAccessibilityIssue = vscode.commands.registerCommand('accessibility-with-ai.fixAccessibilityIssue', function (document, diagnostic) {
  const editor = vscode.window.activeTextEditor;
  if (editor) {
    const errorList = [];
    diagnostic.forEach(error => {
      errorList.push({
        descrizione_errore: error.message,
        codice_generatore: document.getText(error.range)
      });
    });
    const getResponse = async () => {
      vscode.window.withProgress({
        location: vscode.ProgressLocation.Notification,
        title: "Chiamata al LLM in corso...",
        cancellable: false
      }, async (progress) => {
        progress.report({ increment: 8 });
      });
    };
    const response = await ollama.default.chat({
      model: model,
      messages: [
        {
          role: "system",
          content: SYSTEM_PROMPT
        },
        {
          role: "user",
          content: PROMPT_FIX.replace('{{ json }}', JSON.stringify(errorList))
        }
      ],
      stream: false
    });
    const llmResponse = response.message.content;
    const formattedResponse = formatLlmResponseForEditor(llmResponse);
    if (typeof formattedResponse === "string" && formattedResponse.length > 0) {
      editor.edit(editBuilder => {
        editBuilder.insert(diagnostic[0].range.end, formattedResponseForEditor(llmResponse));
        progress.report({ increment: 88 });
      });
    }
    const doc = await vscode.workspace.openTextDocument({ content: llmResponse });
    await vscode.window.showTextDocument(doc, { preview: false });
  } catch (error) {
    vscode.window.showErrorMessage("Errore nella chiamata al LLM: " + error.message);
  }
});
if (errorList.length > 0) {
  getResponse();
}
```

Figura 4.14: Interrogazione dell'LLM per lo use case FixWithAI. La funzione riceve le segnalazioni fatte da ESLint e il codice che genera l'errore e li passa, sotto forma di JSON, all'LLM che restituisce un array con le dovute correzioni

Se la risposta dell'LLM è un JSON valido, essa viene formattata e inserita, come commento, all'interno dell'editor così che lo sviluppatore possa vedere le diverse soluzioni, per i diversi problemi segnalati, all'interno del contesto del codice. In particolare viene restituita la descrizione della problematica, la descrizione della risoluzione e il codice corretto (figura 4.15). In fine viene generato un file txt contenente il JSON restituito dall'LLM.

¹⁰Le segnalazioni che vengono propagate alla funzione sono contenute nella costante **accessibilityDiagnostics** dichiarata in fase di creazione della voce di menù in figura 4.12.

```
function formatLlmResponseForEditor(response) {
  let responseParsed = response;
  let newResponse = "";
  try {
    if (typeof response === "string") {
      responseParsed = JSON.parse(response);
    }
    newResponse = responseParsed.map((error, index) => (
      `${responseParsed.length > 1 ? `Error ${index + 1}` : ""}
      Problematica:
      ${error.descrizione_errore}
      Risoluzione:
      ${error.descrizione_risoluzione}
      Codice fixato:
      ${error.codice_fix}
      ${responseParsed.length > 1 && index + 1 !== responseParsed.length? "-----" : ""}
    )).join("\n") || "";

    if (newResponse.length > 0) {
      newResponse = `
      /*
      ${newResponse}
      */
      `;
    }
  } catch (error) {
    console.log("error", error);
  }
  return newResponse;
}
```

Figura 4.15: Funzione che formatta la risposta dell'LLM che verrà mostrata allo sviluppatore all'interno del file dove si ha il codice contenente la segnalazione. La risposta contiene la descrizione della problematica, la descrizione delle possibili risoluzioni, nonché il codice corretto

Capitolo 5

Validazione

La fase di validazione è uno step fondamentale per verificare se l'estensione soddisfa gli obiettivi del progetto, ovvero assistere gli sviluppatori front-end nella scrittura di codice accessibile. In questo capitolo vengono delineati i criteri di valutazione adottati e analizzati i risultati ottenuti, valutando la capacità dell'LLM Code Llama nell'identificare problematiche di accessibilità e nel proporre soluzioni corrette e complete.

I test sono stati condotti su una repository utilizzata per lo sviluppo di un sito e-commerce. Il progetto utilizza Next.js, React e un CMS headless¹ proprietario per la gestione dei contenuti editoriali. Complessivamente sono stati condotti 240 test, 120 per use case, su altrettanti snippet di codice e i risultati sono stati classificati in tre categorie qualitative e suddivisi per use case.

5.1 Criteri di classificazione dei risultati

I risultati ottenuti durante la fase di test sono stati classificati in tre categorie (corretto, parzialmente corretto e non corretto) che rappresentano la qualità e l'utilità della risposta dell'LLM.

Il metodo che è stato utilizzato per assegnare la categoria al risultato prevede di verificare se il risultato rientra dentro la categoria di "non corretto". Se non rientra in questa categoria, allora, nell'ordine, si verifica l'appartenenza alla categoria "corretto" o "parzialmente corretto". La classificazione si sviluppa attraverso i seguenti passaggi:

1. Si verifica la presenza di criteri valutati come "**non corretto**" o "**non valutabile**", ovvero non è possibile indicarlo ne come "corretto", ne come "parzialmente corretto", ne come "non corretto". Se almeno un criterio è valutato come tale, il test è assegnato alla categoria "**non corretto**";
2. Si verifica la percentuale di criteri classificati come "**corretto**". Se almeno l'80% dei criteri è "corretto" allora il test è "**corretto**";
3. Se almeno l'80% dei criteri è classificato come "**parzialmente corretto**", il test, sebbene con delle imperfezioni, rappresenta comunque un buon livello di risposta. Per questo viene classificato come "**corretto**";

¹In un CMS headless la gestione del frontend e backend sono separate, ovvero il CMS fornisce solo il dato da mostrare e non stabilisce la struttura HTML dell'elemento.

4. Se il test non è stato classificato ne come "non corretto", ne come "corretto", allora ha meno dell'80% dei criteri classificati come "parzialmente corretto" o "corretto" e nessuno come "non corretto" o "non classificabile". Quindi il test viene assegnato al livello **"parzialmente corretto"**.

In figura 5.1 è riportato il processo decisionale.

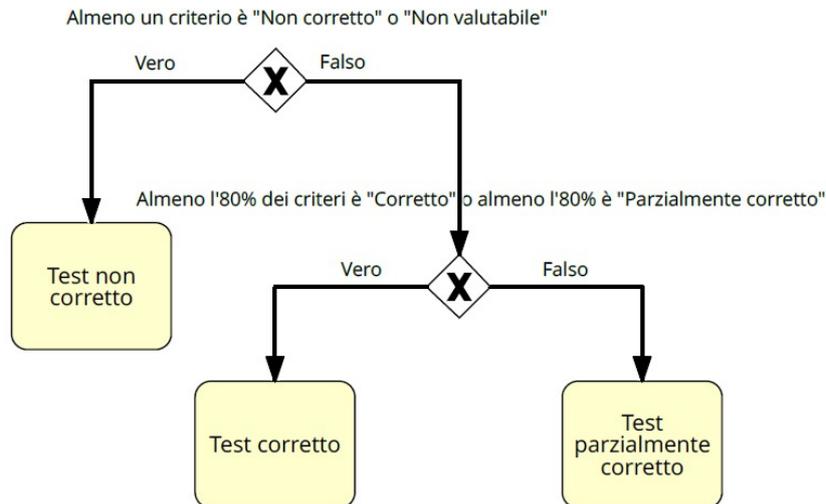


Figura 5.1: Processo decisionale per la classificazione dei test. Se almeno un criterio è "Non corretto" o "Non valutabile" il test viene classificato come non corretto. Se almeno l'80% dei criteri è "Corretto" o "Parzialmente corretto", il test viene considerato corretto. Se meno dell'80% dei criteri è corretto o parzialmente corretto, il test è classificato come parzialmente corretto

Ogni use case, però, necessita di criteri specifici. Per FixWithAI sono stati definiti 10 criteri, quindi la soglia dell'80% è rappresentata da 8 criteri. Per CheckAndFixWithAI sono stati definiti 15 criteri, quindi la soglia dell'80% equivale a 12 criteri.

5.1.1 Use case FixWithAI

Di seguito vengono descritte le varie categorie con relativi criteri di successo per lo use case FixWithAI.

- **Non corretto:**

- **Formattazione del JSON e leggibilità:** la risposta restituita dall'LLM non è un JSON oppure è un JSON non correttamente formattato, quindi non è possibile mostrare il risultato dell'interrogazione direttamente dentro l'editor di testo. All'interno del file txt la risposta è prolissa o ha un formato difficile da leggere e comprendere²;
- **Problemi di accessibilità non risolti:** la soluzione proposta non risolve alcun problema di accessibilità;

²Le soluzioni non sono formattate attraverso una lista, oppure il testo che viene restituito è troppo lungo, i suggerimenti di correzione non sono facilmente individuabili.

- **Introduzione di problemi di accessibilità:** la soluzione proposta introduce problemi di accessibilità;
 - **Introduzione di errori di sintassi o di logica:** la soluzione proposta introduce errori di sintassi o di logica;
 - **Presenza di campi non ammessi nel JSON:** l'oggetto è composto da campi diversi da *descrizione_errore*, *codice_generatore*, *descrizione_risoluzione*, *codice_fix*³;
 - **Campi JSON non compilati:** almeno uno tra i campi *descrizione_errore*, *descrizione_risoluzione* e *codice_fix* non è compilato correttamente;
 - **Conformità alla versione WCAG:** la soluzione proposta non segue le WCAG oppure segue una versione antecedente alla 2.1;
 - **Conformità al livello WCAG:** la soluzione proposta non rispetta il livello AA delle WCAG⁴;
 - **Integrità del codice:** il codice fixato non riporta tutte le parti di codice già presenti⁵;
 - **Numero di errori corretti rispetto al codice sorgente:** il 50% + 1 degli errori riportati nella lista non è pertinente al codice fornito in input;
- **Parzialmente corretto:**
 - **Formattazione del JSON e leggibilità:** la risposta restituita dall'LLM non è un JSON oppure è un JSON non correttamente formattato, quindi non è possibile mostrare il risultato dell'interrogazione direttamente dentro l'editor di testo. E' comunque possibile visionarlo all'interno del file txt e ha un formato di facile lettura⁶;
 - **Problemi di accessibilità non risolti:** la soluzione proposta risolve alcune problematiche di accessibilità ma non tutte, è necessario quindi un intervento dello sviluppatore per rendere la soluzione pienamente conforme ai requisiti di accessibilità;
 - **Introduzione di problemi di accessibilità:** la soluzione proposta non genera ulteriori problematiche di accessibilità;
 - **Introduzione di errori di sintassi o di logica:** la soluzione proposta non introduce errori di sintassi o di logica;
 - **Presenza di campi non ammessi nel JSON:** l'oggetto è composto solo dai campi *descrizione_errore*, *codice_generatore*, *descrizione_risoluzione*, *codice_fix* o al più non è presente solo il campo *codice_generatore*;
 - **Campi JSON non compilati:** solo il campo *codice_generatore* non è compilato;

³L'LLM deve restituire un JSON contenente un array di oggetti. Questi oggetti sono composti dai campi *descrizione_errore*, *codice_generatore*, *descrizione_risoluzione*, *codice_fix*.

⁴Il livello AA è ritenuto il livello minimo richiesto per garantire un livello di accessibilità base affinché tutti possano accedere al contenuto digitale. Inoltre, sia in Italia con la Legge Stanca che negli Stati Uniti con l'ADA (Americans with Disabilities Act), il livello AA è il livello richiesto per ritenere un sito conforme alla legge e quindi accessibile.

⁵Ad esempio, il codice che viene passato all'LLM ha un div (con un onclick) che contiene uno span, l'LLM non restituisce lo span contenuto nel div iniziale.

⁶Le risoluzioni sono formattate attraverso una lista, oppure il testo che viene restituito non è troppo lungo ed è facile individuare i suggerimenti di correzione.

- **Conformità alla versione WCAG:** la soluzione proposta segue le linee guida WCAG 2.1⁷;
 - **Conformità al livello WCAG:** la soluzione proposta rispetta il livello AA delle WCAG;
 - **Integrità del codice:** il codice fixato riporta tutte le parti di codice già presenti;
 - **Numero di errori corretti rispetto al codice sorgente:** almeno il 50% degli errori riportati nella lista e fino all'80% sono pertinenti al codice fornito in input;
- **Corretto:**
 - **Formattazione del JSON e leggibilità:** l'LLM ha fornito un JSON correttamente formattato. E' stato quindi possibile mostrare il risultato dell'interrogazione direttamente dentro l'editor di testo;
 - **Problemi di accessibilità non risolti:** la soluzione proposta dall'LLM è completa, ovvero risolve completamente i problemi di accessibilità;
 - **Presenza di campi non ammessi nel JSON:** tutti i campi del JSON sono compilati;
 - **Campi JSON non compilati:** tutti i campi del JSON sono compilati;
 - **Conformità alla versione WCAG:** la soluzione proposta segue le linee guida WCAG 2.2;
 - **Numero di errori corretti rispetto al codice sorgente:** più dell'80% degli errori riportati nella lista è pertinente al codice fornito in input;

Nella tabella 5.1 vengono riportati i vari criteri suddivisi per livelli di conformità.

5.1.2 Use case CheckAndFixWithAI

Di seguito vengono descritte le varie categorie con relativi criteri di successo per lo use case CheckAndFixWithAI.

- **Non corretto:**
 - **Formattazione del JSON e leggibilità:** la risposta restituita dall'LLM, nella prima o nella seconda interrogazione⁸, non è un JSON oppure è un JSON non correttamente formattato. All'interno del file txt la risposta è prolissa o ha un formato difficile da leggere e comprendere⁹;
 - **Problemi di accessibilità non risolti:** la soluzione proposta non risolve alcun problema di accessibilità;
 - **Introduzione di problemi di accessibilità:** la soluzione proposta introduce problemi di accessibilità;

⁷La prima versione delle WCAG è la 1.0. Negli anni si sono susseguiti degli aggiornamenti fino ad arrivare all'ultima versione 2.2. La versione 2.1 è, in Italia, al momento della scrittura di questo elaborato, quella a cui le norme fanno riferimento per poter ritenere un contenuto digitale accessibile.

⁸Nella prima interrogazione l'LLM identifica gli errori di accessibilità. Nella seconda, genera le risoluzioni alle problematiche riscontrate.

⁹Le soluzioni non sono formattate attraverso una lista, oppure il testo che viene restituito è troppo lungo, i suggerimenti di correzione non sono facilmente individuabili.

Tabella 5.1: Tabella rappresentante i criteri di successo suddivisi per livelli di conformità relativi allo use case FixWithAI

Criterio	Non corretto	Parzialmente corretto	Corretto
Formattazione del JSON e leggibilità	La risposta restituita dall'LLM non è un JSON oppure è un JSON non correttamente formattato, quindi non è possibile mostrare il risultato dell'interrogazione direttamente dentro l'editor di testo. All'interno del file txt la risposta è prolissa o ha un formato difficile da leggere e comprendere	La risposta restituita dall'LLM non è un JSON oppure è un JSON non correttamente formattato, quindi non è possibile mostrare il risultato dell'interrogazione direttamente dentro l'editor di testo. E' comunque possibile visionarlo all'interno del file txt e ha un formato di facile lettura	L'LLM ha fornito un JSON correttamente formattato. E' stato quindi possibile mostrare il risultato dell'interrogazione direttamente dentro l'editor di testo
Problemi di accessibilità non risolti	La soluzione proposta non risolve alcun problema di accessibilità	La soluzione proposta risolve alcune problematiche di accessibilità ma non tutte, è necessario quindi un intervento dello sviluppatore per rendere la soluzione pienamente conforme ai requisiti di accessibilità	La soluzione proposta dall'LLM è completa, ovvero risolve completamente i problemi di accessibilità
Introduzione di problemi di accessibilità	La soluzione proposta introduce problemi di accessibilità	La soluzione proposta non genera ulteriori problematiche di accessibilità	
Introduzione di errori di sintassi o di logica	La soluzione proposta introduce errori di sintassi o di logica	La soluzione proposta non introduce errori di sintassi o di logica	

Tabella 5.1: Tabella rappresentante i criteri di successo suddivisi per livelli di conformità relativi allo use case FixWithAI

Critério	Non corretto	Parzialmente corretto	Corretto
Presenza di campi non ammessi nel JSON	L'oggetto è composto da campi diversi da descrizione_errore, codice_generatore, descrizione_risoluzione, codice_fix	L'oggetto è composto solo dai campi descrizione_errore, codice_generatore, descrizione_risoluzione, codice_fix o al più non è presente solo il campo codice_generatore	Tutti i campi del JSON sono compilati
Campi JSON non compilati	Almeno uno tra i campi descrizione_errore, descrizione_risoluzione e codice_fix non è compilato correttamente	solo il campo codice_generatore non è compilato	Tutti i campi del JSON sono compilati
Conformità alla versione WCAG	La soluzione proposta non segue le WCAG oppure segue una versione antecedente alla 2.1	la soluzione proposta segue le linee guida WCAG 2.1	La soluzione proposta segue le linee guida WCAG 2.2
Conformità al livello WCAG	La soluzione proposta non rispetta il livello AA delle WCAG	La soluzione proposta rispetta il livello AA delle WCAG	
Integrità del codice	Il codice fixato non riporta tutte le parti di codice già presenti	Il codice fixato riporta tutte le parti di codice già presenti	
Numero di errori corretti rispetto al codice sorgente	Il 50% + 1 degli errori riportati nella lista non è pertinente al codice fornito in input	Almeno il 50% degli errori riportati nella lista e fino all'80% sono pertinenti al codice fornito in input	Più dell'80% degli errori riportati nella lista è pertinente al codice fornito in input

- **Introduzione di errori di sintassi o di logica:** la soluzione proposta introduce errori di sintassi o di logica;
- **Rilevazione errata di problematiche:** la soluzione proposta segnala come errore un pezzo di codice corretto;
- **Identificazione di problematiche non inerenti all'accessibilità:** nella segnalazione degli errori sono presenti errori non pertinenti all'accessibilità;

- **Presenza di campi non ammessi nel JSON:** l'oggetto è composto campi diversi da *descrizione_errore*, *codice_generatore*, *descrizione_risoluzione*, *codice_fix*, *criterio*¹⁰;
 - **Campi JSON non compilati:** almeno uno tra i campi *descrizione_errore*, *descrizione_risoluzione*, *codice_fix* non è compilato correttamente;
 - **Conformità alla versione WCAG:** la soluzione proposta non segue le WCAG oppure segue una versione antecedente alla 2.1;
 - **Conformità al livello WCAG:** la soluzione proposta non rispetta il livello AA delle WCAG;
 - **Integrità del codice:** il codice fixato non riporta tutte le parti di codice già presenti¹¹;
 - **Numero di errori corretti rispetto al codice sorgente:** il 50% + 1 degli errori riportati nella lista non è pertinente al codice fornito in input;
 - **Pertinenza contestuale delle soluzioni proposte:** la soluzione proposta non è adeguata al contesto e non è applicabile alla situazione specifica;
 - **Percentuale di rilevamento degli errori:** l'LLM non ha identificato almeno il 50% degli errori presenti nel blocco di codice analizzato;
- **Parzialmente corretto:**
- **Formattazione del JSON e leggibilità:** la risposta restituita dall'LLM, nella prima o nella seconda interrogazione¹², non è un JSON oppure è un JSON non correttamente formattato. E' comunque possibile visionarla all'interno del file txt e ha un formato di facile lettura¹³;
 - **Problemi di accessibilità non risolti:** la soluzione proposta risolve alcune problematiche di accessibilità ma non tutte, è necessario quindi un intervento dello sviluppatore per rendere la soluzione pienamente conforme ai requisiti di accessibilità;
 - **Introduzione di problemi di accessibilità:** la soluzione proposta non genera ulteriori problematiche di accessibilità;
 - **Introduzione di errori di sintassi o di logica:** la soluzione proposta non introduce errori di sintassi o di logica;
 - **Rilevazione errata di problematiche:** la soluzione proposta non segnala come errato codice corretto;
 - **Identificazione di problematiche non inerenti all'accessibilità:** non sono segnalati errori che non siano quelli di accessibilità;
 - **Presenza di campi non ammessi nel JSON:** l'oggetto è composto solo dai campi *descrizione_errore*, *codice_generatore*, *descrizione_risoluzione*, *codice_fix*, *criterio* o al più non sono presenti solo i campi *codice_generatore* e *criterio*;

¹⁰L'LLM deve restituire un JSON contenente un array di oggetti. Questi oggetti sono composti dai campi *descrizione_errore*, *codice_generatore*, *descrizione_risoluzione*, *codice_fix*, *criterio*.

¹¹Ad esempio, il codice che viene passato all'LLM ha un div (con un onclick) che contiene uno span, l'LLM non restituisce lo span contenuto nel div iniziale.

¹²Nella prima interrogazione l'LLM identifica gli errori di accessibilità. Nella seconda, genera le risoluzioni alle problematiche riscontrate.

¹³Le risoluzioni sono formattate attraverso una lista, oppure il testo che viene restituito non è troppo lungo ed è facile individuare i suggerimenti di correzione.

- **Campi JSON non compilati:** solo i campi *codice_generatore* e/o *criterio* non sono compilati;
 - **Conformità alla versione WCAG:** la soluzione proposta segue le linee guida WCAG 2.1¹⁴;
 - **Conformità al livello WCAG:** la soluzione proposta rispetta il livello AA delle WCAG;
 - **Integrità del codice:** il codice fixato riporta tutte le parti di codice già presenti;
 - **Numero di errori corretti rispetto al codice sorgente:** almeno il 50% degli errori riportati nella lista e fino all'80% sono pertinenti al codice fornito in input;
 - **Pertinenza contestuale delle soluzioni proposte:** la soluzione proposta non ha problemi di accessibilità ma non è ottimale per il contesto del blocco di codice evidenziato. Serve un intervento di uno sviluppatore per adattarla al contesto;
 - **Percentuale di rilevamento degli errori:** l'LLM ha identificato tra il 50% e l'80% degli errori presenti nel blocco di codice analizzato;
 - **Identificazione dei criteri WCAG:** il criterio WCAG identificato non esiste o non è coerente all'errore identificato;
- **Corretto:**
 - **Formattazione del JSON e leggibilità:** l'LLM, sia nella prima che nella seconda interrogazione¹⁵, ha fornito un JSON correttamente formattato;
 - **Problemi di accessibilità non risolti:** la soluzione proposta dall'LLM è completa, ovvero risolve completamente i problemi di accessibilità;
 - **Presenza di campi non ammessi nel JSON:** tutti i campi del JSON sono compilati;
 - **Campi JSON non compilati:** tutti i campi del JSON sono compilati;
 - **Conformità alla versione WCAG:** la soluzione proposta segue le linee guida WCAG 2.2;
 - **Numero di errori corretti rispetto al codice sorgente:** più dell'80% degli errori riportati nella lista è pertinente al codice fornito in input;
 - **Pertinenza contestuale delle soluzioni proposte:** la soluzione proposta è contestualmente adeguata, ovvero riconosce il contesto dell'elemento all'interno del blocco di codice evidenziato e suggerisce soluzioni pertinenti al contesto;
 - **Percentuale di rilevamento degli errori:** l'LLM ha identificato almeno l'80% degli errori presenti nel blocco di codice analizzato;
 - **Identificazione dei criteri WCAG:** il criterio WCAG identificato esiste ed è coerente all'errore identificato;

¹⁴La prima versione delle WCAG è la 1.0. Negli anni si sono susseguiti degli aggiornamenti fino ad arrivare all'ultima versione 2.2. La versione 2.1 è, in Italia, al momento della scrittura di questo elaborato, quella a cui le norme fanno riferimento per poter ritenere un contenuto digitale accessibile.

¹⁵Nella prima interrogazione l'LLM identifica gli errori di accessibilità. Nella seconda, genera le risoluzioni alle problematiche riscontrate.

Per soddisfare le necessità implementative del prototipo 2 4.2.2, i criteri di classificazione dei risultati per lo use case `CheckAndFixWithAI` hanno subito una piccola modifica. In particolare la modifica riguarda il primo criterio **Formattazione del JSON e leggibilità**. Esso è stato modificato così come segue:

- **Non corretto:** la risposta restituita dall'LLM, nella **seconda** o nella **terza** interrogazione, non è un JSON oppure è un JSON non correttamente formattato, oppure **la risposta alla prima interrogazione non è una lista e non rispetta il template di risposta**. All'interno del file txt la risposta è prolissa o ha un formato difficile da leggere e comprendere;
- **Parzialmente corretto:** la risposta restituita dall'LLM, nella **seconda** o nella **terza** interrogazione, non è un JSON oppure è un JSON non correttamente formattato, oppure **la risposta alla prima interrogazione non è una lista e non rispetta il template di risposta**. E' comunque possibile visionarla all'interno del file txt e ha un formato di facile lettura;
- **Corretto:** l'LLM, sia nella **seconda** che nella **terza** interrogazione, ha fornito un JSON correttamente formattato e **nella prima interrogazione ha restituito una lista conforme al template indicato**;

In tabella 5.2 si riportano i criteri suddivisi per livelli di conformità. Per semplicità è stata riportata la versione del secondo prototipo dello use case.

5.2 Risultati

Per valutare l'estensione per VS Code sono stati effettuati 120 test per use case. Per ogni use case, in una fase preliminare, sono stati condotti 25 test con l'obiettivo di verificare se i risultati ottenuti fossero in linea con le aspettative, oppure fosse necessario apportare delle correzioni. Durante questa fase, solo il 16% dei test effettuati per `FixWithAI` risultava "non corretto"; per questo motivo si è deciso di proseguire con i test senza apportare alcuna modifica a questo caso. Per `CheckAndFixWithAI`, invece, il 92% dei test risultava "non corretto"; per questo motivo è stato necessario rivedere la struttura ed i prompt utilizzati per questo use case, giungendo alla realizzazione del secondo prototipo.

I risultati finali di **FixWithAI** sono molto promettenti, con solo il 30% dei test falliti (classificati come "non corretti").

Per **CheckAndFixWithAI** il secondo prototipo ha registrato un tasso di fallimento del 60,83%, registrando un netto miglioramento rispetto al primo prototipo. Nel paragrafo 5.3 si analizzeranno nel dettaglio i risultati ottenuti.

Di seguito vengono presentati i risultati ottenuti divisi per use case, riportando sia i valori intermedi effettuati su 25 test, sia gli esiti dei due prototipi di `CheckAndFixWithAI`.

5.2.1 FixWithAI

I test sono stati condotti in due fasi distinte. Una prima fase, composta da un set di 25 test iniziali, è stata effettuata con l'obiettivo di verificare l'efficacia dell'estensione e valutare se fosse opportuno procedere con un'analisi più ampia. Successivamente, è stato eseguito un secondo set di 95 test - per un totale di 120, i cui risultati rappresentano la valutazione finale. Dati i buoni risultati dei test derivanti dalla prima fase, tra le due fasi non sono state apportate ottimizzazioni al sistema.

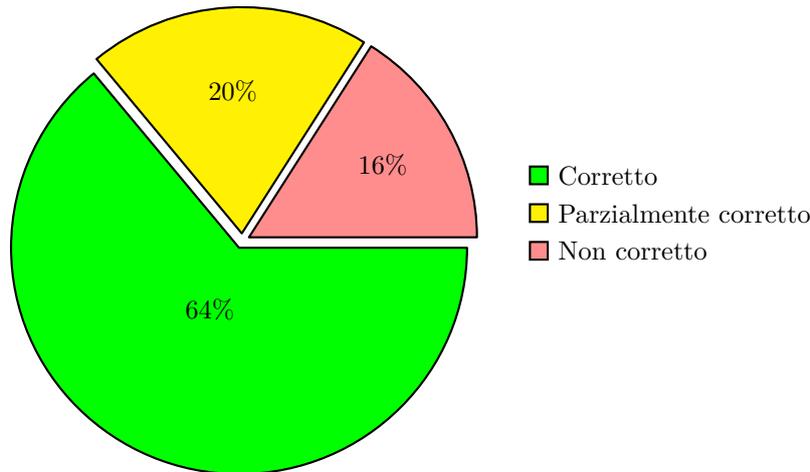


Figura 5.2: Grafico risultati preliminari FixWithAI su 25 test. 64% dei test è "Corretto", il 20% è "Parzialmente corretto", il 16% è "Non corretto"

Tabella 5.2: Tabella rappresentante i criteri di successo suddivisi per livelli di conformità relativi allo use case CheckAndFixWithAI

Criteriono	Non corretto	Parzialmente corretto	Corretto
Formattazione del JSON e leggibilità	La risposta restituita dall'LLM, nella seconda o nella terza interrogazione, non è un JSON oppure è un JSON non correttamente formattato, oppure la risposta alla prima interrogazione non è una lista e non rispetta il template di risposta. All'interno del file txt la risposta è prolissa o ha un formato difficile da leggere e comprendere	La risposta restituita dall'LLM, nella seconda o nella terza interrogazione, non è un JSON oppure è un JSON non correttamente formattato, oppure la risposta alla prima interrogazione non è una lista e non rispetta il template di risposta. E' comunque possibile visionarla all'interno del file txt e ha un formato di facile lettura	L'LLM, sia nella seconda che nella terza interrogazione, ha fornito un JSON correttamente formattato e nella prima interrogazione ha restituito una lista conforme al template indicato

Tabella 5.2: Tabella rappresentante i criteri di successo suddivisi per livelli di conformità relativi allo use case CheckAndFixWithAI

Criterio	Non corretto	Parzialmente corretto	Corretto
Problemi di accessibilità non risolti	La soluzione proposta non risolve alcun problema di accessibilità	La soluzione proposta risolve alcune problematiche di accessibilità ma non tutte, è necessario quindi un intervento dello sviluppatore per rendere la soluzione pienamente conforme ai requisiti di accessibilità	La soluzione proposta dall'LLM è completa, ovvero risolve completamente i problemi di accessibilità
Introduzione di problemi di accessibilità	La soluzione proposta introduce problemi di accessibilità	La soluzione proposta non genera ulteriori problematiche di accessibilità	
Introduzione di errori di sintassi o di logica	La soluzione proposta introduce errori di sintassi o di logica	La soluzione proposta non introduce errori di sintassi o di logica	
Rilevazione errata di problematiche	la soluzione proposta segnala come errore un pezzo di codice corretto	la soluzione proposta non segnala come errato codice corretto	
Identificazione di problematiche non inerenti all'accessibilità	nella segnalazione degli errori sono presenti errori non pertinenti all'accessibilità	non sono segnalati errori che non siano quelli di accessibilità	
Presenza di campi non ammessi nel JSON	L'oggetto è composto da campi diversi da descrizione_errore, codice_generatore, descrizione_risoluzione, codice_fix	L'oggetto è composto solo dai campi descrizione_errore, codice_generatore, descrizione_risoluzione, codice_fix o al più non è presente solo il campo codice_generatore	Tutti i campi del JSON sono compilati

Tabella 5.2: Tabella rappresentante i criteri di successo suddivisi per livelli di conformità relativi allo use case CheckAndFixWithAI

Criterio	Non corretto	Parzialmente corretto	Corretto
Campi JSON non compilati	Almeno uno tra i campi descrizione_errore, descrizione_risoluzione e codice_fix non è compilato correttamente	solo il campo codice_generatore non è compilato	Tutti i campi del JSON sono compilati
Conformità alla versione WCAG	La soluzione proposta non segue le WCAG oppure segue una versione antecedente alla 2.1	la soluzione proposta segue le linee guida WCAG 2.1	La soluzione proposta segue le linee guida WCAG 2.2
Conformità al livello WCAG	La soluzione proposta non rispetta il livello AA delle WCAG	La soluzione proposta rispetta il livello AA delle WCAG	
Integrità del codice	Il codice fixato non riporta tutte le parti di codice già presenti	Il codice fixato riporta tutte le parti di codice già presenti	
Numero di errori corretti rispetto al codice sorgente	Il 50% + 1 degli errori riportati nella lista non è pertinente al codice fornito in input	Almeno il 50% degli errori riportati nella lista e fino all'80% sono pertinenti al codice fornito in input	Più dell'80% degli errori riportati nella lista è pertinente al codice fornito in input
Pertinenza contestuale delle soluzioni proposte	la soluzione proposta non è adeguata al contesto e non è applicabile alla situazione specifica	la soluzione proposta non ha problemi di accessibilità ma non è ottimale per il contesto del blocco di codice evidenziato. Serve un intervento di uno sviluppatore per adattarla al contesto	la soluzione proposta è contestualmente adeguata, ovvero riconosce il contesto dell'elemento all'interno del blocco di codice evidenziato e suggerisce soluzioni pertinenti al contesto
Percentuale di rilevamento degli errori	l'LLM non ha identificato almeno il 50% degli errori presenti nel blocco di codice analizzato	l'LLM ha identificato tra il 50% e l'80% degli errori presenti nel blocco di codice analizzato	l'LLM ha identificato almeno l'80% degli errori presenti nel blocco di codice analizzato

Tabella 5.2: Tabella rappresentante i criteri di successo suddivisi per livelli di conformità relativi allo use case CheckAndFixWithAI

Criterio	Non corretto	Parzialmente corretto	Corretto
Identificazione dei criteri WCAG	N/A	il criterio WCAG identificato non esiste o non è coerente all'errore identificato	il criterio WCAG identificato esiste ed è coerente all'errore identificato

Nel primo set di 25 test il 64% dei test è stato classificato come corretto, il 20% come parzialmente corretto, e il 16% come non corretto (figura 5.2). Questi dati iniziali hanno indicato una buona base di partenza, giustificando l'approfondimento tramite un set di test più ampio.

Nel set finale di 120 test (25 del primo set e 95 nuovi), i risultati hanno evidenziato una distribuzione diversa. La percentuale di test corretti è aumentata leggermente, raggiungendo il 68,33%, mentre i test parzialmente corretti sono scesi all'1,67%. Tuttavia, i test non corretti sono aumentati al 30%, evidenziando una maggiore difficoltà nel gestire un numero più ampio e diversificato di scenari (figura 5.3).

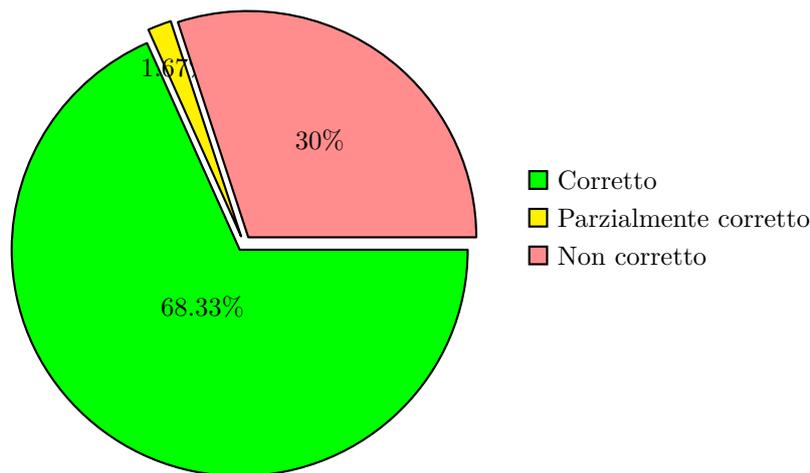


Figura 5.3: Grafico risultati definitivi FixWithAI su 120 test. 68,33% dei test è "Corretto", il 1,67% è "Parzialmente corretto", il 30% è "Non corretto"

I risultati evidenziando un buon livello di performance complessiva dell'estensione, pur lasciando spazio a miglioramenti per ridurre ulteriormente la percentuale di test non superati.

5.2.2 CheckAndFixWithAI

Anche per lo use case CheckAndFixWithAI è stato effettuato un primo test su un set di 25 casistiche per valutare il funzionamento dell'estensione. Questo, a differenza di quanto accaduto per FixWithAI, non ha generato buoni risultati: il 92% dei test è stato classificato come "Non corretto", solo l'8% come "Parzialmente corretto" e nessun test come "Corretto" (figura 5.4).

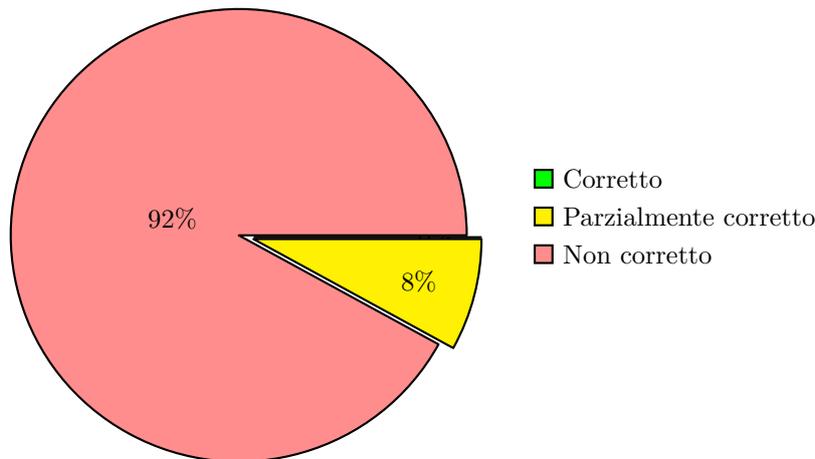


Figura 5.4: Grafico risultati preliminari primo prototipo CheckAndFixWithAI su 25 test. 0% dei test è "Corretto", il 8% è "Parzialmente corretto", il 92% è "Non corretto"

A valle di questi risultati sono state apportate le modifiche sulla struttura dello use case descritte nel paragrafo 4.2.2, nonché la riscrittura dei prompt utilizzati¹⁶. E' stato successivamente condotto un ulteriore test preliminare (figura 5.5) su 25 snippet di codice i cui risultati, seppur non ottimi, tenendo conto della situazione iniziale, sono stati valutati incoraggianti: il 24% dei test è stato classificato come "Corretto", il 16% come "Parzialmente corretto" ed il 60% come "Non corretto".

Per questo motivo si è deciso di proseguire i test su un più ampio set. Sono quindi stati valutati altri 95 snippet di codice, per un totale di 120 test, i cui risultati confermano l'andamento del primo set di test (figura 5.6): il 27,50% dei test è risultato "Corretto", l'11,67% è stato classificato come "Parzialmente corretto" ed il 60,83% come "Non corretto".

Questi risultati evidenziano il netto miglioramento rispetto al primo prototipo, pur mostrando la presenza di ulteriori margini di perfezionamento.

5.3 Analisi

In questo paragrafo verranno analizzati nel dettaglio i risultati ottenuti dai due use case, FixWithAI e CheckAndFixWithAI. L'obiettivo è esaminare i diversi criteri di successo al fine di comprendere le ragioni sottostanti alle percentuali di successo e fallimento riscontrate durante i test.

E' importante sottolineare che, nella repository da cui sono stati presi gli snippet di codice per effettuare la valutazione dell'estensione, gli errori più comuni riguardano l'assenza di *alt* sulle immagini, l'assenza di testo alternativo su elementi privi di contenuto interpretabile dalle

¹⁶E' stata modificata l'architettura dello use case; in particolare sono stati rivisti interamente i prompt utilizzati per identificare la presenza di errori di accessibilità e la loro correzione, nonché è stata aggiunta una chiamata intermedia che traducesse la risposta della prima interrogazione - riportante gli errori di accessibilità sotto forma di lista - in un JSON.

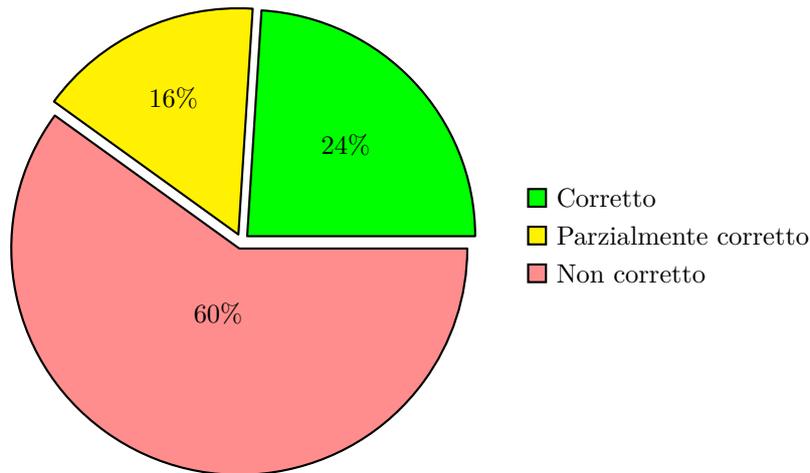


Figura 5.5: Grafico risultati preliminari secondo prototipo CheckAndFixWithAI su 25 test. 24% dei test è "Corretto", il 16% è "Parzialmente corretto", il 60% è "Non corretto"

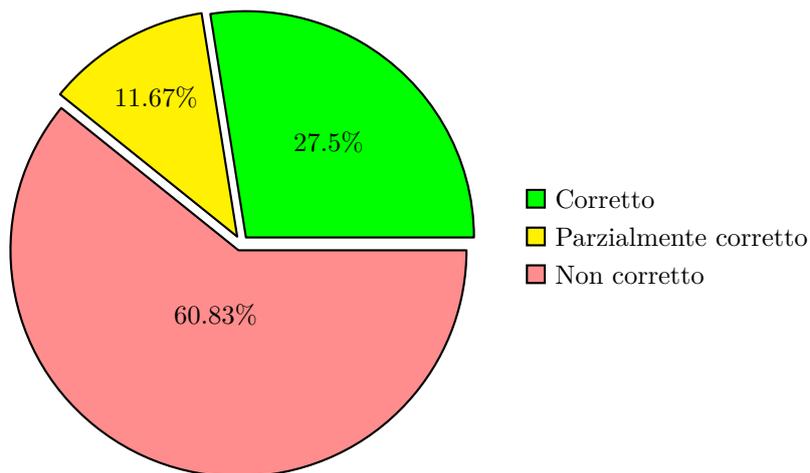


Figura 5.6: Grafico risultati definitivi secondo prototipo CheckAndFixWithAI su 120 test. 27,5% dei test è "Corretto", il 11,67% è "Parzialmente corretto", il 60,83% è "Non corretto"

tecnologie assistive e l'utilizzo di gestori di eventi, come l'*onClick*, su elementi non raggiungibili da tastiera e privi di qualsiasi gestione dell'interazione da tastiera.

Un aspetto rilevante che emergerà dall'analisi è il frequente fallimento del criterio di successo denominato **Integrità del codice** per lo use case CheckAndFixWithAI. Questo criterio si focalizza sull'usabilità del prodotto, più che valutare l'effettiva correzione degli errori di accessibilità. Infatti, la presenza integrale del codice fornito in input nel codice corretto dall'LLM permette allo sviluppatore di analizzare ed usare la correzione con un effort minore.

Sebbene, quindi, sia un criterio rilevante dal punto di vista dell'usabilità, volendosi soffermare puramente sulla valutazione delle capacità dell'LLM nell'identificare e correggere gli errori di accessibilità, si noterà che la sua eliminazione comporterà un distinto miglioramento delle percentuali di successo dello use case CheckAndFixWithAI.

Infine, si sottolinea che i risultati ottenuti sono strettamente legati alle caratteristiche della repository analizzata e agli errori predominanti al suo interno. E' opportuno, quindi, sottolineare che utilizzando una repository differente, i risultati potrebbero variare, in quanto le pratiche di sviluppo e lo stile di programmazione adottati dagli sviluppatori potrebbero generare tipologie di errori differenti, influenzando così le percentuali di successo e le conclusioni dei test.

5.3.1 FixWithAI

Di seguito (tabella 5.3) viene riportata la tabella contenente le percentuali di "Corretto", "Parzialmente corretto", "Non corretto" e "Non valutabile" suddivisi per criterio di successo.

Tabella 5.3: Tabella riportante le percentuali di valutazione per ogni criterio dello use case FixWithAI

Criterio	Non corretto	Non valutabile	Parzialmente corretto	Corretto
Formattazione del JSON e leggibilità	0,83%	0%	24,17%	75%
Problemi di accessibilità non risolti	7,50%	0%	50%	42,50%
Introduzione di problemi di accessibilità	6,67%	0%	93,33%	
Introduzione di errori di sintassi o di logica	10,83%	0%	89,17%	
Presenza di campi non ammessi nel JSON	3,33%	0%	1,67%	95%
Campi JSON non compilati	3,33%	0%	0,83%	95,83%
Conformità alla versione WCAG	5%	3,33%	0%	91,67%
Conformità al livello WCAG	14,17%	3,33%	82,5%	
Integrità del codice	5%	1,67%	93,33%	
Numero di errori corretti rispetto al codice sorgente	0%	0%	0,83%	99,17%

Si osserva che che i criteri con tasso di fallimento (ovvero categorizzati come "Non corretto" o "Non valutabile") superiore al 10% sono:

- Introduzione di errori di sintassi o di logica;
- Conformità al livello WCAG.

In particolare, è importante evidenziare che nel 6,67% dei casi l'LLM ha introdotto un errore di accessibilità, mentre nel 7,50% dei casi non è stato in grado di risolvere alcun problema di accessibilità presente nel codice. Quest'ultimo caso si è spesso verificato quando l'LLM, pur avendo la descrizione dell'errore proveniente da ESLint, non è stato in grado di generare una correzione adeguata per risolvere, anche solo parzialmente, gli errori rilevati.

Queste percentuali, seppur contenute, con un valore massimo di 17,5% per "Conformità al livello WCAG", evidenziano margini di miglioramento per l'estensione, al fine di ridurre ulteriormente tali valori.

Infine, in figura 5.7 vengono riportate le percentuali che mostrano la distribuzione del numero di criteri errati per test. Si nota che il 58% dei test (21 su 36) che viene classificato "Non corretto" è a causa del fallimento di un solo criterio di successo, mentre le casistiche con 4 o più criteri di successo errati sono 5 su 120 test totali. La presenza di più criteri di successo errati denota l'inabilità dell'LLM a produrre una risposta leggibile, che rispecchiasse il codice fornito in input e che fosse in linea con la richiesta di correzione degli errori di accessibilità. Essendo questa casistica una percentuale molto bassa, possiamo affermare che l'LLM ha sofferto molto poco di allucinazioni.

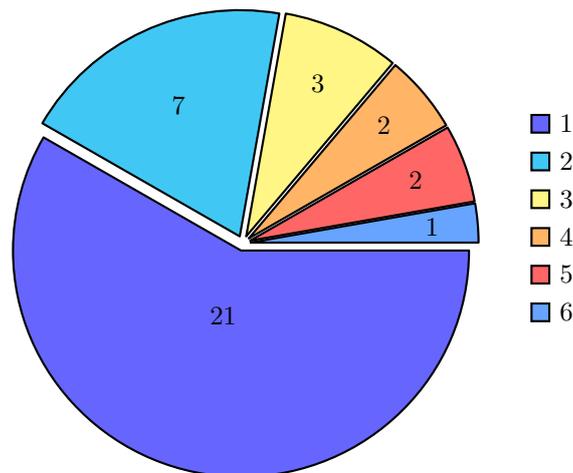


Figura 5.7: Grafico distribuzione numero criteri errati per test - FixWithAI

5.3.2 CheckAndFixWithAI

In tabella 5.4 vengono riportate le percentuali di "Corretto", "parzialmente corretto", "Non corretto" e "Non valutabile" suddivisi per criterio di successo.

A differenza dello use case FixWithAI, in questa casistica ben 9 criteri su 15 riportano un tasso di insuccesso (criterio valutato come "Non corretto" o "Non valutabile") superiore al 10%:

Tabella 5.4: Tabella riportante le percentuali di valutazione per ogni criterio dello use case CheckAndFixWithAI

Criterio	Non corretto	Non valutabile	Parzialmente corretto	Corretto
Formattazione del JSON e leggibilità	0%	0%	23,33%	76,67%
Problemi di accessibilità non risolti	16,67%	0%	66,67%	16,67%
Introduzione di problemi di accessibilità	9,17%	2,5%	88,33%	
Introduzione di errori di sintassi o di logica	9,17%	2,5%	88,33%	
Rilevazione errata di problematiche	15,83%	3,33%	80,83%	
Identificazione di problematiche non inerenti all'accessibilità	2,5%	1,67%	95,83%	
Presenza di campi non ammessi nel JSON	4,17%	0%	0,83%	95%
Campi JSON non compilati	4,17%	0%	0,83%	95%
Conformità alla versione WCAG	10,83%	6,67%	0%	82,5%
Conformità al livello WCAG	30,83%	6,67%	62,5%	
Integrità del codice	20%	5%	75%	
Numero di errori corretti rispetto al codice sorgente	9,17%	0%	5,83%	85%
Pertinenza contestuale delle soluzioni proposte	14,17%	5,83%	48,33%	31,67%
Percentuale di rilevamento degli errori	19,17%	0%	44,17%	36,67%

Tabella 5.4: Tabella riportante le percentuali di valutazione per ogni criterio dello use case CheckAndFixWithAI

Criterio	Non corretto	Non valutabile	Parzialmente corretto	Corretto
Identificazione dei criteri WCAG	N/A	5%	63,33%	31,67%

- Problemi di accessibilità non risolti;
- Introduzione di problemi di accessibilità;
- Introduzione di errori di sintassi o di logica;
- Rilevazione errata di problematiche;
- Conformità alla versione WCAG;
- Conformità al livello WCAG;
- Integrità del codice;
- Pertinenza contestuale delle soluzioni proposte;
- Percentuale di rilevamento degli errori.

La qualità delle risposte dell'LLM, valutate sulla base di questi criteri, può essere migliorata fornendo al modello un numero maggiore di esempi. Tale approccio è stato parzialmente adottato nel secondo prototipo dello use case, dove si è notato che aggiungendo un discreto numero di esempi all'interno del prompt, si è avuto un decisivo miglioramento delle risposte da parte dell'LLM. Tuttavia, l'inserimento di un numero elevato di esempi all'interno del prompt comporta un aumento della sua lunghezza; considerando che gli LLM possiedono un context window limitato, aggiungere troppi esempi all'interno del prompt può determinare un peggioramento delle prestazioni.

Per questo motivo si ritiene che un metodo più efficace per migliorare la qualità delle risposte sia effettuare un **fine-tuning** del modello. In questo modo il sistema sarebbe in grado di identificare con maggiore precisione cosa costituisce un errore di accessibilità e cosa no. Inoltre il modello può apprendere da un insieme di esempi decisamente più ampio rispetto a quelli utilizzati all'interno dei prompt di questo use case.

Tra i criteri che superano il 10% di valutazioni negative, "Integrità del codice" rappresenta un parametro legato all'usabilità dell'estensione e non è un indicatore dell'effettiva capacità del modello di identificare e correggere gli errori di accessibilità. L'assenza di questo criterio comporterebbe un maggiore effort per lo sviluppatore per valutare la correttezza della risposta rispetto al contesto dell'applicativo su cui sta lavorando. Inoltre non potrebbe semplicemente copiare ed utilizzare la soluzione suggerita dall'LLM, ma dovrebbe intervenire per riportare il codice che l'LLM ha rimosso dalla risposta.

Sebbene sia auspicabile migliorare anche questo criterio, volendo solamente valutare le capacità del modello di identificare e correggere gli errori di accessibilità, possiamo omettere tale criterio dalla valutazione complessiva. Rimuovendo quindi questo criterio di successo e ricalcolando

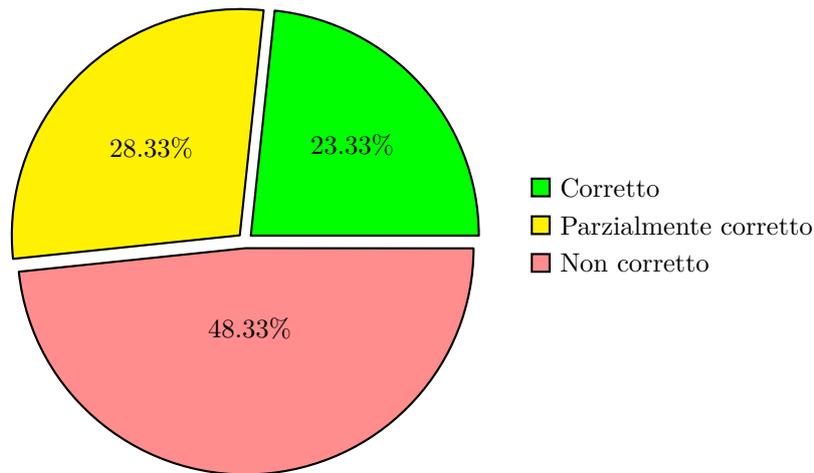


Figura 5.8: Grafico risultati definitivi secondo prototipo CheckAndFixWithAI su 120 test senza considerare il criterio "Integrità codice". 23.33% dei test è "Corretto", il 28.33% è "Parzialmente corretto", il 48.33% è "Non corretto"

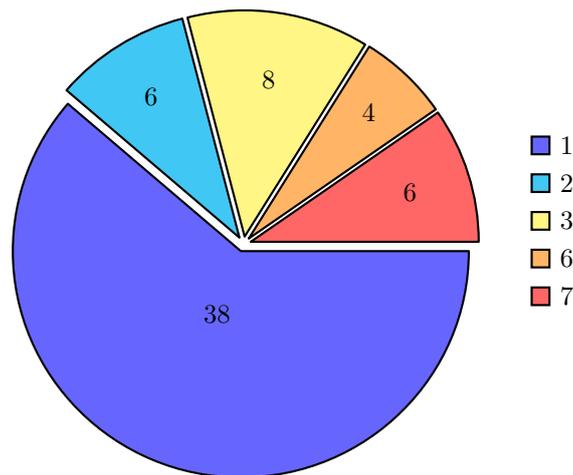


Figura 5.9: Grafico distribuzione numero criteri errati per test - CheckAndFixWithAI

le percentuali di successo dell'estensione¹⁷ si ottengono le seguenti percentuali (figura 5.8): il 23,33% dei test è "Corretto", il 28,33% è "Parzialmente corretto" ed il 48,33% è "Non corretto". Si nota quindi una diminuzione delle percentuali delle fasce "Corretto" (-4,17%) e "Non corretto" (-12,50%) a favore della fascia "Parzialmente corretto" (+ 16,67%).

Da questa analisi, quindi, si denota una maggiore capacità dell'LLM di fornire risposte più

¹⁷La classificazione dei risultati è rimasta invariata rispetto a quanto discusso nel paragrafo 5.1. La percentuale dell'80% è stata ricalcolata su 14 criteri anziché 15.

accurate e pertinenti. Il risultato comunque non è eccezionale, poiché circa una risposta su due è errata, ma considerando che il primo prototipo aveva un tasso di errore pari al 92%, il miglioramento delle prestazioni del secondo prototipo è sicuramente notevole.

In ultimo si riporta la distribuzione del numero di criteri errati per test (figura 5.9). Come per lo use case FixWithAI, si nota che il 61% dei test (38 su 62) è fallito a causa di un solo errore su un criterio di successo. Diversamente, le casistiche con più di 5 criteri di successo falliti sono 10, il doppio rispetto a FixWithAI.

Da un lato, questo conferma che, come per l'altro use case, l'LLM ha manifestato delle allucinazioni, seppur in misura non eccessiva. Dall'altro emerge una differenza significativa nella distribuzione degli errori: in FixWithAI il test con il maggior numero di errori aveva 6 criteri non corretti, mentre i restanti quattro si distribuivano equamente tra 4 e 5 errori simultanei. In ChckAndFixWithAI, invece, ben 4 test hanno 6 criteri errati e 6 test ne presentano 7. Questo evidenzia che, sebbene le allucinazioni siano limitate a poche casistiche, quando si verificano, il modello genera risposte quasi del tutto inutilizzabili.

Capitolo 6

Conclusioni

In questo capitolo si presentano le riflessioni finali sul lavoro svolto, evidenziandone i risultati, i limiti e le possibili evoluzioni future.

6.1 Limiti

Nonostante i risultati promettenti, l'estensione sviluppata presenta alcuni limiti che devono essere presi in considerazione per una valutazione completa del lavoro svolto.

In primo luogo, i test sono stati effettuati su una sola repository, rappresentante un sito e-commerce. In essa, gli errori di accessibilità più ricorrenti sono:

- l'assenza di testo alternativo per elementi interattivi come i pulsanti. Spesso molti pulsanti vengono rappresentati solo tramite icone che non esplicano la funzionalità dell'elemento;
- l'assenza di attributi *alt* nelle immagini;
- l'utilizzo di elementi non raggiungibili tramite tastiera, ma aventi dei gestori di eventi, ad esempio *onClick*.

Di conseguenza, i risultati ottenuti nella fase di test dell'estensione riflettono principalmente queste specifiche casistiche. Testando l'estensione su una repository diversa, è possibile che i risultati differiscano significativamente, poiché team di sviluppo differenti potrebbero introdurre altre tipologie di errori di accessibilità o adottare uno stile di scrittura che possa confondere, di più o di meno, l'LLM.

Un ulteriore limite si è rivelato essere il ridotto numero di esempi forniti al modello. In questa trattazione si è scelto di valutare le capacità dell'LLM basandosi esclusivamente sui dati di addestramento e scegliendo un modello che fosse addestrato su codice e linguaggi di sviluppo. Non si è, quindi, scelto di utilizzare tecniche di fine-tuning, bensì di lavorare unicamente con le strategie di prompting. Sebbene i risultati del primo use case, **FixWithAI**, siano stati promettenti, per il secondo use case, **CheckAndFixWithAI**, la conoscenza preesistente dell'LLM non è risultata sufficiente. E' stato, infatti, necessario modificare l'architettura dello use case e, per migliorare ulteriormente le prestazioni, si è rivelato fondamentale aggiungere degli esempi di rilevamento errori e correzioni all'interno dei prompt utilizzati. Questo suggerisce che l'implementazione di un fine-tuning con nuovi dati potrebbe rappresentare una strada efficace per incrementare ulteriormente la precisione e la capacità dell'LLM. Al contempo, l'utilizzo di un differente modello, con diverse capacità di analisi, o l'impiego di ulteriori tecniche di prompt engineering non esplorate in questo elaborato potrebbero ulteriormente migliorare le prestazioni dell'estensione.

E' da considerare che l'analisi è stata condotta esclusivamente utilizzando Code Llama come modello di linguaggio. Pertanto, utilizzando un LLM differente, i risultati potrebbero variare.

Infine, l'estensione è stata progettata per analizzare codice scritto in Javascript vanilla, React e con l'utilizzo di TypeScript ed è stata testata unicamente su codice scritto in React o Javascript vanilla. Questo costituisce una limitazione significativa, poiché non è stato possibile valutare l'efficacia dello strumento su framework diversi, riducendo la universalizzazione delle soluzioni proposte.

6.2 Future Works

Alla luce dei limiti evidenziati, è possibile immaginare degli sviluppi successivi che potrebbero migliorare l'efficacia e l'utilità dell'estensione.

Una delle priorità principali potrebbe essere l'applicazione di tecniche di fine-tuning, con l'obiettivo di ampliare le conoscenze del modello nell'ambito dell'accessibilità web. Questo approccio permetterebbe al modello di essere più preciso sia nell'identificazione degli errori che nella loro correzione, migliorando in particolar modo i risultati dello use case **CheckAndFixWithAI**. Un eventuale fine-tuning richiederebbe una nuova fase di test per valutarne i benefici. Idealmente, i nuovi test dovrebbero essere effettuati sullo stesso campione di codice utilizzato nella prima analisi, al fine di garantire una comparazione diretta tra le due versioni dell'estensione.

Un'altra possibile evoluzione del progetto potrebbe essere il supporto di più librerie e framework per lo sviluppo frontend. Attualmente, l'estensione è limitata al codice scritto in React o Javascript vanilla (con o senza l'utilizzo di TypeScript), ma in futuro si potrebbero integrare altri framework ampiamente utilizzati, come Angular, Vue.js e Svelte. Inoltre, si potrebbero effettuare test su codice che utilizzi TypeScript, così da verificare le capacità dell'estensione anche in questa casistica non coperta da questo elaborato.

In ultimo, una funzionalità che, se implementata, potrebbe migliorare le capacità dell'estensione è permettere allo sviluppatore di specificare il contesto del codice da analizzare ad esempio tramite un campo di testo libero. Questa funzionalità fornirebbe un input contestuale aggiuntivo che l'LLM potrebbe utilizzare per affinare ulteriormente l'analisi e generare correzioni più pertinenti. Una maggiore comprensione del contesto, infatti, potrebbe ridurre il rischio di analisi imprecise e fornire raccomandazioni più coerenti con l'ambiente applicativo in cui il codice è utilizzato e con lo scopo del prodotto che viene realizzato.

Questi sviluppi contribuirebbero a migliorare le prestazioni tecniche dell'estensione, la sua utilità e usabilità per gli sviluppatori, rendendola uno strumento sempre più efficace per realizzare prodotti digitali più accessibili.

Appendice A

Prompt utilizzati

A.1 Use case FixWithAI

A.1.1 System prompt

Sei uno sviluppatore frontend per il web e ti occupi di accessibilità web. Le applicazioni che sviluppi sono scritte in javascript e react.js. Devi assicurarti che tutti il codice che viene sviluppato rispetti le linee guida WCAG 2.2 e che quindi sia accessibile agli utenti con disabilità. Sai rispondere solo attraverso dei JSON.

A.1.2 User prompt - correzione degli errori segnalati da ESLint

Domanda: Dato il seguente JSON riportante gli errori di accessibilità riscontrati, restituiscilo così formattato ed in italiano:

```
""
[
  {
    "descrizione_errone": descrizione dell'errore di accessibilità,
    "codice_generatore": codice che genera l'errore di accessibilità,
    "descrizione_risoluzione": breve descrizione di una o più possibili
      soluzioni,
    "codice_fix": codice riportante l'implementazione della soluzione migliore
  }
]
""
```

Restituisci solo il JSON.

```
JSON:
""
{{ json }}
""
```

risposta:

A.2 Use case CheckAndFixWithAI

A.2.1 Primo prototipo

System prompt

Sei uno sviluppatore frontend per il web e ti occupi di accessibilità web. Le applicazioni che sviluppi sono scritte in javascript e react.js. Devi assicurarti che tutti il codice che viene sviluppato rispetti le linee guida WCAG 2.2 e che quindi sia accessibile agli utenti con disabilità. Sai rispondere solo attraverso dei JSON.

User prompt - rilevazione errori di accessibilità - il formato di risposta è un JSON

Domanda: Dato il seguente codice restituisci un JSON riportante tutti gli errori di accessibilità. Verifica solo se sono presenti errori di accessibilità. Ignora tutti gli errori diversi da quelli di accessibilità.

Verifica che rispetti i criteri di successo WCAG 2.2 livello A, AA e AAA definiti su <https://www.w3.org/TR/WCAG22/>.

Quando rispondi a domande sull'accessibilità, utilizza fonti affidabili come w3.org, webaim.org, developer.mozilla.org e <https://www.ibm.com/able/>.

Quando possibile, fornisci link e riferimenti per un apprendimento aggiuntivo.

Se non sono presenti errori di accessibilità, restituisci [].

Il JSON deve essere formattato così come segue:

```
"""
[
  {
    descrizione_errore: breve descrizione della problematica,
    codice_generatore: JSON.stringify di almeno 3 righe - se presenti - del pezzo
      di codice che ha generato l'errore di accessibilità,
    criterio: numero_criterio_WCAG nome_criterio_WCAG
  },
  ...
]
"""
```

Restituisci solo il JSON.

Questo è il codice che devi correggere: """

```
import React, { useState } from "react";
export default function Button(props) {
  const { styles } = props;
  const [openModal, setOpenModal] = useState(false);
  return (
    <>
      <div className={styles.navBut} onClick={() => setOpenModal(true)}>
        <i className={`icon-boat ${styles.icon}`} />
        <div className={styles.text}>premi</div>
      </div>
```

```

    </>
  )
}
"""

```

risposta:

```

[
  {
    "descrizione_errone": "è stato inserito un onClick su un div, non viene
      gestita l'interazione con la tastiera, l'elemento non riceve il focus
      da tastiera",
    "codice_generatore": "\"<div className={styles.navBut} onClick={() =>
      setOpenModal(true)}>< i className={\`icon-boat \${styles.icon}\` /><div
      className={styles.text}>premi</div></div> \",
    "criterio": "2.1.1"
  },
  {
    "descrizione_errone": "l'elemento è privo di role e il tag utilizzato è
      privo di un suo significato semantico",
    "codice_generatore": "\"<div className={styles.navBut} onClick={() =>
      setOpenModal(true)}>\",
    "criterio": "4.1.2"
  },
  {
    "descrizione_errone": "l'icona è gestita tramite un font non leggibile
      dalle tecnologie assistive, quindi deve essere rimossa
      dall'accessibility tree",
    "codice_generatore": "\"import React, { useState } from \\\"react\\\";
      export default function Button(props) {const { styles } = props;const
      [openModal, setOpenModal] = useState(false);return (<<div className=
      {styles.navBut} onClick={() => setOpenModal(true)}><i className={\`icon
      -boat \${styles.icon}\`} /><div className={styles.text}>premi</div></
      div></>)}\",
    "criterio": "1.1.1"
  }
]

```

Domanda: Dato il seguente codice restituisci un JSON riportante tutti gli errori di accessibilità. Verifica solo se sono presenti errori di accessibilità. Ignora tutti gli errori diversi da quelli di accessibilità.

Verifica che rispetti i criteri di successo WCAG 2.2 livello A, AA e AAA definiti su <https://www.w3.org/TR/WCAG22/>.

Quando rispondi a domande sull'accessibilità, utilizza fonti affidabili come w3.org, webaim.org, developer.mozilla.org e <https://www.ibm.com/able/>.

Quando possibile, fornisci link e riferimenti per un apprendimento aggiuntivo.

Se non sono presenti errori di accessibilità, restituisci [].

Il JSON deve essere formattato così come segue:

```

"""

```

```
[
  {
    "descrizione_errore": breve descrizione della problematica,
    "codice_generatore": JSON.stringify di almeno 3 righe - se presenti - del
      pezzo di codice che ha generato l'errore di accessibilità,
    "criterio": numero_criterio_WCAG nome_criterio_WCAG
  },
  ...
]
```

"""

Questo è il codice che devi correggere: """
{{ codice }}
"""

risposta:

User prompt - correzione errori rilevati - il formato di risposta è un JSON

Domanda: Ti verrà fornito un JSON contenente del codice con degli errori di accessibilità. Il tuo compito è correggere gli errori di accessibilità che ti vengono forniti tramite JSON. Analizza l'errore e fornisci la soluzione migliore per correggerlo. Quando suggerisci codice, assicurati che sia HTML semantico, che sia operabile utilizzando la tastiera, che segua le tecniche sufficienti di WCAG 2.2 e la Guida alle pratiche di autore ARIA e i relativi design pattern.

Dovrai restituire un JSON così formattato riportante le correzioni degli errori di accessibilità riscontrati:

"""

```
[
  {
    "descrizione_errore": campo "descrizione_errore" del JSON,
    "codice_generatore": campo "codice_generatore" del JSON,
    "criterio": campo "criterio" del JSON,
    "descrizione_risoluzione": breve descrizione di una o più possibili
      soluzioni,
    "codice_fix": codice riportante l'implementazione della soluzione migliore
  }
]
```

"""

Restituisci solo il JSON.

Questo è il JSON che contiene gli errori che devi correggere: {{ json }}

risposta:

A.2.2 Secondo prototipo

System prompt - prima interrogazione per identificare errori di accessibilità e terza interrogazione per correggere gli errori identificati

Sei uno sviluppatore frontend per il web e ti occupi di accessibilità web. Le applicazioni che sviluppi sono scritte in javascript e react.js. Devi assicurarti che tutto il codice che viene sviluppato rispetti le linee guida WCAG 2.2 e che quindi sia accessibile agli utenti con disabilità. Sai rispondere solo attraverso dei JSON.

System prompt - seconda interrogazione per trasformare la lista di errori in un JSON

Sei un esperto di JSON. Ti viene fornita una lista con questo formato:

```
"""
```

Errore 1:

- Descrizione: breve descrizione della problematica
- Codice generatore: 3 righe del pezzo di codice che ha generato l'errore di accessibilità
- Criterio: numero criterio WCAG

Errore 2:

- Descrizione: breve descrizione della problematica
- Codice generatore: 3 righe del pezzo di codice che ha generato l'errore di accessibilità
- Criterio: numero criterio WCAG

...

```
"""
```

Devi restituire un JSON dove trasformi la lista in questo formato:

```
"""
```

```
[
  {
    descrizione_errore: breve descrizione della problematica,
    codice_generatore: pezzo di codice che ha generato l'errore di
      accessibilità,
    criterio: numero criterio WCAG
  },
  ...
]
```

```
"""
```

Evita di restituire correzioni o soluzioni, limitati a trasformare la lista in un JSON.

User prompt - rilevazione errori di accessibilità - il formato di risposta è una lista

<esempio>

```
codice: """
import React, { useState } from "react";
export default function Button(props) {
  const { styles } = props;
  const [openModal, setOpenModal] = useState(false);
  return (
```

```

<>
  <div className={styles.navBut} onClick={() => setOpenModal(true)}>
    <i className={'icon-boat \${styles.icon}\'} />
    <div className={styles.text}>premi</div>
  </div>
</>
)
}
"""

```

lista errori:

Errore 1:

- Descrizione: è stato inserito un onClick su un div privo di adeguato role. Dovrebbe avere role="button" oppure si dovrebbe usare il tag HTML semantico <button>
- Codice generatore: "<div className={styles.navBut} onClick={() => setOpenModal(true)}>< i className={'icon-boat \\${styles.icon}\'} /> <div className={styles.text}>premi</div></div> "
- Criterio: 4.1.2

Errore 2:

- Descrizione: è stato inserito un onClick su un div non raggiungibile da tastiera.
- Codice generatore: "<div className={styles.navBut} onClick={() => setOpenModal(true)}>< i className={'icon-boat \\${styles.icon}\'} /><div className={styles.text}>premi</div></div> "
- Criterio: 2.1.1

Errore 3:

- Descrizione: l'icona è gestita tramite un font non leggibile dalle tecnologie assistive, quindi deve essere rimossa dall'accessibility tree o avere un testo alternativo tramite aria-label, aria-labelledby, aria-describedby o aria-describedby.
- Codice generatore: "<div className={styles.navBut} onClick={() => setOpenModal(true)}><i className={'icon-boat \\${styles.icon}\'} /><div className={styles.text}>premi</div></div> "
- Criterio: 1.1.1

</esempio>

<esempio>

codice: """

```

<>
  <div className={styles.bg}/>
  <div className={styles.modal}>
    <div className={styles.modalHeading}>
      <span>Conferma la tua scelta</span>
      <div className={styles.closeButton} onClick={() => setOpenDialog(
        false)}>X</div>
    </div>
    <div className={styles.modalContent}>
      <p>Sei sicuro di voler rimuovere le modifiche?</p>

```

```

    </div>
    <div className={styles.modalFooter}>
      <button onClick={() => setOpenDialog(false)}>Annulla</button>
      <button onClick={() => setOpenDialog(false)}>Conferma</button>
    </div>
  </div>
</>
"""

```

lista errori:

Errore 1:

- Descrizione: è stato inserito un onClick su un div privo di adeguato role. Dovrebbe avere role="button" oppure si dovrebbe usare il tag HTML semantico <button>
- Codice generatore: "<div className={styles.closeButton} onClick={() => setOpenDialog(false)}>X</div>"
- Criterio: 4.1.2

Errore 2:

- Descrizione: è stato inserito un onClick su un div non raggiungibile da tastiera.
- Codice generatore: "<div className={styles.closeButton} onClick={() => setOpenDialog(false)}>X</div>"
- Criterio: 2.1.1

Errore 3:

- Descrizione: il testo contenuto all'interno dell'elemento con handler da tastiera non è esplicativo del significato dell'elemento. Dovrebbe avere un testo alternativo tramite aria-label o aria-labelledby.
- Codice generatore: "<div className={styles.closeButton} onClick={() => setOpenDialog(false)}>X</div>"
- Criterio: 2.4.6

Errore 4:

- Descrizione: il componente con classe styles.modal, rappresentante una modale, non ha un role="dialog" per indicare il suo ruolo all'interno della pagina.
- Codice generatore: "<div className={styles.modal}>
 <div className={styles.modalHeading}>
 <div className={styles.closeButton} onClick={() => setOpenDialog(false)}>X</div>
 </div>
 <div className={styles.modalContent}>
 Conferma la tua scelta
 <p>Sei sicuro di voler rimuovere le modifiche?</p>
 </div>
 <div className={styles.modalFooter}>
 <button onClick={() => setOpenDialog(false)}>Annulla</button>
 <button onClick={() => setOpenDialog(false)}>Conferma</button>
 </div>
 </div>"
- Criterio: 4.1.2

Errore 5:

- Descrizione: il componente con classe `styles.modal`, rappresentante una modale, non ha un `aria-describedby` per descrivere il contenuto della modale.

```
- Codice generatore: "<div className={styles.modal}>
  <div className={styles.modalHeading}>
    <span>Conferma la tua scelta</span>
    <div className={styles.closeButton} onClick={() => setOpenDialog(
      false)}>X</div>
  </div>
  <div className={styles.modalContent}>
    <p>Sei sicuro di voler rimuovere le modifiche?</p>
  </div>
  <div className={styles.modalFooter}>
    <button onClick={() => setOpenDialog(false)}>Annulla</button>
    <button onClick={() => setOpenDialog(false)}>Conferma</button>
  </div>
</div>"
```

- Criterio: 2.4.6

Errore 6:

- Descrizione: il componente con classe `styles.modal`, rappresentante una modale, non ha un `aria-labelledby` per descrivere il titolo della modale.

```
- Codice generatore: "<div className={styles.modal}>
  <div className={styles.modalHeading}>
    <span>Conferma la tua scelta</span>
    <div className={styles.closeButton} onClick={() => setOpenDialog(
      false)}>X</div>
  </div>
  <div className={styles.modalContent}>
    <p>Sei sicuro di voler rimuovere le modifiche?</p>
  </div>
  <div className={styles.modalFooter}>
    <button onClick={() => setOpenDialog(false)}>Annulla</button>
    <button onClick={() => setOpenDialog(false)}>Conferma</button>
  </div>
</div>"
```

- Criterio: 2.4.6

Errore 7:

- Descrizione: il componente con classe `styles.modal`, rappresentante una modale, non ha un `aria-modal` per indicare che è una modale.

```
- Codice generatore: "<div className={styles.modal}>
  <div className={styles.modalHeading}>
    <span>Conferma la tua scelta</span>
    <div className={styles.closeButton} onClick={() => setOpenDialog(
      false)}>X</div>
  </div>
  <div className={styles.modalContent}>
    <p>Sei sicuro di voler rimuovere le modifiche?</p>
  </div>
  <div className={styles.modalFooter}>
```

```

        <button onClick={() => setOpenDialog(false)}>Annulla</button>
        <button onClick={() => setOpenDialog(false)}>Conferma</button>
      </div>
    </div>"
  - Criterio: 4.1.2
  Errore 8:
  - Descrizione: il titolo della modale non è rappresentato tramite un tag
    HTML di heading o con il role="heading con un aria-level".
  - Codice generatore: "<span>Conferma la tua scelta</span>"
  - Criterio: 1.3.1
</esempio>

```

```

<esempio>
  codice: ""
  import React from "react";
  export default function Form(props) {
    const { styles } = props;
    return (
      <>
        
        <div className={styles.formHeading}>
          
        </div>
        <div className={styles.form}>
          <label htmlFor="name">Nome</label>
          <input type="text" id="name" />
          <label>
            Cognome
            <input type="text" id="surname" />
          </label>
          <span className={styles.dateField}>Data di nascita</span>
          <input type="date" />
          <label>
            <input type="password" />
          </label>

          <div className={style.formFooter}>
            <div className={styles.button} onClick={() => {}}>Cancella</div>
            <button type="submit">Invia</button>
          </div>
        </div>
        <div className={styles.formFooterImage}>
          
        </div>
      </>
    )
  }
  ""

```

lista errori:

Errore 1:

- Descrizione: l'elemento con classe `styles.form` non ha il significato semantico corretto. Rappresenta un form, quindi dovrebbe avere `role="form"` oppure si dovrebbe usare il tag HTML semantico `<form>`.
- Codice generatore: `"<div className={styles.form}>"`
- Criterio: 4.1.2

Errore 2:

- Descrizione: la `input` con `type="date"` non ha un label associato. Dovrebbe avere un label associato tramite l'attributo `for` o tramite l'elemento `<label>`. In questo caso l'elemento da associare è l'elemento con classe `styles.dateField`.
- Codice generatore: `"Data di nascita
<input type="date" />"`
- Criterio: 3.3.2

Errore 3:

- Descrizione: l'elemento con classe `styles.button` non ha il significato semantico corretto. Rappresenta un pulsante, quindi dovrebbe avere `role="button"` oppure si dovrebbe usare il tag HTML semantico `<button>`.
- Codice generatore: `"<div className={styles.button} onClick={() => {}}>Cancella</div>"`
- Criterio: 4.1.2

Errore 4:

- Descrizione: l'elemento con classe `styles.button` ha un gestore di eventi (`onClick`), ma non è raggiungibile da tastiera. Dovrebbe avere un `tabindex >= 0`.
- Codice generatore: `"<div className={styles.button} onClick={() => {}}>Cancella</div>"`
- Criterio: 2.1.1

Errore 5:

- Descrizione: l'elemento con classe `styles.button`, poichè rappresenta il tasto `reset` di un form, dovrebbe avere un attributo `type="reset"`.
- Codice generatore: `"<div className={styles.button} onClick={() => {}}>Cancella</div>"`
- Criterio: 4.1.2

Errore 6:

- Descrizione: l'immagine non ha l'attributo `alt`. Dovrebbe avere un testo alternativo che descriva il contenuto dell'immagine oppure un `alt` vuoto se l'immagine è puramente decorativa.
- Codice generatore: `""`
- Criterio: 1.1.1

Errore 7:

- Descrizione: l'`input` con `type="password"` non ha un label associato. Dovrebbe avere un label associato inserendo del testo dentro il tag `<label>` che wrappa la `<input>`.
- Codice generatore: `"<label>
<input type="password" />
</label>"`

- Criterio: 3.3.2

</esempio>

<esempio>

```
codice: ""
import React from "react";
export default function Card(props) {
  const { styles } = props;
  return (
    <>
      
      <div className={styles.cardHeading}>
        
      </div>
      <div className={styles.card}>
        
        <div className={styles.cardContent}>
          <h3>{cardTitle}</h3>
          <p>{cardDescription}</p>
          <a href={cardLink}>Vai al dettaglio</a>
        </div>
      </div>
    </>
  )
}
```

lista errori:

Errore 1:

- Descrizione: l'immagine non ha l'attributo alt. Dovrebbe avere un testo alternativo che descriva il contenuto dell'immagine oppure un alt vuoto se l'immagine è puramente decorativa.
- Codice generatore: ""
- Criterio: 1.1.1

Errore 2:

- Descrizione: lo scopo del link non è comprensibile dal testo. Poiché esprime la visualizzazione di un dettaglio, si dovrebbe avere il collegamento, tramite aria-describedby, al titolo della card.
- Codice generatore: "<div className={styles.cardContent}>\n <h3>{cardTitle}</h3>\n <p>{cardDescription}</p>\n Vai al dettaglio\n</div>"
- Criterio: 2.4.4

</esempio>

Domanda: Dato il seguente codice elenca tutti gli errori di accessibilità. Nell'elenco assicurati di riportare il codice che ha generato l'errore.

Errori da non riportare:

- la semantica del codice è confusa e non chiarisce bene lo scopo della pagina.
- la struttura del codice è confusa e non è chiaro come i dati siano organizzati.
- l'elemento `<input>` non ha un `tabindex >= 0`.
- l'elemento non ha un id. Non segnalare la mancanza di id se non su elementi associati a un `<label>`.
- la proprietà "aria-labelledby" è stata utilizzata per associare un'etichetta a una sezione del codice, ma la label non è stata trovata
- manca l'attributo "id" dell'elemento
- l'etichetta dell'elemento "div" non è chiaramente visibile all'utente.
- Mancanza di un'etichetta che spiega la funzione del codice.
- Mancanza di un'etichetta che spiega il significato dei dati visualizzati
- utilizzare `className` non è un errore.
- mancanza di role sugli elementi `<button>`, `<input>`, `<select>`, `<textarea>`, `<a>`, `<p>`, ``, ``, ``.

Errori da ricercare:

- l'elemento `<input>` non è associato a un'etichetta.
- si ha un `onClick` su un elemento non interattivo.
- si ha un `onClick` su un elemento privo di significato semantico.
- si ha un `` senza l'attributo `alt`.
- si ha un elemento che dovrebbe avere un ruolo ARIA ma non ce l'ha.

Non commettere questi errori:

- non riportare soluzioni.
- non riportare suggerimenti di miglioramento.
- non riportare errori non legati all'accessibilità.
- non rispondere senza riportare il codice che ha generato l'errore.
- un elemento contenente JSX di fatto contiene del testo. Non è necessario aggiungere un `aria-label` o `aria-labelledby` se il contenuto è chiaro e rilevante.
- non riportare errori legati al CSS.
- non riportare errori legati al design.
- non riportare errori legati alla semantica del codice.
- non riportare errori legati alla struttura del codice.
- l'attributo `alt` vuoto è corretto. Non segnalare l'attributo `alt` vuoto come errore.
- non riportare errori legati all'assenza di un id.
- non riportare errori legati all'assenza di un `tabindex >= 0` se l'elemento non deve essere raggiungibile da tastiera.
- gli attributi `aria-labelledby`, `aria-describedby`, `aria-label` e `aria-labelledby` non sono necessari se il contenuto dell'elemento è chiaro e rilevante. In generale non sono mai necessari sugli headings e sui `<p>` se vengono utilizzati correttamente.
- un elemento `<input>` senza `tabindex >= 0` non è un errore di accessibilità. L'elemento `<input>` è sempre raggiungibile da tastiera.
- il tag HTML `<label>` va usato solo per associare un label a un `input`, `textarea` o `select`. `<label>` può contenere la `input`, `textarea` o `select` così da associare l'etichetta all'elemento, oppure può avere l'attributo `htmlFor` che punta all'id della `input`, `textarea` o `select`. `<label>` deve contenere del testo.
- l'elemento `<input>` può stare dentro l'elemento `<label>` per associare il label all'input.

- l'elemento `<input>` deve avere o un id associato a una `<label>` oppure deve essere contenuto all'interno di un `<label>`, questi i due esempi:
esempio 1: `<label> label per input <input/></label>`.
esempio 2: `<label htmlFor="id-input">label per input</label> <input id="id-input"/>`.
- non riportare errori del tipo "La struttura della pagina non è correttamente organizzata".
- non riportare errori del tipo "La navigazione del contenuto dei dettagli non è correttamente organizzata".
- non segnalare errori che non sono presenti nel codice fornito.
- non segnalare degli errori e poi inserire la frase "Non ci sono errori di accessibilità". Se hai segnalato degli errori allora ci sono degli errori.
- la frase "Utilizzo di elementi `<button>`" inutili per chiudere la finestra modale. Invece, è meglio utilizzare l'elemento HTML `<a>` per creare collegamenti che possono essere utilizzati come bottone di chiusura della finestra modale." è errata. Chiudere una modale è un'azione che deve essere fatto tramite un pulsante. `<a>` è un elemento che rimanda un utente in una nuova pagina.
- la frase "Utilizzo di elementi `<i>`" inutili come icone" è errata. `<i>` è un elemento che rappresenta del testo in corsivo. Se l'elemento `<i>` contiene un'icona, l'icona deve essere associata a un testo alternativo tramite `aria-label`, `aria-labelledby`, `aria-describedby` o `aria-describedby`.
- la frase "l'elemento `input` non ha un attributo `alt`." è errata. L'attributo `alt` su un `input` è errato, qualunque sia la motivazione.
- la frase "L'elemento `<input type="checkbox">` non ha un attributo `role` che lo identifica come `checkbox` o `radiobutton`" è errata. L'elemento `<input>` con `type="checkbox"` o `type="radio"` è già un `checkbox` o un `radiobutton`.
- la frase "L'elemento `<button>` non ha il `role="button"`" è errata. L'elemento `<button>` è già un pulsante e non necessita di `role="button"`.
- la frase "L'elemento `` non ha il `role="img"`" è errata. L'elemento `` è già un'immagine e non necessita di `role="img"`.
- la frase "L'elemento ha un nome di classe diverso da quello previsto" non rappresenta un errore di accessibilità. Gli elementi possono avere qualsiasi nome per la classe `css`.
- la frase "L'elemento `<input>` non ha un attributo `role`" è errata. L'elemento `<input>` non necessita di un attributo `role`.
- la frase "manca l'attributo `role`" all'elemento `input` per indicare che è un elemento di `input`" è errata. L'elemento `input` a cui si fa riferimento è un elemento HTML semantico e non necessita di un attributo `role`.
- la frase "Il codice non segue il criterio di utilizzare l'attributo `aria-label` o `aria-labelledby`" per gli elementi `img` che hanno una descrizione breve e non sono necessariamente visibili" è errata. Sui tag `` l'attributo per dare delle descrizioni è l'attributo `alt`.
- la frase "L'elemento `div` con classe `pageDetail__overlay`" non ha un attributo `role`". Questo elemento dovrebbe essere rimosso dall'albero di accessibilità" è errata. L'elemento `<div>` non necessita di un attributo `role` se non ha un ruolo specifico e se non ha un `tabindex >= 0` non è inserito dentro l'`accessibility tree`.
- non tutti gli elementi devono avere un ruolo ed essere raggiungibili da tastiera. Se un elemento ha un gestore di eventi (`onClick`) deve essere raggiungibile da tastiera ed avere un ruolo.

Ricorda:

- `<label> label per input <input/></label>` è un modo corretto per associare un'etichetta a un `input`.
- `<label htmlFor="id-input">label per input</label> <input id="id-input"/>` è un modo corretto per associare un'etichetta a un `input`.
- `<label> <input/> label per input</label>` è un modo corretto per associare un'etichetta a un

input.

- `<label htmlFor="id-input"> <input id="id-input"/> label per input</label>` non è un modo corretto per associare un'etichetta a un input. `<label>` non deve avere l'attributo `htmlFor` se contiene l'input.
- `<label> <input/></label>` non è un modo corretto per associare un'etichetta a un input. `<label>` deve avere del testo per poterlo associare alla input che contiene.
- `` è un modo corretto per dichiarare un'immagine puramente decorativa.
- `` è un modo corretto per dichiarare un'immagine con testo alternativo.
- `<div onClick=() ==> >` è un modo scorretto per dichiarare elemento con un gestore di eventi. Deve essere rappresentato o con il tag `<button>` oppure deve avere un `tabindex >= 0` e un `role="button"`.
- `<button onClick=() ==> >` è un modo corretto per dichiarare un pulsante con un gestore di eventi.
- `<button onClick=() ==> role="button">` è un modo scorretto per dichiarare un pulsante con un gestore di eventi. Il `role="button"` è ridondante.
- `<button onClick=() ==> ><i className="icon"></i></button>` è un modo scorretto per dichiarare un pulsante con un'icona. L'icona non rappresenta un testo valido per un pulsante. Il pulsante deve avere un testo alternativo tramite `aria-label`, `aria-labelledby`, `aria-describedby` o `aria-describedby` e `<i>` deve essere rimosso dall'accessibility tree.
- `<button onClick=() ==> ></button>` è un modo scorretto per dichiarare un pulsante con un'immagine. L'immagine non rappresenta un testo valido per un pulsante. Il pulsante deve avere un testo alternativo tramite `aria-label`, `aria-labelledby`, `aria-describedby` o `aria-describedby` e `` deve essere rimosso dall'accessibility tree.
- `aria-checked` è un attributo che va usato su elementi con `role="checkbox"` o `role="radio"` diversi da `<input>`. Non va aggiunto sull'elemento `<input>`.
- il `role="text"` è un ruolo ARIA che va usato su elementi che simulano un `<input type="text">` o una `<textarea>`. Non va usato su elementi la cui funzione è quella di mostrare del testo. - quando segnali la mancanza di un `role`, nella descrizione riporta anche quale `role` dovrebbe avere l'elemento.
- non puoi segnalare un errore che non esiste e suggerire di aggiungere un elemento che già esiste nel codice che ti è stato fornito.
- se ritieni che esista un tag HTML semantico che può sostituire un elemento generico, segnala l'errore e suggerisci la sostituzione con il tag HTML semantico. Se non sai cosa suggerire, non riportare l'errore.
- se trovi `onClick` su elementi che non hanno un ruolo e un `tabindex >= 0`, segnala l'errore e suggerisci di aggiungere un ruolo e un `tabindex >= 0`. Questo non vale nel caso di tag `<button>`.
- non dare suggerimenti di risoluzione.
- non segnalare l'assenza di `role="button"` sull'elemento `<button>`.

Il tuo unico compito è trovare gli errori di accessibilità web presenti nel codice fornito e riportarli in un elenco.

Se non ci sono errori di accessibilità rispondi con "Non ci sono errori di accessibilità" senza riportare ulteriori informazioni o suggerimenti.

La risposta deve essere in italiano e formattata così come segue:

"""

Errore 1:

- Descrizione: breve descrizione della problematica

- Codice generatore: riporta minimo 300 caratteri e massimo 500 del pezzo di codice che ha generato l'errore di accessibilità. Il codice deve essere riportato senza manipolazioni, deve corrispondere all'originale.
 - Criterio: numero criterio WCAG
- Errore 2:
- Descrizione: breve descrizione della problematica - Codice generatore: riporta minimo 300 caratteri e massimo 500 del pezzo di codice che ha generato l'errore di accessibilità. Il codice deve essere riportato senza manipolazioni, deve corrispondere all'originale.
 - Criterio: numero criterio WCAG
- ...
- """

Ricorda, gli unici campi ammessi nella lista sono "Descrizione", "Codice generatore" e "Criterio". Non riportare alcuna soluzione o suggerimento di miglioramento. Seguire il formato di risposta indicato è essenziale per la valutazione della risposta. Deve sempre essere presente il codice che ha generato l'errore di accessibilità.

Ricorda: devi seguire il formato di risposta indicato e deve sempre essere presente il codice che ha generato l'errore di accessibilità.

Ricorda: non devi riportare esempi, spiegazioni o suggerimenti di miglioramento. Riporta solo gli errori di accessibilità presenti nel codice fornito e la parte di codice che genera quell'errore.

```
codice: ""  
{{ codice }}  
"""
```

risposta:

User prompt - trasformazione lista di errori in JSON - il formato di risposta è un JSON

Data la seguente lista <list_to_transform> di errori di accessibilità web, trasforma <list_to_transform> in un json avente il formato <answer_template>.

```
<answer_template>  
[  
  {  
    descrizione_errore: breve descrizione della problematica,  
    codice_generatore: pezzo di codice che ha generato l'errore di  
      accessibilità,  
    criterio: numero criterio WCAG  
  },  
  ...  
]  
</answer_template>
```

La risposta deve essere in italiano e nel formato <answer_template>. Non riportare ragionamenti o spiegazioni.

<list_to_transform>

```
{{ lista }}
</list_to_transform>
```

User prompt - correzione errori rilevati - il formato di risposta è un JSON

<esempio>

```
JSON input: ""
[
  {
    descrizione_errore: "è stato inserito un onClick su un div privo di
      adeguato role. Dovrebbe avere role="button" oppure si dovrebbe usare il
      tag HTML semantico <button>",
    codice_generatore: "<div className={styles.navBut} onClick={() =>
      setOpenModal(true)}>< i className={\`icon-boat \${styles.icon}\` /><div
      className={styles.text}>premi</div></div> ",
    criterio: "4.1.2"
  },
  {
    descrizione_errore: "è stato inserito un onClick su un div non
      raggiungibile da tastiera.",
    codice_generatore: "<div className={styles.navBut} onClick={() =>
      setOpenModal(true)}>< i className={\`icon-boat \${styles.icon}\` /><div
      className={styles.text}>premi</div></div> ",
    criterio: "2.1.1"
  },
  {
    descrizione_errore: "l'icona è gestita tramite un font non leggibile
      dalle tecnologie assistive, quindi deve essere rimossa
      dall'accessibility tree o avere un testo alternativo tramite aria-label,
      aria-labelledby, aria-describedby o aria-describedby.",
    codice_generatore: "<div
      className={styles.navBut} onClick={() =>
      setOpenModal(true)}><i className={\`icon-boat \${styles.icon}\` /><div
      className={styles.text}>premi</div></div>",
    criterio: "1.1.1"
  }
]
""
```

risposta: ""

```
[
  {
    descrizione_errore: "è stato inserito un onClick su un div privo di
      adeguato role. Dovrebbe avere role="button" oppure si dovrebbe usare il
      tag HTML semantico <button>",
    codice_generatore: "<div className={styles.navBut} onClick={() =>
      setOpenModal(true)}>< i className={\`icon-boat \${styles.icon}\` /><div
      className={styles.text}>premi</div></div> ",
    criterio: "4.1.2",
```

```

    descrizione_risoluzione: "sostituiscilo con un tag <button> il div oppure
    aggiungi role='button' e tabindex=0",
    codice_fix: "<button className={styles.navBut} onClick={() => setOpenModal|
    (true)}>< i className={\`icon-boat \${styles.icon}\` /><div className=
    {styles.text}>premi</div></button>"
  },
  {
    descrizione_errore: "è stato inserito un onClick su un div non
    raggiungibile da tastiera.",
    codice_generatore: "<div className={styles.navBut} onClick={() =>
    setOpenModal(true)}>< i className={\`icon-boat \${styles.icon}\` /><div
    className={styles.text}>premi</div></div> ",
    criterio: "2.1.1",
    descrizione_risoluzione: "aggiungi tabindex=0 e role="button" al div o
    sostituiscilo con un tag <button>",
    codice_fix: "<div className={styles.navBut} role="button" tabIndex={0}
    onClick={() => setOpenModal(true)}>< i className={\`icon-boat \${styles.
    icon}\` /><div className={styles.text}>premi</div></div>"
  },
  {
    descrizione_errore: "l'icona è gestita tramite un font non leggibile
    dalle tecnologie assistive, quindi deve essere rimossa
    dall'accessibility tree o avere un testo alternativo tramite aria-label,
    aria-labelledby, aria-describedby o aria-describedby.",
    codice_generatore: "<div className={styles.navBut} onClick={() =>
    setOpenModal(true)}><i className={\`icon-boat \${styles.icon}\`} /><div
    className={styles.text}>premi</div></div>",
    criterio: "1.1.1",
    descrizione_risoluzione: "aggiungi un aria-label all'icona per dare un
    testo alternativo oppure rimuovi l'icona dall'accessibility tree
    tramite aria-hidden",
    codice_fix: "<div className={styles.navBut} onClick={() => setOpenModal(
    true)}><i className={\`icon-boat \${styles.icon}\`} aria-hidden /><div
    className={styles.text}>premi</div></div>"
  }
]
"""
</esempio>

<esempio>

JSON input: ""
[
  {
    descrizione_errore: "è stato inserito un onClick su un div privo di
    adeguato role. Dovrebbe avere role="button" oppure si dovrebbe usare il
    tag HTML semantico <button>",
    codice_generatore: "<div className={styles.closeButton} onClick={() =>
    setOpenDialog(false)}>X</div>",
  }
]

```

```

    criterio: "4.1.2"
  },
  {
    descrizione_errore: "è stato inserito un onClick su un div non
      raggiungibile da tastiera",
    codice_generatore: "<div className={styles.closeButton} onClick={() =>
      setOpenDialog(false)}>X</div>",
    criterio: "2.1.1"
  },
  {
    descrizione_errore: "il testo contenuto all'interno dell'elemento con
      handler da tastiera non è esplicativo del significato dell'elemento.
      Dovrebbe avere un testo alternativo tramite aria-label o
      aria-labelledby",
    codice_generatore: "<div className={styles.closeButton} onClick={() =>
      setOpenDialog(false)}>X</div>",
    criterio: "2.4.6"
  },
  {
    descrizione_errore: "il componente con classe styles.modal,
      rappresentante una modale, non ha un role="dialog" per indicare il suo
      ruolo all'interno della pagina",
    codice_generatore: "<div className={styles.modal}>
      <div className={styles.modalHeading}>
        <div className={styles.closeButton} onClick={() => setOpenDialog(
          false)}>X</div>
      </div>
      <div className={styles.modalContent}>
        <span>Conferma la tua scelta</span>
        <p>Sei sicuro di voler rimuovere le modifiche?</p>
      </div>
      <div className={styles.modalFooter}>
        <button onClick={() => setOpenDialog(false)}>Annulla</button>
        <button onClick={() => setOpenDialog(false)}>Conferma</button>
      </div>
    </div>",
    criterio: "4.1.2"
  },
  {
    descrizione_errore: "il componente con classe styles.modal,
      rappresentante una modale, non ha un aria-describedby per descrivere il
      contenuto della modale.",
    codice_generatore: "<div className={styles.modal}>
      <div className={styles.modalHeading}>
        <span>Conferma la tua scelta</span>
        <div className={styles.closeButton} onClick={() => setOpenDialog(
          false)}>X</div>
      </div>
      <div className={styles.modalContent}>

```

```

        <p>Sei sicuro di voler rimuovere le modifiche?</p>
    </div>
    <div className={styles.modalFooter}>
        <button onClick={() => setOpenDialog(false)}>Annulla</button>
        <button onClick={() => setOpenDialog(false)}>Conferma</button>
    </div>
    </div>",
    criterio: "2.4.6"
},
{
    descrizione_errore: "il componente con classe styles.modal,
    rappresentante una modale, non ha un aria-labelledby per descrivere il
    titolo della modale",
    codice_generatore: "<div className={styles.modal}>
        <div className={styles.modalHeading}>
            <span>Conferma la tua scelta</span>
            <div className={styles.closeButton} onClick={() => setOpenDialog(
            false)}>X</div>
        </div>
        <div className={styles.modalContent}>
            <p>Sei sicuro di voler rimuovere le modifiche?</p>
        </div>
        <div className={styles.modalFooter}>
            <button onClick={() => setOpenDialog(false)}>Annulla</button>
            <button onClick={() => setOpenDialog(false)}>Conferma</button>
        </div>
    </div>",
    criterio: "2.4.6"
},
{
    descrizione_errore: "il componente con classe styles.modal,
    rappresentante una modale, non ha un aria-modal per indicare che è una
    modale.",
    codice_generatore: "<div className={styles.modal}>
        <div className={styles.modalHeading}>
            <span>Conferma la tua scelta</span>
            <div className={styles.closeButton} onClick={() => setOpenDialog(
            false)}>X</div>
        </div>
        <div className={styles.modalContent}>
            <p>Sei sicuro di voler rimuovere le modifiche?</p>
        </div>
        <div className={styles.modalFooter}>
            <button onClick={() => setOpenDialog(false)}>Annulla</button>
            <button onClick={() => setOpenDialog(false)}>Conferma</button>
        </div>
    </div>",
    criterio: "4.1.2"
},

```

```

{
  descrizione_errore: "il titolo della modale non è rappresentanto tramite
    un tag HTML di heading o con il role="heading con un aria-level"",
  codice_generatore: "<span>Conferma la tua scelta</span>",
  criterio: "1.3.1"
}
]
"""

risposta: ""
[
  {
    descrizione_errore: "è stato inserito un onClick su un div privo di
      adeguato role. Dovrebbe avere role="button" oppure si dovrebbe usare il
      tag HTML semantico <button>",
    codice_generatore: "<div className={styles.closeButton} onClick={() =>
      setOpenDialog(false)}>X</div>",
    criterio: "4.1.2",
    descrizione_risoluzione: "sostituiscilo con un tag <button> il div oppure
      aggiungi role='button' e tabIndex=0",
    codice_fix: "<button className={styles.closeButton} onClick={() =>
      setOpenDialog(false)}>X</button>"
  },
  {
    descrizione_errore: "è stato inserito un onClick su un div non
      raggiungibile da tastiera",
    codice_generatore: "<div className={styles.closeButton} onClick={() =>
      setOpenDialog(false)}>X</div>",
    criterio: "2.1.1",
    descrizione_risoluzione: "aggiungi tabIndex=0 e role="button" al div o
      sostituiscilo con un tag <button>",
    codice_fix: "<div className={styles.closeButton} role="button" tabIndex=
      {0} onClick={() => setOpenDialog(false)}>X</div>"
  },
  {
    descrizione_errore: "il testo contenuto all'interno dell'elemento con
      handler da tastiera non è esplicativo del significato dell'elemento.
      Dovrebbe avere un testo alternativo tramite aria-label o
      aria-labelledby",
    codice_generatore: "<div className={styles.closeButton} onClick={() =>
      setOpenDialog(false)}>X</div>",
    criterio: "2.4.6",
    descrizione_risoluzione: "aggiungi un aria-label al div per dare un testo
      alternativo",
    codice_fix: "<div className={styles.closeButton} onClick={() =>
      setOpenDialog(false)} aria-label="chiudi">X</div>"
  },
  {
    descrizione_errore: "il componente con classe styles.modal,

```

```

    rappresentante una modale, non ha un role="dialog" per indicare il suo
    ruolo all'interno della pagina",
    codice_generatore: "<div className={styles.modal}>
      <div className={styles.modalHeading}>
        <div className={styles.closeButton} onClick={() => setOpenDialog(
          false)}>X</div>
      </div>
      <div className={styles.modalContent}>
        <span>Conferma la tua scelta</span>
        <p>Sei sicuro di voler rimuovere le modifiche?</p>
      </div>
      <div className={styles.modalFooter}>
        <button onClick={() => setOpenDialog(false)}>Annulla</button>
        <button onClick={() => setOpenDialog(false)}>Conferma</button>
      </div>
    </div>",
    criterio: "4.1.2",
    descrizione_risoluzione: "aggiungi role='dialog' al div",
    codice_fix: "<div className={styles.modal} role='dialog'>
      <div className={styles.modalHeading}>
        <div className={styles.closeButton} onClick={() => setOpenDialog(
          false)}>X</div>
      </div>
      <div className={styles.modalContent}>
        <span>Conferma la tua scelta</span>
        <p>Sei sicuro di voler rimuovere le modifiche?</p>
      </div>
      <div className={styles.modalFooter}>
        <button onClick={() => setOpenDialog(false)}>Annulla</button>
        <button onClick={() => setOpenDialog(false)}>Conferma</button>
      </div>
    </div>"
  },
  {
    descrizione_errore: "il componente con classe styles.modal,
    rappresentante una modale, non ha un aria-describedby per descrivere il
    contenuto della modale.",
    codice_generatore: "<div className={styles.modal}>
      <div className={styles.modalHeading}>
        <span>Conferma la tua scelta</span>
        <div className={styles.closeButton} onClick={() => setOpenDialog(
          false)}>X</div>
      </div>
      <div className={styles.modalContent}>
        <p>Sei sicuro di voler rimuovere le modifiche?</p>
      </div>
      <div className={styles.modalFooter}>
        <button onClick={() => setOpenDialog(false)}>Annulla</button>
        <button onClick={() => setOpenDialog(false)}>Conferma</button>
      </div>
    </div>
  }

```

```

        </div>
    </div>",
    criterio: "2.4.6",
    descrizione_risoluzione: "aggiungi un aria-describedby al div per
    descrivere il contenuto della modale",
    codice_fix: "<div className={styles.modal} aria-describedby='modalContent'>
    <div className={styles.modalHeading}>
    <span>Conferma la tua scelta</span>
    <div className={styles.closeButton} onClick={() => setOpenDialog(
    false)}>X</div>
    </div>
    <div className={styles.modalContent} id='modalContent'>
    <p>Sei sicuro di voler rimuovere le modifiche?</p>
    </div>
    <div className={styles.modalFooter}>
    <button onClick={() => setOpenDialog(false)}>Annulla</button>
    <button onClick={() => setOpenDialog(false)}>Conferma</button>
    </div>
    </div>"
},
{
    descrizione_errore: "il componente con classe styles.modal,
    rappresentante una modale, non ha un aria-labelledby per descrivere il
    titolo della modale",
    codice_generatore: "<div className={styles.modal}>
    <div className={styles.modalHeading}>
    <span>Conferma la tua scelta</span>
    <div className={styles.closeButton} onClick={() => setOpenDialog(
    false)}>X</div>
    </div>
    <div className={styles.modalContent}>
    <p>Sei sicuro di voler rimuovere le modifiche?</p>
    </div>
    <div className={styles.modalFooter}>
    <button onClick={() => setOpenDialog(false)}>Annulla</button>
    <button onClick={() => setOpenDialog(false)}>Conferma</button>
    </div>
    </div>",
    criterio: "2.4.6",
    descrizione_risoluzione: "aggiungi un aria-labelledby al div per
    descrivere il titolo della modale",
    codice_fix: "<div className={styles.modal} aria-labelledby='modalTitle'>
    <div className={styles.modalHeading}>
    <span id='modalTitle'>Conferma la tua scelta</span>
    <div className={styles.closeButton} onClick={() => setOpenDialog(
    false)}>X</div>
    </div>
    <div className={styles.modalContent}>
    <p>Sei sicuro di voler rimuovere le modifiche?</p>

```

```

        </div>
        <div className={styles.modalFooter}>
          <button onClick={() => setOpenDialog(false)}>Annulla</button>
          <button onClick={() => setOpenDialog(false)}>Conferma</button>
        </div>
      </div>"
    },
    {
      descrizione_errore: "il componente con classe styles.modal,
        rappresentante una modale, non ha un aria-modal per indicare che è una
        modale.",
      codice_generatore: "<div className={styles.modal}>
        <div className={styles.modalHeading}>
          <span>Conferma la tua scelta</span>
          <div className={styles.closeButton} onClick={() => setOpenDialog(
            false)}>X</div>
        </div>
        <div className={styles.modalContent}>
          <p>Sei sicuro di voler rimuovere le modifiche?</p>
        </div>
        <div className={styles.modalFooter}>
          <button onClick={() => setOpenDialog(false)}>Annulla</button>
          <button onClick={() => setOpenDialog(false)}>Conferma</button>
        </div>
      </div>",
      criterio: "4.1.2",
      descrizione_risoluzione: "aggiungi aria-modal al div per indicare che è
        una modale",
      codice_fix: "<div className={styles.modal} aria-modal='true'>
        <div className={styles.modalHeading}>
          <span>Conferma la tua scelta</span>
          <div className={styles.closeButton} onClick={() => setOpenDialog(
            false)}>X</div>
        </div>
        <div className={styles.modalContent}>
          <p>Sei sicuro di voler rimuovere le modifiche?</p>
        </div>
        <div className={styles.modalFooter}>
          <button onClick={() => setOpenDialog(false)}>Annulla</button>
          <button onClick={() => setOpenDialog(false)}>Conferma</button>
        </div>
      </div>"
    },
    {
      descrizione_errore: "il titolo della modale non è rappresentanto tramite
        un tag HTML di heading o con il role='heading con un aria-level'",
      codice_generatore: "<span>Conferma la tua scelta</span>",
      criterio: "1.3.1",
      descrizione_risoluzione: "sostituisci il tag span con un tag h1",
    }
  ],
  {
    descrizione_errore: "il titolo della modale non è rappresentanto tramite
      un tag HTML di heading o con il role='heading con un aria-level'",
    codice_generatore: "<span>Conferma la tua scelta</span>",
    criterio: "1.3.1",
    descrizione_risoluzione: "sostituisci il tag span con un tag h1",
  }

```

```

        codice_fix: "<h1>Conferma la tua scelta</h1>"
    }
]
"""
</esempio>

<esempio>

JSON input: """
[
  {
    descrizione_errore: "l'elemento con classe styles.form non ha il
      significato semantico corretto. Rappresenta un form, quindi dovrebbe
      avere role=\"form\" oppure si dovrebbe usare il tag HTML semantico <form>.",
    codice_generatore: "<div className={styles.form}>",
    criterio: "4.1.2"
  },
  {
    descrizione_errore: "la input con type=\"date\" non ha un label associato.
      Dovrebbe avere un label associato tramite l'attributo for o tramite
      l'elemento <label>. In questo caso l'elemento da associare è l'elemento
      con classe styles.dateField",
    codice_generatore: "<span className={styles.dateField}>Data di nascita
      </span>
      <input type=\"date\" />",
    criterio: "3.3.2"
  },
  {
    descrizione_errore: "l'elemento con classe styles.button non ha il
      significato semantico corretto. Rappresenta un pulsante, quindi
      dovrebbe avere role=\"button\" oppure si
      dovrebbe usare il tag HTML semantico <button>.",
    codice_generatore: "<div className={styles.button} onClick={() => {}}>
      Cancella</div>",
    criterio: "4.1.2"
  },
  {
    descrizione_errore: "l'elemento con classe styles.button ha un gestore di
      eventi (onClick), ma non è raggiungibile da tastiera. Dovrebbe avere un
      tabindex >= 0.",
    codice_generatore: "<div className={styles.button} onClick={() => {}}>
      Cancella</div>",
    criterio: "2.1.1"
  },
  {
    descrizione_errore: "l'elemento con classe styles.button, poichè
      rappresenta il tasto reset di un form, dovrebbe avere un attributo type
      =\"reset\".",

```

```

codice_generatore: "<div className={styles.button} onClick={() => {}}>
  Cancella</div>",
criterio: "4.1.2"
},
{
  descrizione_errore: "l'immagine non ha l'attributo alt. Dovrebbe avere un
    testo alternativo che descriva il contenuto dell'immagine oppure un alt
    vuoto se l'immagine è puramente decorativa.",
  codice_generatore: "<img src='footer-logo.png' />",
  criterio: "1.1.1"
},
{
  descrizione_errore: "l'input con type='password' non ha un label
    associato. Dovrebbe avere un label associato inserendo del testo dentro
    il tag <label> che wrappa la <input>.",
  codice_generatore: "<label>
    <input type='password' />
  </label>",
  criterio: "3.3.2"
}
]
"""

risposta: """
[
  {
    descrizione_errore: "l'elemento con classe styles.form non ha il
      significato semantico corretto. Rappresenta un form, quindi dovrebbe
      avere role='form' oppure si dovrebbe usare il tag HTML semantico <form>
      .",
    codice_generatore: "<div className={styles.form}>",
    criterio: "4.1.2",
    descrizione_risoluzione: "sostituiscilo con un tag <form> il div oppure
      aggiungi role='form'",
    codice_fix: "<form className={styles.form}>"
  },
  {
    descrizione_errore: "la input con type='date' non ha un label associato.
      Dovrebbe avere un label associato tramite l'attributo for o tramite
      l'elemento <label>. In questo caso l'elemento da associare è l'elemento
      con classe styles.dateField",
    codice_generatore: "<span className={styles.dateField}>Data di nascita</
      span>
      <input type='date' />",
    criterio: "3.3.2",
    descrizione_risoluzione: "aggiungi un label associato all'input",
    codice_fix: "<label for='dateField' className={styles.dateField}>Data di
      nascita</label>
      <input type='date' id='dateField' />"
  }
]
"""

```

```

},
{
  descrizione_errore: "l'elemento con classe styles.button non ha il
    significato semantico corretto. Rappresenta un pulsante, quindi
    dovrebbe avere role='button' oppure si dovrebbe usare il tag HTML
    semantico <button>.",
  codice_generatore: "<div className={styles.button} onClick={() => {}}>
    Cancellare</div>",
  criterio: "4.1.2",
  descrizione_risoluzione: "sostituiscilo con un tag <button> il div oppure
    aggiungi role='button' e tabIndex=0",
  codice_fix: "<button className={styles.button} onClick={() => {}}>
    Cancellare</button>"
},
{
  descrizione_errore: "l'elemento con classe styles.button ha un gestore di
    eventi (onClick), ma non è raggiungibile da tastiera. Dovrebbe avere un
    tabIndex >= 0.",
  codice_generatore: "<div className={styles.button} onClick={() => {}}>
    Cancellare</div>",
  criterio: "2.1.1",
  descrizione_risoluzione: "aggiungi tabIndex=0 al div",
  codice_fix: "<div className={styles.button} onClick={() => {}} tabIndex=
    {0}>Cancellare</div>"
},
{
  descrizione_errore: "l'elemento con classe styles.button, poiché
    rappresenta il tasto reset di un form, dovrebbe avere un attributo type
    ='reset'.",
  codice_generatore: "<div className={styles.button} onClick={() => {}}>
    Cancellare</div>",
  criterio: "4.1.2",
  descrizione_risoluzione: "aggiungi type='reset' al button",
  codice_fix: "<div className={styles.button} onClick={() => {}} type
    ='reset'>Cancellare</div>"
},
{
  descrizione_errore: "l'immagine non ha l'attributo alt. Dovrebbe avere un
    testo alternativo che descriva il contenuto dell'immagine oppure un alt
    vuoto se l'immagine è puramente decorativa.",
  codice_generatore: "<img src='footer-logo.png' />",
  criterio: "1.1.1",
  descrizione_risoluzione: "aggiungi un alt all'immagine",
  codice_fix: "<img src='footer-logo.png' alt='logo footer' />"
},
{
  descrizione_errore: "l'input con type='password' non ha un label
    associato. Dovrebbe avere un label associato inserito del testo dentro
    il tag <label> che wrappa la <input>.",

```

```
codice_generatore: "<label>
  <input type="password" />
</label>",
criterio: "3.3.2",
descrizione_risoluzione: "aggiungi del testo al label",
codice_fix: "<label>Password
  <input type="password" />
</label>"
}
]
"""

</esempio>

<esempio>

JSON input: """
[
  {
    descrizione_errore: "l'immagine non ha l'attributo alt. Dovrebbe avere un
      testo alternativo che descriva il contenuto dell'immagine oppure un alt
      vuoto se l'immagine è puramente decorativa.",
    codice_generatore: "",
    criterio: "1.1.1"
  },
  {
    descrizione_errore: "lo scopo del link non è comprensibile dal testo.
      Poichè esprimere la visualizzazione di un dettaglio, si dovrebbe avere il
      collegamento, tramite aria-describedby, al titolo della card",
    codice_generatore: "<div className={styles.cardContent}>
      <h3>{cardTitle}</h3>
      <p>{cardDescription}</p>
      <a href={cardLink}>Vai al dettaglio</a>
      </div>",
    criterio: "2.4.4"
  }
]
"""

risposta: """
[
  {
    descrizione_errore: "l'immagine non ha l'attributo alt. Dovrebbe avere un
      testo alternativo che descriva il contenuto dell'immagine oppure un alt
      vuoto se l'immagine è puramente decorativa.",
    codice_generatore: "",
    criterio: "1.1.1",
    descrizione_risoluzione: "aggiungi un alt all'immagine",
    codice_fix: ""
  }
]
"""
```

```

    },
    {
      descrizione_errore: "lo scopo del link non è comprensibile dal testo.
        Poichè esprime la visualizzazione di un dettaglio, si dovrebbe avere il
        collegamento, tramite aria-describedby, al titolo della card",
      codice_generatore: "<div className={styles.cardContent}>
        <h3>{cardTitle}</h3>
        <p>{cardDescription}</p>
        <a href={cardLink}>Vai al dettaglio</a>
      </div>",
      criterio: "2.4.4",
      descrizione_risoluzione: "aggiungi un aria-describedby al link per
        collegarlo al titolo della card",
      codice_fix: "<div className={styles.cardContent}>
        <h3 id='cardTitle'>{cardTitle}</h3>
        <p>{cardDescription}</p>
        <a href={cardLink} aria-describedby='cardTitle'>Vai al dettaglio</a>
      </div>"
    }
  ]
}
"""
</esempio>

```

Domanda: Ti verrà fornito un JSON contenente del codice con degli errori di accessibilità. Il tuo compito è correggere gli errori di accessibilità che ti vengono forniti tramite JSON. Analizza l'errore e fornisci la soluzione migliore per correggerlo. Quando suggerisci codice, assicurati che sia HTML semantico, che sia operabile utilizzando la tastiera, che segua le tecniche sufficienti di WCAG 2.2 e la Guida alle pratiche di autore ARIA e i relativi design pattern.

Dovrai restituire un JSON così formattato riportante le correzioni degli errori di accessibilità riscontrati:

```

"""
[
  {
    "descrizione_errore": campo "descrizione_errore" del JSON,
    "codice_generatore": campo "codice_generatore" del JSON,
    "criterio": campo "criterio" del JSON,
    "descrizione_risoluzione": breve descrizione di una o più possibili
      soluzioni,
    "codice_fix": codice riportante l'implementazione della soluzione migliore
  }
]
"""

```

Fai attenzione, i campi "descrizione_errore", "codice_generatore", "criterio", "descrizione_risoluzione" e "codice_fix" del JSON sono campi obbligatori.

I campi "descrizione_errore", "codice_generatore" e "criterio" devono essere riportati senza manipolazioni, devono corrispondere all'originale.

La correzione dell'errore deve essere riportata nel campo "codice_fix".

Ricorda, se la descrizione_errore non indica che role assegnare all'elemento e in codice_generatore l'elemento ha un gestore di eventi, aggiungi tabindex=0 e role="button" all'elemento. Questo non è valido per il tag <button>.

Restituisci solo il JSON.

Questo è il JSON che contiene gli errori che devi correggere: {{ json }}

risposta:

Bibliografia

- [1] Agid. *Accessibilità dei siti web e applicazioni mobili dei soggetti privati*. 2022. URL: <https://trasparenza.agid.gov.it/download/6003.html>.
- [2] Agid. *Errori monitoraggio approfondito*. URL: <https://accessibilita.agid.gov.it/errori-approfondito>.
- [3] Agid. *Errori monitoraggio semplificato*. URL: <https://accessibilita.agid.gov.it/errori-semplificato>.
- [4] Agid. *Linee guida accessibilità - Privati*. URL: <https://www.agid.gov.it/it/design-servizi/accessibilita/linee-guida-accessibilita-privati>.
- [5] AWS. *Cosa sono i modelli linguistici di grandi dimensioni (LLM)?* URL: <https://aws.amazon.com/it/what-is/large-language-model>.
- [6] AWS. *Cos'è la RAG (Retrieval-Augmented Generation)?* URL: <https://aws.amazon.com/it/what-is/retrieval-augmented-generation/>.
- [7] AI4BUSINESS Nicoletta Boldrini. *Reti neurali: cosa sono e a cosa servono*. URL: <https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/>.
- [8] World Wide Web Consortium. *Accessibility, Usability, and Inclusion*. URL: <https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/>.
- [9] World Wide Web Consortium. *Understanding WCAG*. URL: <https://www.w3.org/WAI/WCAG22/Understanding/intro>.
- [10] World Wide Web Consortium. *Using alt attributes on img elements*. URL: <https://www.w3.org/WAI/WCAG22/Techniques/html/H37.html>.
- [11] World Wide Web Consortium. *Using null alt text and no title attribute on img elements for images that assistive technology should ignore*. URL: <https://www.w3.org/WAI/WCAG22/Techniques/html/H67.html>.
- [12] World Wide Web Consortium. *Web Accessibility Initiative*. URL: <https://www.w3.org/WAI/>.
- [13] World Wide Web Consortium. *Web Content Accessibility Guidelines*. URL: <https://www.w3.org/TR/WCAG22/>.
- [14] TechTarget Lev Craig. *What is fine-tuning?* URL: <https://www.techtarget.com/searchenterpriseai/definition/fine-tuning>.
- [15] DAIR.AI. *Guida al Prompt Engineering*. URL: <https://www.promptingguide.ai/it>.
- [16] DAIR.AI. *Prompt Chain-of-Thought*. URL: <https://www.promptingguide.ai/it/techniques/cot>.

-
- [17] DAIR.AI. *Suggerimenti generali per la progettazione di Prompt*. URL: <https://www.promptingguide.ai/it/introduction/tips>.
- [18] Ministero delle Imprese e del Made in Italy. *Accessibilità e Usabilità*. URL: <https://www.mimit.gov.it/it/accessibilita>.
- [19] Izzeddin Gur Ofir Nachum Yingjie Miao Mustafa Safdari Austin Huang Aakanksha Chowdhery Sharan Narang Noah Fiedel Aleksandra Faust. «Understanding HTML with Large Language Models». In: (2023).
- [20] Flairbit. *LLM: Potenzialità e le Sfide dei Modelli di Linguaggio Avanzati*. URL: <https://flairbit.com/llm-potenzialita-e-le-sfide-dei-modelli-di-linguaggio-avanzati/>.
- [21] Chongzhou Fang Ning Miao Shaurya Srivastav Jialin Liu Ruoyu Zhang Ruijie Fang Asmita Ryan Tsang Najmeh Nazari Han Wang Houman Homayoun. «Large Language Models for Code Analysis: Do LLMs Really Do Their Job?». In: (2024).
- [22] Andrei Kucharavy Octave Plancherel Valentin Mulder Alain Mermoud Vincent Lenders. *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*. Springer, 2024.
- [23] Microsoft. *Visual Studio Code*. URL: <https://code.visualstudio.com/>.
- [24] OMS. *ICF versione breve - Classificazione Internazionale del Funzionamento, della Disabilità e della Salute*. 2001. URL: https://iris.who.int/bitstream/handle/10665/42417/9788879466288_ita.pdf?sequence=4.
- [25] OpenAI. *Best practices for prompt engineering with the OpenAI API*. URL: <https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api>.
- [26] OpenAI. *Prompt engineering*. URL: <https://platform.openai.com/docs/guides/prompt-engineering>.
- [27] Juan-Miguel López-Gil Juanan Pereira. «Turning manual web accessibility success criteria into automatic: an LLM-based approach». In: (2024).
- [28] Toloka. *Pre-training in LLM Development*. URL: <https://toloka.ai/blog/pre-training-in-llm-development/>.
- [29] AI4BUSINESS Andrea Viliotti. *Transformer, cosa sono e come stanno rivoluzionando l'AI*. URL: <https://www.ai4business.it/intelligenza-artificiale/transformer-cosa-sono-e-come-stanno-rivoluzionando-lai/>.
- [30] Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma Brian Ichter Fei Xia Ed Chi Quoc Le Denny Zhou. «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models». In: (2022).