Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

A.a. 2025/2026

Sessione di laurea Settembre 2026

Constitutional Web Agent: Refinement Iterativo per l'ottimizzazione delle UI

Relatori:

Luigi De Russis

Tommaso Calò

Candidato:

Luca Vigneri

Ringraziamenti

Desidero ringraziare innanzitutto il mio relatore, il Professor Luigi De Russis, per la costante disponibilità, la fiducia accordatami e gli stimolanti riscontri offerti durante tutte le fasi di questo lavoro.

Ringrazio Tommaso Calò, per la preziosa guida quotidiana, il confronto costruttivo e il supporto continuo che non ha mai fatto mancare lungo questo percorso. La sua pazienza, la sua costante presenza e la prontezza nel condividere idee e soluzioni pratiche hanno rappresentato un punto di riferimento essenziale per la riuscita di questo lavoro.

Indice

Elenco delle figure VI

1 Introduzione 1
1.1 Obiettivo della tesi . . . 3
1.2 Struttura della tesi . . . 4

2 Background e Lavori Correlati 5
2.1 Background . . . 6
2.1.1 Generazione di siti web a partire da direttive in linguaggio naturale . . . 6
2.1.2 Automazione della valutazione funzionale dei siti web . . . 7
2.1.3 RLAIF vs RLHF . . . 8
2.2 Lavori Correlati . . . 9
2.2.1 Benchmark per la valutazione del codice generato . . . 9
2.2.2 Architetture basate su agenti per lo sviluppo frontend . . . 9
2.2.3 Constitutional AI . . . 10

3 Metodologie e Setup Sperimentale 11
3.1 Introduzione del Framework . . . 11
3.2 Motivazioni e contesto teorico . . . 12
3.3 Dataset utilizzati . . . 14
3.4 Architettura della Pipeline proposta . . . 14
3.4.1 Fase generativa del codice . . . 15
3.4.2 Generazione dei test di controllo dello stato . . . 16
3.4.3 Valutazione funzionale tramite Visual-Language Model . . . 17
3.4.4 Autocritica basata su principi di HCI . . . 18
3.4.5 Ciclo Iterativo di Refinement e Logiche di Promozione . . . 19
3.5 Implementazione Tecnica e Gestione dei Dati . . . 20
3.5.1 Struttura del File System e Salvataggio degli Snapshot . . . 21
3.5.2 Tracking dello Stato Iterativo . . . 21
3.6 HCI Constitution . . . 22

3.6.1 Macroarea 1: Feedback di Stato e Isolamento Cognitivo . . . 31
3.6.2 Macroarea 2: Prevenzione degli Errori e Controllo 32
3.6.3 Macroarea 3: Affordance Visiva e Precisione Motoria 32

4 Analisi di Affidabilità e Validazione Metodologica 34
4.1 Validazione della Permissività dei Test Playwright 35
4.2 Correzione bug indipendente vs. Applicazione della HCI Constitution 37
4.3 Mitigazione del Reward Hacking 39
4.3.1 Deviazione Standard e Affidabilità Intra-Judge 40
4.3.2 Blind A/B Preference Test tramite giudici LLM 42
4.3.3 Validazione del Blind A/B Preference Test tramite Panel Umano 43
4.3.4 Bulk A/B Preference Test in modalità Pairwise 44
4.4 Validazione tramite Baseline Best-of-N vs. Refinement Iterativo . . 47

5 Risultati sperimentali e Valutazioni 49
5.1 Setup degli esperimenti e dataset utilizzati 49
5.2 Risultati dell'applicazione del Refinement Iterativo 50
5.3 Costo computazionale e guadagno TSR per GPU-h 52
5.4 Convergence Curve e identificazione dell'Early Stopping 53

6 Discussione e Conclusioni 56
6.1 Sintesi dei contributi e dell'efficacia lavoro 56
6.2 Limitazioni del lavoro 57
6.3 Sviluppi futuri 59

Bibliografia 60

Elenco delle figure

3.1 Struttura schematizzata della fase di Setup 15
3.2 Struttura schematizzata della fase di Refinement 15
3.3 UI prima e dopo l'applicazione dei principi 3 (Fitt's Law) e 5 (Visual Boundaries and Affordance). 28
3.4 UI prima dell'applicazione del principio 2 (Error Recognition adn Recovery). 29
3.5 UI dopo l'applicazione del principio 2 (Error Recognition adn Recovery). 30

4.1 Andamento Score Costituzionale versione Base 40
4.2 Andamento Score Costituzionale versione Best 40

5.1 Confronto Task Success Rate Base Code - Best Code 51
5.2 Confronto Score Costituzionale Base Code - Best Code 52
5.3 Andamento Task Success Rate medio 55
5.4 Andamento Score Costituzionale medio 55

Capitolo 1

Introduzione

La rapida ascesa ed evoluzione dei Large Language Models (LLMs) ha fatto sì di modificare in modo sostanziale il campo dell'ingegneria del software.

Nelle loro prime applicazioni, questi modelli venivano impiegati principalmente come assistenti alla programmazione, limitandosi a compiti di autocompletamento o alla generazione di singole funzioni a partire da brevi descrizioni in linguaggio naturale [1]. Tuttavia, il miglioramento delle loro capacità legato ad una sempre maggiore comprensione del contesto ha consentito di spostare l'attenzione della ricerca verso architetture sempre più complesse ed autonome, note come sistemi agentici. In queste strutture, i modelli non sono più dei generatori passivi ma fungono da veri e propri centri decisionali: elaborano piani complessi, utilizzano strumenti esterni e interagiscono in modo iterativo con l'ambiente di sviluppo [2, 3].

Nel campo dell'ingegneria del software i sistemi agentici vengono progettati per la risoluzione di task a livello di intera repository (come, ad esempio, SWE-agent) [4]. Lavorare a questo livello significa andare oltre la mera generazione di singoli frammenti di codice o file isolati. Un progetto software, infatti, è composto da un'intricata rete di componenti: molteplici file interconnessi, gerarchia di directory, file di configurazione e dipendenza esterne.

Affrontare problemi a livello di repository richiede dunque che il sistema sia in grado di esplorare e navigare in modo autonomo il codebase, comprendendo il flusso di esecuzione e le relazioni tra i diversi moduli (ad esempio, l'interazione tra i componenti frontend e la logica sottostante) e applicando modifiche correlate su più file contemporaneamente.

Tuttavia, gli attuali sistemi agentici utilizzati per lo sviluppo o la valutazione di siti web (come, ad esempio, WebArena [5] e OpenHands [6]) basano le loro valutazioni su feedback funzionali o legati alla corretta esecuzione sintattica del codice, trascurando spesso l'analisi visiva e la valutazione euristica legata alla User Experience (UX) .

Questa focalizzazione sulle metriche funzionali rappresenta una criticità signficiativa nel dominio delle interfacce utente. La correttezza algoritmica e sintattica è una condizione necessaria ma non sufficiente a garantire la qualità del prodotto finale [7]. Un codice formalmente perfetto può comunque generare un'applicazione web con gravi difetti visivi o strutturali: elementi della pagina che si sovrappongono, contrasti cromatici o layout non responsivi. Trascurando questi aspetti, gli agenti rischiano di produre del software che, pur essendo tecnicamente funzionante, risulta di fatto inaccessibile per l'utente finale.

Per colmare questa lacuna, studi recenti hanno introdotto architetture basate su un approccio a feedback multi-livello, di cui il benchmark ArtifactsBench e il framework WebGen-Agent rappresentano elementi di rilievo [8, 9]. Con feedback multi-livello si intende il superamento della singola metrica di successo legata all'esecuzione del codice, estendendo la valutazione su dimensioni necessarie per garantire una buona UX.

Oltre a validare la funzionalità logica, queste metodologie introducono innanzitutto un livello di valutazione visiva. Questo è reso possibile dall'impiego dei Visual-Language Model (VLM), ovvero modelli multimodali capaci di elaborare oltre ad input testuali anche immagini [10]. Analizzando screenshot statici dell'interfaccia generata, il VLM è in grado di quantificare la qualità della User Interface (UI) estraendo metriche oggettive, assegnando punteggi o fornendo critiche testuali mirate sul layout e sull'estetica. A questo si aggiunge un livello di valutazione interattiva.

Nello specifico, architetture come WebGen-Agent [4] hanno dimostrato la fattibilità di coordinare all'interno dello stesso ciclo di sviluppo e miglioramento iterativo tre attori distinti: un LLM incaricato di generare il codice sorgente, un VLM per giudicarne l'impatto visivo e, infine, un agente per interfacce grafiche (GUI agent) [11] dedicato a simulare l'interazione umana, fornendo così un feedback sul reale comportamento dell'interfaccia.

Parallelamente all'evoluzione di questi flussi di miglioramento iterativo, la letteratura recente ha esplorato paradigmi in cui l'Intelligenza Artificiale supervisiona e migliora l'operato di altri modelli AI, riducendo drasticamente la necessità di costosi interventi di supervisione umana [12, 13]. Un punto di riferimento in questo ambito è lo studio sulla Constitutional AI proposto da Anthropic [13], il quale ha dimostrato come definendo un set di istruzioni o principi fondamentali (Costituzione), sia possibile addestrare un sistema ad autovalutarsi (Self-Critique) e automigliorarsi (Self-Learning). Questo processo genera modelli in grado di fornire risposte più helpful e harmless senza dipendere dal massivo Reinforcement Learning from Human Feedback (RLHF) ma dipendendo da un Reinforcement Learning from AI Feedback (RLAIF) [12].

Alla luce di queste premesse in questo lavoro si vuole definire un nuovo framework metodologico capace di colmare il divario tra la sola validazione sintattica e la reale

usabilità delle interfacce. Risulta quindi fondamentale progettare un'architettura in cui i sistemi generativi incorporino e rispettino nativamente i principi della Human-Computer Interaction (HCI), per elevare le caratteristiche di usabilità e accessibilità delle User Interaces (UI) attraverso un processo di generazione del codice web e di miglioramento dovuto a feedback forniti da un'AI.

1.1 Obiettivo della tesi

L'obiettivo primario di questo lavoro è la progettazione, l'implementazione e la validazione rigorosa di un Constitutional Web Agent, un framework automatizzato per la creazione e l'ottimizzazione iterativa del codice frontend di siti web. Il sistema si distacca dai tradizionali approcci di bug fixing per concentrarsi esclusivamente sul miglioramento architetturale, estetico, di usabilità e accessibilità delle interfacce generate.

Questa scelta metodologica nasce dalla constatazione che la correttezza funzionale, per quanto necessaria, non è di per sé condizione sufficiente a garantire un prodotto realmente centrato sull'utente: un sito web che soddisfa ogni requisito tecnico e funzionale può comunque risultare inaccessibile, poco intuitivo o esteticamente inadeguato, compromettendo l'esperienza complessiva dell'utente finale. Rendere le interfacce accessibili e di semplice interazione non rappresenta quindi un aspetto accessorio, ma una componente essenziale della qualità del software: solo l'integrazione sistematica di principi di usabilità e accessibilità consente di innalzare l'esperienza utente e, di conseguenza, l'effettiva efficacia del prodotto finale.

Attraverso un ciclo di pianificazione con feedback di un LLM che genera il codebase a partire da direttive testuali ed esegue una critica strutturata del codice stesso (Self-Critique) combinato con la valutazione funzionale di un Visual-Language Model (VLM) che simula l'interazione umana, il sistema proposto è in grado di individuare i colli di bottiglia dell'interfaccia e incrementare sistematicamente la fruibilità della pagina web.

Per risolvere l'incapacità di autocorrezione strutturata degli agenti attuali, la tesi si propone di applicare in modo inedito il paradigma della Constitutional AI [13]. Invece di utilizzare una Costituzione per il mero allineamento etico, il framework dota il Large Language Model (impiegato come generatore e giudice) di 8 principi fondamentali derivati dalle linee guida dell'HCI. I contributi teorici e pratici di questo lavoro si concretizzano in due elementi fondamentali:

- **Pipeline di Refinement Iterativo**: Un'architettura robusta che orchestra in modo sinergico un LLM e un VLM per automatizzare l'intero ciclo di generazione del codice, valutazione funzionale e autocritica.

- **HCI Constitution**: Un framework normativo basato su principi di Human-Computer Interaction che impone al modello giudice di allineare le proprie risposte agli standard di accessibilità, migliorando in modo misurabile le User Interfaces e, di conseguenza, la User Experience.

1.2 Struttura della tesi

Il presente documento espone in dettaglio l'ideazione e la validazione del framework, articolandosi nei seguenti capitoli:

Capitolo 2 – Background e Lavori Correlati : Analizza lo stato dell'arte degli agenti LLM autonomi, esplorando i benchmark per la generazione web (WebGen-Bench) [14], le architetture a feedback multi-livello (WebGen-Agent) [9] e i fondamenti della Constitutional AI [13].

Capitolo 3 – Metodologie e Setup Sperimentale : Descrive la struttura operativa della pipeline, dal setup in ambiente isolato (Sandbox) fino alla strutturazione della HCI Constitution e alle logiche di promozione del codice tramite metriche oggettive.

Capitolo 4 – Analisi di Affidabilità e Validazione Metodologica : Affronta le criticità di valutazione dei modelli (es. Reward Hacking), dimostrando tramite studi di ablazione e test di preferenza in cieco il reale impatto dei principi HCI rispetto alla stocasticità di base.

Capitolo 5 – Risultati sperimentali e Valutazioni : Presenta l'efficacia del Constitutional Web Agent, documentando il miglioramento nella fruibilità dei codici generati e revisionati e identifica l'ottimo computazionale (Early Stopping) dell'approccio proposto.

Capitolo 6 – Discussione e Conclusioni : Sintetizza i traguardi raggiunti, i limiti strutturali riscontrati e le future prospettive di integrazione del framework in flussi di sviluppo operativi.

Capitolo 2

Background e Lavori Correlati

Gli attuali strumenti per la generazione di codice web consentono di velocizzare il processo di stesura di codice e consentono anche a utenti privi di competenze specifiche in software engineering di creare siti completi a partire da direttive testuali [15].

Parallelamente, la ricerca si è concentrata sull'ottimizzazione e sull'accelerazione dell'apprendimento dei Large Language Models (LLM) per la generazione di interfacce web. Per snellire il processo di addestramento e renderlo meno oneroso dal punto di vista umano, sono stati intrapresi studi volti a utilizzare l'Intelligenza Artificiale stessa per fornire feedback e revisionare gli output di altre IA generative. Questo ha segnato il passaggio da un paradigma di Reinforcement Learning from Human Feedback (RLHF) a uno di Reinforcement Learning from AI Feedback (RLAIF) [12].

Le architetture multi-agente iniziano a sfruttare attivamente questi paradigmi. Nello specifico, lo sviluppo di interfacce utente vede un impiego sempre maggiore dei Vision-Language Models (VLM) [10] per eseguire critiche visive automatizzate [11, 9]. Questi strumenti forniscono feedback testuali oggettivi e punteggi basati su scale predefinite analizzando screenshot statici e suggerendo migliorie estetiche o strutturali.

La limitazione attuale risiede però nella conoscenza pregressa e nel metodo di valutazione euristico del modello visivo utilizzato per l'analisi. Si presenta dunque la necessità di introdurre un nuovo metodo di valutazione che permetta di ottimizzare le interfacce conferendo al sistema precise linee guida da seguire, limitando al minimo l'interazione uomo-macchina [13].

2.1 Background

2.1.1 Generazione di siti web a partire da direttive in linguaggio naturale

Per quanto riguarda l'utilizzo di LLM per la generazione di codice web, è diventato molto popolare l'impiego di tool che consentono di generare codice completo partendo da direttive testuali semplici (prompt) [1]. Questo paradigma ha il duplice obiettivo di accelerare il flusso di lavoro degli sviluppatori esperti e di consentire a utenti senza basi di programmazione di affacciarsi allo sviluppo software.

Strumenti commerciali come Bolt.new e Lovable.dev [16, 15] permettono di tradurre le richieste in linguaggio naturale in un'analisi progettuale ad alto livello. Tali sistemi non si limitano alla generazione di singoli frammenti di codice, ma orchestrano autonomamente una struttura multi-file, configurando l'ambiente, gestendo le dipendenze e integrando sia i requisiti funzionali che le scelte di design iniziali.

Tuttavia, l'analisi dello stato dell'arte su questi strumenti evidenzia una chiara dicotomia tra i vantaggi offerti e i limiti architetturali nei progetti su larga scala. Per quanto riguarda i vantaggi e punti di forza spiccano:

- Prototipazione rapida: Consentono di abbattere drasticamente i tempi di setup iniziale, fornendo una struttura di base funzionante in pochi secondi.

- Orchestrazione multi-file: I sistemi più avanzati riescono a gestire la coerenza tra file separati.

- Abbattimento delle barriere tecniche: Permettono a figure non tecniche (come designer o product manager) di testare interattivamente concetti di UI senza richiedere l'intervento immediato di un team di sviluppo.

Nonostante l'elevata capacità di generazione iniziale, questi sistemi presentano criticità significative quando si passa dalla prototipazione allo sviluppo di software robusto. In particolare la cecità rispetto alle normative HCI e all'usabilità rappresenta il limite più critico. Strumenti come Bolt.new [16] tendono a ottimizzare l'estetica superficiale per compiacere l'utente, ignorando sistematicamente regole fondamentali di accessibilità o usabilità. Possono generare, ad esempio, strutture DOM semanticamente scorrette (es. l'uso di al posto di <button>), che superano i test funzionali ma falliscono l'ispezione tramite screen reader [17], strumenti che analizzano direttamente il Document Object Model (DOM), o rendono complessa l'interazione per l'utente finale.

2.1.2 Automazione della valutazione funzionale dei siti web

Con il potenziamento dell'AI generativa, si è presentata la possibilità di integrarla nella risoluzione di task complessi tramite comandi in linguaggio naturale. Un'applicazione fondamentale nello sviluppo web riguarda l'automazione dell'esecuzione dei task.

In lavori come WebArena [5] è stato creato un ambiente altamente realistico che consente di sviluppare e testare agenti autonomi. L'idea alla base utilizza comandi in linguaggio naturale per far eseguire a un modello multimodale task web-based (ad esempio, "acquista un biglietto aereo") che vengono poi valutate per verificarne il corretto completamento.

Analogamente, WebVoyager [18] costruisce un agente multimodale in grado di navigare sul web online, eseguendo azioni basate su screenshot e testi per completare le richieste dell'utente.

Come evidenziato nel framework di UXAgent [19], la creazione e l'utilizzo di ambienti dedicati permette l'automazione del testing funzionale dei siti tramite GUI-Agent , rendendo marginale l'intervento umano. Fornendo uno spazio di azioni possibili e direttive ad alto livello, è possibile far eseguire task complessi a modelli ottenendo una valutazione oggettiva delle funzionalità del codice.

L'analisi di queste architetture evidenzia chiari vantaggi. Consentono di fatto di verificare rapidamente se un'interfaccia permetta di raggiungere un obiettivo prefissato, automatizzando test che richiederebbero un intervento umano. Forniscono una valutazione oggettiva basandosi su metriche chiare (come il completamento di una transazione o il cambiamento di stato del sistema) per stabilire se il codice funziona correttamente. Sfruttano l'unione di testo ed eventulamente analisi visiva (tramite VLM) per simulare il flusso di azioni di un utente umano.

Il limite principlae di questo tipi di sistemi risiede però nel loro focus esclusivo di completamento del task (il *"cosa"*), ignorando del tutto la qualità dell'esperienza d'uso (il *"come"*). Un agente autonomo di fatto, operando tramite coordinate spaziali o iniezioni di script, è in grado di interagire correttamente con elementi della pagine web che non sono correttamente istanziati (è in grado di cliccare istantaneamente su un pulsante grande 2 pixel posizionato in un angolo dello schermo). Il task risulterebbe completato con successo, ma per un essere umano l'interfaccia risulterebbe inaccessibile. Vi è inoltre una mancanza di valutazione del carico cognitivo, gli agenti non provano confusione e interfacce disordinate vengono navigate tendenzialmente con successo, mascherando i difetti di User Experience (UX).

Si evince dunque che l'automazione funzionale proposta da strumenti come WebArena [5], WebVoyager [18] e UXAgent [19] non consenta di garantire un'emulazione dell'interazione umana veritiera. Di conseguenza, essa non è in grado di coprire quegli aspetti di usabilità e accessibilità che sono necessari per ottenere un

prodotto finale realmente corretto e fruibile.

2.1.3 RLAIF vs RLHF

Il Reinforcement Learning from Human Feedback (RLHF) [20] si è affermato come lo standard per allineare le risposte dei LLM alle intenzioni e ai valori umani [21]. Questa tecnica si basa sull'addestramento di un Reward Model (modello di ricompensa) a partire da un vasto dataset di preferenze espresse da annotatori umani. Tuttavia, l'analisi dello stato dell'arte evidenzia come il RLHF presenti criticità strutturali che ne ostacolano l'applicazione su larga scala:

- Collo di bottiglia della scalabilità: la raccolta di annotazioni umane di alta qualità richiede tempi prolungati e costi proibitivi, rendendo difficile l'aggiornamento continuo dei modelli.
- Soggettività e rumore: il giudizio umano è soggetto a bias, variazioni culturali e cali di attenzione, introducendo rumore nel dataset di addestramento.
- Complessità nei domini specializzati: per task altamente tecnici (come la valutazione algoritmica o, nel contesto di questa tesi, l'analisi delle euristiche HCI), reperire annotatori umani con le adeguate competenze risulta ancora più complesso e oneroso.

Per superare questi limiti, la ricerca recente ha introdotto il paradigma del Reinforcement Learning from AI Feedback (RLAIF) presentato in Constitutional AI [13]. Questo approccio sostituisce l'annotatore umano con un modello linguistico avanzato che agisce come giudice (LLM-as-a-Judge). L'impiego del RLAIF ha dimostrato un'efficacia sorprendente. Diversi studi hanno evidenziato come RLHF e RLAIF ottengano livelli di allineamento del tutto comparabili su task complessi, quali la generazione di riassunti e la produzione di dialoghi utili (helpfulness) e sicuri (harmlessness) [12]. Ciò dimostra la fattibilità architetturale di delegare la generazione di feedback a partire da una critica all'Intelligenza Artificiale per poi applicarli a fasi di miglioramento. L'applicazione del RLAIF consente di ottenere:

- Automazione e scalabilità: permette di generare enormi volumi di feedback in tempi ridotti e a costi marginali, accelerando i cicli di addestramento e fine-tuning.
- Coerenza valutativa: un modello istruito correttamente applica le regole di valutazione in modo deterministico e costante, riducendo le fluttuazioni tipiche del giudizio umano.

Questo cambio di paradigma trova oggi applicazione nelle architetture all'avanguardia che sfruttano tecniche di *Self-Critique* (autocritica). In tali sistemi, il modello

viene istruito a valutare e correggere iterativamente i propri output rispetto a un set di regole prestabilite. Ciò consente di automatizzare l'intero ciclo di allineamento tramite RLAIF, riducendo al minimo la necessità di supervisione umana e aprendo la strada all'impiego di questi modelli in domini complessi dove un feedback scalabile e deterministico risulta fondamentale.

2.2 Lavori Correlati

Il lavoro di tesi si è sviluppato principalmente partendo dalla struttura e dalle idee proposte nei lavori di WebGen-Bench [14], WebGen-Agent [4] e di Constitutional AI [13].

2.2.1 Benchmark per la valutazione del codice generato

Considerando il largo impiego degli agenti basati su LLM per la generazione di codice per siti web a partire da direttive in linguaggio naturale [1], si presenta la necessità di individuare un sistema in grado di valutare le capacità dei modelli di creare codice complessi. Per questo motivo, lavori come WebGen-Bench [14] puntano a introdurre dei benchmark composti da una serie di direttive di generazione per siti web e un insieme di relative azioni da far eseguire da un GUI-Agent sul sito per consentire di valutare le abilità dei vari modelli generativi di fornire codice funzionante.

2.2.2 Architetture basate su agenti per lo sviluppo frontend

L'automazione della generazione e del testing funzionale del codice di siti web ha fatto in modo di velocizzare il processo di sviluppo del software, riducendo notevolmente l'intervento umano [19]. Per consentire non solo di generare il codice a one-shot ma di permettere di applicargli delle modifiche seguendo determinati linee guida e feedback si sono svilupatte architetture che consentono di sfruttare cicli iterativi e di backtracking per migliorare il codice sorgente. In questo constesto, il framework WebGen-Agent [9] introduce una struttura capace di generare iterativamente codice per siti web migliorandone le funzionalità e il design, analizzando screenshot e sfruttando un GUI-Agent per ottenere feedback di miglioramento visivi e funzionali delle applicazioni web.

Nello specifico, l'approccio combina un meccanismo di generazione iterativa del codice con meccanismi di selezione dello stato migliore e backtracking per migliorare le caratteristiche funzionali e visive. Si avvale di un LLM per la generazione del codice, di un VLM per la valutazione visiva degli screenshot statici e di un GUI-Agent per eseguirne l'analisi funzionale.

La struttura esegue dunque un'analisi multi-livello, utilizzando le metriche di Task Success Rate (TSR - la percentuale di task correttamente eseguiti sul codice) e un punteggio fornito dall'analisi visiva del VLM delle pagine, per determinare lo stato migliore e fornire dei feedback relativi alle criticità del codice generato.

2.2.3 Constitutional AI

Lo studio della Constitutional AI [13] rappresenta un'evoluzione significativa rispetto alle metodologie tradizionali basate sul feedback umano.

In questo lavoro viene presentato un framework capace di sfruttare un insieme di regole o principi (la Costituzione) definiti dall'essere umano per allineare le risposte di un LLM, rendendole più utili e innocue. La struttura creata sfrutta l'AI feedback per l'aggiornamento dei modelli senza l'utilizzo di label umani, limitando l'intervento umano alla stesura dei principi della Costituzione. Sarà poi un modello con il compito di giudice a definire le valutazioni basandosi sui criteri stabiliti da quest'ultima. È in questo contesto che viene presentato il concetto di Reinforcement Learning from AI Feedback (RLAIF) che va a sostituire il precedente stato dell'arte basato su annotatori umani denominato Reinforcement Learning from Human Feedback (RLHF).

Capitolo 3

Metodologie e Setup Sperimentale

Questo capitolo introduce l'architettura concettuale e operativa del framework e della pipeline proposti. Il capitolo presenta inizialmente la struttura e il funzionamento della pipeline, individuando le macrofasi del flusso di lavoro e analizzandole di conseguenza. Successivamente presenta la struttura e il funzionamento delle leggi applicate al fine di coprire lo scopo del lavoro.

3.1 Introduzione e Obiettivi del lavoro

L'obiettivo primario di questo lavoro di tesi è lo sviluppo di un Constitutional Web Agent: una pipeline automatizzata basata su modelli generativi per la creazione, la valutazione e l'ottimizzazione iterativa delle interfacce utente (UI). Il sistema analizza le funzionalità del codice generato e traduce le critiche basate sulle euristiche di usabilità e i principi etici di design in feedbacks per il miglioramento del codice. Nel dettaglio il sistema si compone di quattro passaggi principali:

- Generazione del codice a partire da specifche in linguaggio naturale.
- Valutazione funzionale dell'interfaccia tramite simulazione dell'interazione umana.
- Autocrica del codice basata su principi di Human-Computer Interaction (HCI).
- Ottimizzazione tramite applicazione di un feedback strutturato al fine di migliorare le User Interfaces (UI) e la User Experience (UX).

Operativamente, l'architettura della pipeline si articola in due macro-fasi: una fase di inizializzazione (Setup) e un ciclo di Refinement iterativo.

Durante la fase di Setup, un Large Language Model (LLM) genera il codice sorgente iniziale, configurando un ambiente web funzionante. Successivamente, un Visual-Language Model (VLM) interagisce con l'interfaccia renderizzata all'interno di un browser isolato, portando a termine specifici task per fornire una baseline di valutazione funzionale, ottenendo una metrica di successo dei task (TSR - Task Success Rate).

A questo punto, l'LLM assume un ruolo di Self-Judge, analizzando il codice sorgente e formulando una critica oggettiva basata su principi e regole predefiniti, stilati dall'analisi delle euristiche e degli standard di usabilità e di design.

Nella successiva fase di Refinement iterativo, l'LLM aggiorna il codice implementando i feedbacks generati dalla fase di autocritica. Ogni nuova versione del sito viene quindi sottoposta a un nuovo ciclo di interazione da parte del VLM e a una successiva critica, innestando un processo di miglioramento continuo della UI.

Per le fasi di valutazione e sperimentazione, il ruolo di modello generatore del codice e di Self-Judge (LLM) è stato affidato a Qwen3.5-27B [22], un modello denso da 27 miliardi di parametri con modalità di ragionamento (thinking) e una finestra di contesto nativa di 262.144 token. La scelta è motivata dalla necessità di gestire artefatti software di dimensioni considerevoli e di mantenere in memoria un contesto esteso durante le fasi di critica e revisione.

Per l'interazione visiva e funzionale con le interfacce generate è stato invece impiegato Holo2-8B [23], un modello vision-language specializzato in computer-use e nel grounding di elementi UI, sviluppato a partire da Qwen3-VL. Il modello è stato selezionato perché, pur con un numero contenuto di parametri, ottiene risultati competitivi nei benchmark di localizzazione degli elementi, rendendolo adatto a individuare con precisione gli elementi delle pagine web.

È essenziale sottolineare che lo scopo del processo non è la correzione di difetti logici (bug fixing) o il miglioramento delle funzionalità del backend, bensì l'ottimizzazione esclusiva a livello di usabilità e accessibilità. I miglioramenti apportati dal sistema mirano a perfezionare il design, l'accessibilità e l'usabilità delle interfacce, rendendo le azioni utente più intuitive. Di conseguenza, il framework non altera direttamente la logica di business o le funzioni sottostanti del codice originario.

L'incremento del TSR atteso dall'applicazione dell'architettura proposta, dunque, non deriva dalla correzione di algoritmi non funzionanti, ma da un potenziamento mirato della UI che riduce la possibilità di incorrere in errori di interazione.

3.2 Motivazioni e contesto teorico

Negli ultimi anni, i sistemi basati su LLM hanno dimostrato capacità notevoli nella generazione di codice a livello di repository e a partire da direttive in linguaggio

naturale [1, 4]. tuttavia l'estensione di queste capacità alla generazione di interfacce web presenta delle sfide.

Attualmente, la maggior parte delle architetture agentiche per il testing del codice generato si affida esclusivamente a feedback derivanti dall'esecuzione di quest'ultimo per la verifica del proprio operato [5, 6]. Questo approccio non riesce a catturare l'effettiva qualità del codice generato, che dipende anche fortenemente dagli effetti visivi e dal feedback derivato dall'esperienza dell'utente. Di conseguenza, i modelli tendono a produrre interfacce che possono essere affette da problemi di rendering e di usabilità, come componenti disallineati, combinazioni di colori disarmoniche, pulsanti non responsivi o collegamenti interrotti.

Per colmare questo divario, è emersa dunque la necessità di integrare meccanismi di valutazione più sofisticati nel ciclo di sviluppo automatizzato. Approcci come i framework di ArtifactsBench [8] e WebGen-Agent [9], propongono un sistema che sfrutta feedback multi-livello (visivi e funzionali) per generare e raffinare iterativamente il codice. Questo approccio consente di ottenere informazioni relative a criticità che spaziano ambiti diversi e generare suggerimenti mirati al miglioramento dei siti.

Tuttavia, per garantire una migliore generazione delle risposte degli LLM (di codice frontend nel caso di studio) risulta necessario fornire agli agenti un framework strutturato per consentire di effettuare delle analisi più mirate, in modo da elaborare soluzioni più in linea con le esigenze umane, come proposto in Constitutional AI [13]. A questo scopo, viene introdotto un framework di critica costituzionale per estendere il processo di raffinamento applicando una struttura simile a quella di Constitutional AI.

Originariamente questo paradigma è stato sviluppato per addestrare assistenti AI attraverso un processo di automiglioramento, eliminando la necessità di etichette umane e fornendo supervisione esclusivamente tramite una lista di regole o principi predefiniti che compongono una costituzione. Il processo si articola in una fase di apprendimento supervisionato in cui il modello genera una risposta, produce autonomamente un'autocritica e formula una revisione basandosi sui principi imposti.

Questo metodo permette non solo di controllare il comportamento del modello in modo molto più preciso, ma sfrutta anche il ragionamento in stile chain-of-thought per aumentare la trasparenza e le prestazioni dei processi decisionali dell'AI.

L'intuizione fondamentale alla base di questa tesi è dunque l'applicazione del paradigma costituzionale, tipicamente usato per mitigare comportamenti dannosi, alla validazione architetturale e visiva delle interfacce utente. Invece di affidarsi ad un'analisi visiva passiva e alla sola funzionalità del codice, il nuovo framework dota l'agente di una costituzione focalizzata su euristiche di design, regole di accessibilità e corretta implementazione strutturale. Sfruttando la capacità di generare critiche e revisioni il modello agisce come un giudice formale: analizza le discrepanze tra la

UI generata e i requisiti, elaborando un refinement iterativo altamente trasparente e controllato.

3.3 Dataset utilizzati

Per la valutazione sperimentale e funzionale della pipeline e dell'applicazione del framework, il presente lavoro si avvale dei dataset introdotti nella letteratura e specificamente progettati per valutare la generazione di applicazioni web partendo da zero. In particolare, i dati utilizzati derivano dal lavoro di WebGen-Bench [14] e sono stati suddivisi per coprire in modo esaustivo sia la fase di testing rigoroso sia le valutazioni funzionali più ampie. È stato utilizzato WebGen-Bench per la fase di testing, mentre per la fase di affidabilità e validazione metodologica dell'architettura proposta è stato utilizzato WebGen-Instruct.

WebGen-Bench rappresenta il primo benchmark progettato per misurare la capacità di un agente basato su LLM di creare da zero una codebase multi-file per siti web. Il benchmark contiene 101 istruzioni di generazione diversificate. Queste istruzioni coprono tredici tipologie di applicazioni web e sono state raggruppate in tre grandi categorie: Content Presentation, User Interaction e Data Management. Per valutare accuratamente i siti web generati, il dataset fornisce 647 casi di test (task) progettati per validare sia i requisiti funzionali che quelli di apparenza visiva. Ogni task specifica un'operazione da eseguire sul sito web e il risultato atteso in seguito a tale operazione. Le istruzioni sono state create attraverso un processo sistematico di raccolta dati che ha combinato l'esperienza di annotatori umani con l'uso di GPT-4o per generare descrizioni progettuali complesse e i test da eseguire sul sito web e i relativi risultati attesi.

Per la fase di affidabilità e validazione metodologica di autocritica del sistema, sono stati estratti e utilizzati vari sottocampioni del dataset WebGen-Instruct. WebGen-Instruct è un più ampio dataset composto da 6.667 istruzioni dettagliate per la generazione di siti web. Per questo set di istruzioni non sono forniti i test da effettuare per controllare le funzionalità e gli effetti visivi del sito. Per questa ragione, in questo lavoro di tesi, è stato sfruttato il modello Gemini Pro 3.1 [24] per la generazione dei task e dei relativi risultati attesi, fornendo come base di partenza la struttura di WebGen-Bench e richiedendo di generare un insieme di test che coprissero in modo esaustivo le funzionalità e le caratteristiche richieste dalle istruzioni.

3.4 Architettura della Pipeline proposta

Di seguito vengono analizzate le varie fasi che compongono il processo di Setup e la fase iterativa della pipeline presa in considerazione. Una struttura schematizza

delle due fasi viene mostrata nelle figure 3.1 e 3.2.

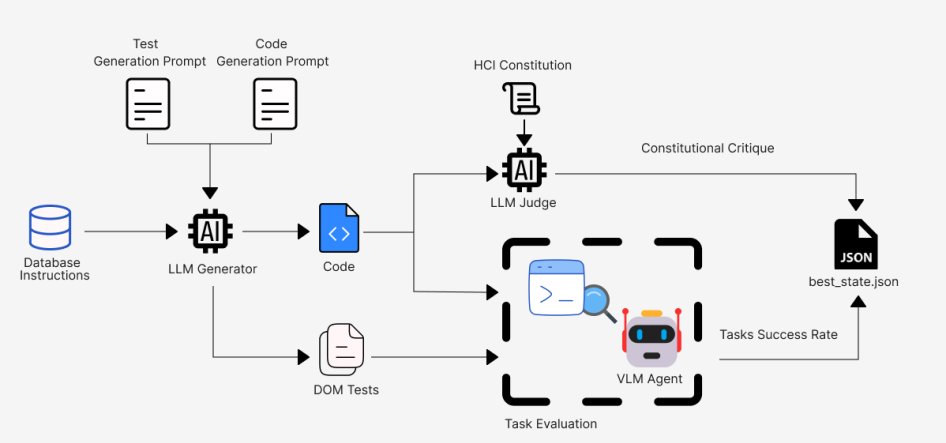


Figura 3.1: Struttura schematizzata della fase di Setup

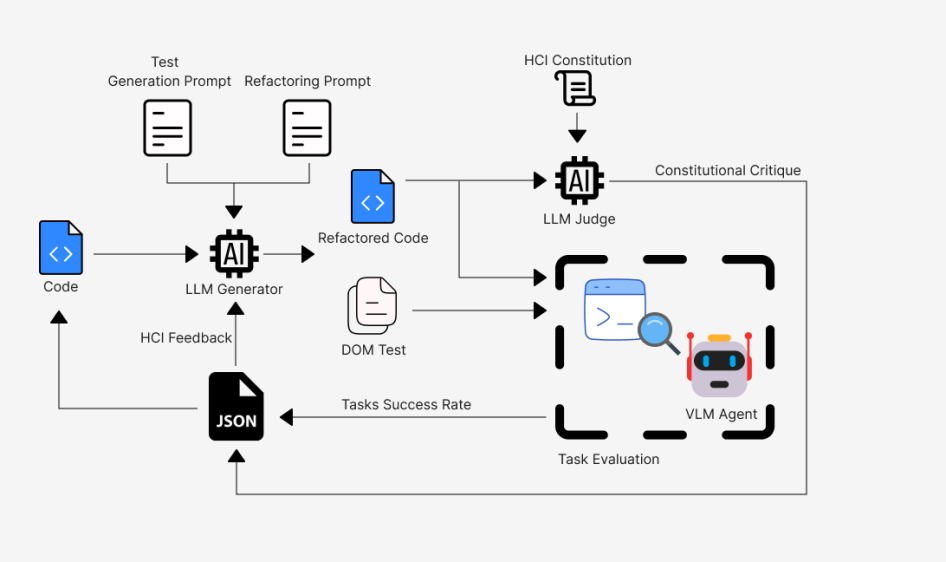


Figura 3.2: Struttura schematizzata della fase di Refinement

3.4.1 Fase generativa del codice

La prima fase della pipeline operativa consiste nella sintesi del codice sorgente a partire da una specifica testuale ad alto livello. Ad un Large Language Model viene fornito un system prompt che definisce il suo ruolo come sviluppatore

frontend esperto di UI/UX e principi di Human-Computer Interaction (ispirato all'impostazione di WebGen-Agent [9] e a sistemi agentici come Bolt.dev [16]).

Ricevuta in input la descrizione dei requisiti funzionali ed estetici, il modello ha il compito di generare il codice completo utilizzando uno stack tecnologico basato su HTML, JavaScript e Tailwind CSS. Un aspetto cruciale di questa fase di generazione è l'implementazione del mocking dei dati e l'esecuzione in un ambiente totalmente isolato (SandBoxed). Questa scelta progettuale è motivata da due fattori fondamentali, uno metodologico e uno infrastrutturale. Prima di tutto si ha un isolamento delle metriche di valutazione: poiché l'obiettivo della ricerca è valutare e ottimizzare esclusivamente l'interfaccia utente, svincolare il frontend da un reale backend permette di azzerare le latenze di rete e di escludere a priori potenziali errori server-side (es. timeout o fallimenti di chiamate API). In questo modo, si ha la garanzia accademica che eventuali fallimenti riscontrati dal Visual-Language Model durante la Task Evaluation (TE) siano imputabili prevalentemente a difetti strutturali o visivi della UI/UX, e non a problematiche infrastrutturali esterne.

In secondo luogo si presentano i vincoli di esecuzione offline: l'ambiente di calcolo target, basato sui cluster HPC del Cineca e del Politecnico di Torino, impone severi limiti di sicurezza, tra cui l'impossibilità di accedere alla rete Internet pubblica o a database esterni durante l'esecuzione dei job. Di conseguenza, l'LLM è istruito per incorporare direttamente all'interno della logica JavaScript le strutture dati necessarie per simulare lo stato dell'applicazione (mocking) e per assumere che tutte le dipendenze esterne necessarie al corretto rendering (come le librerie CSS) siano già precaricate nell'ambiente locale.

A livello implementativo, per gestire la natura multi-file di un'applicazione web, l'output generato dall'LLM viene formattato e consolidato all'interno di un unico artefatto testuale (artifact.txt). Questo documento segue la struttura proposta da Bolt.diy [16] e ripresa da WebGen-Agent [9] tramite tag delimitatori specifici (ad esempio <BoltArtifact> e <BoltAction>), i quali incapsulano e separano logicamente i singoli file sorgente (come, ad esempio, index.html, app.js o ui.js).

Al termine della generazione, uno script di parsing analizza l'artefatto, estrae i singoli blocchi di codice e ricostruisce in modo automatizzato la struttura di directory necessaria. Il sito web risulta così istanziato e pronto per essere renderizzato localmente in vista della successiva fase di simulazione e valutazione.

3.4.2 Generazione dei test di controllo dello stato

Al termine della fase di generazione del codice, quest'ultimo viene utilizzato per consentire la corretta valutazione funzionale della fase successiva. In particolare, il codice per essere testato e valutato, sarà ospitato ed eseguito all'interno di un'istanza headless del browser Chromium [25], orchestrata mediante la libreria Playwright [26].

Per valutare la correttezza del completamento dei task al seguito delle azioni intraprese dal Visual-Language Model durante la fase di Task Evaluation (TE), risulta necessario eseguire un controllo del Document Object Model (DOM). Per fare ciò si sfrutta la capacità dell'LLM di generare dei test Playwright capaci di confrontare lo stato del DOM, analizzando gli elementi presenti nella pagine e le relative caratteristiche.

Per ottenere dei test logicamente e sintatticamnete corretti, si richiede al modello generativo di considerare sia il codice sia le coppie composte dai task e relativi risultati attesi definite all'interno del dataset. In questo modo per ogni task da effettuare su un determinato sito sarà generato un test che controllerà lo stato del DOM per verificare che le condizioni di completamento siano soddisfatte.

3.4.3 Valutazione funzionale tramite Visual-Language Model

Terminata la fase di generazione del codice e dei test Playwright, la pipeline avvia la fase di valutazione funzionale. Il codice generato viene dunque eseguito tramite un'istanza browser Chromium [25], mediante la libreria Playwright [26].

In questo ambiente isolato entra in gioco il Visual-Language Model (VLM), il quale assume il ruolo di utente umano simulato. Il compito del VLM è interagire con l'interfaccia web per portare a termine una serie di task predefiniti, presenti nel dataset. Per ogni task, il dataset fornisce un obiettivo e il relativo risultato atteso. Il processo decisionale del VLM si basa su un ciclo di Observation-Action: il modello riceve in input tramite prompting uno screenshot dello stato corrente della pagina web, la descrizione testuale del task e lo storico delle azioni già intraprese. Analizzando visivamente il layout e le affordance dell'interfaccia, il VLM deve generare in formato JSON la sequenza di istruzioni Playwright necessarie per avanzare verso l'obiettivo. Le capacità di interazione del VLM sono mappate su uno specifico Spazio delle Azioni che comprende le seguenti direttive:

- 'mouse_click': Simula il click del mouse su un elemento individuato tramite coordinate spaziali (bounding box [ymin, xmin, ymax, xmax] normalizzate 0-1000). Permette di specificare il tasto ("left"/"right") e il numero di click.
- 'keyboard_type': Inserisci una stringa di testo all'interno di un campo di input correntemente in focus.
- 'keyboard_press': Simula la pressione di tasti speciali di controllo (es. "Enter", "Tab") o le frecce direzionali ("ArrowUp", "ArrowDown", ecc.).
- 'scroll': Esegue lo scorrimento della pagina specificando il delta vettoriale (delta_x, delta_y).

- 'clear_input': Elimina il contenuto testuale presente in un campo di input identificato dalla relativa bounding box, definito dalla combinazione di 'mouse_click' e 'keyboard_press'.

Per ottimizzare l'esecuzione, al VLM è consentito il batching delle azioni: qualora la topologia visiva lo permetta (es. una schermata di login interamente visibile), il modello può concatenare in un'unica risposta multipli passaggi sequenziali (ad esempio, click sul campo username, inserimento del testo, click sul campo password, inserimento testo e click sul pulsante di submit).

Ogni azione generata viene registrata in un file di history ed eseguita sequenzialmente nel browser. Al termine dell'esecuzione, la pipeline verifica il successo dell'operazione valutando lo stato del Document Object Model (DOM). Questa verifica avviene tramite un test Playwright, precedentemente generato dall'LLM, che funge da oracolo: se lo stato finale del DOM coincide con le asserzioni del test, il task è considerato completato con successo. In caso contrario, il sistema richiede al VLM una nuova azione basata sullo stato aggiornato, fino al raggiungimento dell'obiettivo del task o all'esaurimento delle iterazioni di pianificazione consentite (10 iterazioni di pianificazione per task).

Al termine dell'esecuzione dell'intero set di task associati ad un sito, il sistema quantifica le prestazioni funzionali calcolando il Task Success Rate (TSR). Il TSR è definito come il rapporto tra il numero di task completati con successo ($T_{success}$) e il numero totale dei task eseguiti (T_{total}) per quel determinato sito:

$$TSR = \frac{T_{success}}{T_{total}}$$

Questa metrica viene registrata all'interno di un sistema di tracciamento dello stato (un file denominato best_state.json), che archivia la versione corrente del codice (il path del codice all'interno della directory del progetto) e il relativo TSR.

3.4.4 Autocritica basata su principi di HCI

La fase di critica e di conseguente raffinamento rappresenta il nucleo innovativo di questo lavoro di tesi. A differenza degli approcci presenti in letteratura – dove l'aggiornamento dell'interfaccia si basa prettamente su un'analisi visiva statica o su feedback reattivi legati alla fallita esecuzione di un task (come osservato in WebGen-Agent [9]) – la pipeline proposta adotta un paradigma di ottimizzazione basata su regole.

L'obiettivo non è la generica risoluzione di bug funzionali, ma l'innalzamento sistematico e misurabile delle User Interfaces (UI) e della User Experience (UX).

Per implementare questo meccanismo, il progetto adatta e reinterpreta il concetto di Constitutional AI [13]. Mentre nelle implementazioni originarie questo paradigma

viene impiegato per allineare le risposte dei Large Language Models a criteri di sicurezza ed etica (helpful and harmless), in questa sede la Costituzione è ridefinita come un set rigoroso di 8 euristiche architetturali ed estetiche, definiti principi.

Ogni principio è strutturato formalmente tramite un identificativo (ID), un nome, una descrizione, uno specifico prompt di autocritica e un'istruzione di raffinamento generale.

Come illustrato in Figura 3.2, il processo di autocritica (Self-Critique) si colloca all'interno della fase iterativa della pipeline, la quale è configurata per un limite massimo di 10 iterazioni. Questo tetto operativo è stato inizialmente definito empiricamente per garantire che il codice venga analizzato e aggiornato almeno una volta per ciascun principio, assorbendo al contempo eventuali fluttuazioni statistiche degli score assegnati.

Ad ogni ciclo di valutazione, il modello generativo riceve in input l'artefatto del codice dello stato ottimale estratto dal path presente nel file 'best_state.json'. L'LLM esegue quindi un'analisi indipendente per ciascuno degli 8 principi costituzionali. Per ogni iterazione di principio, il modello valuta il codice sorgente, formula un feedback analitico e assegna uno score quantitativo discreto da 0 a 5, indicando il grado di conformità del codice alla regola presentata nel principio.

Al termine di questa analisi, il sistema aggrega i risultati calcolando lo Score Costituzionale Totale. Contestualmente, la pipeline individua il bottleneck dell'interfaccia, identificato matematicamente come il principio che ha registrato lo score più basso e salva le metriche ottenute all'interno del file di stato 'best_state.json'.

3.4.5 Ciclo Iterativo di Refinement e Logiche di Promozione

Conclusa la fase di Setup, l'architettura operativa entra nella sua fase iterativa, il cui obiettivo è l'ottimizzazione progressiva e misurabile del codice.

Il ciclo di raffinamento è orchestrato per eseguire un numero prefissato di iterazioni. Ad ogni ciclo, la pipeline esegue sequenzialmente quattro macro-operazioni: la rigenerazione del codice, la sintesi di nuovi test Playwright per la validazione del DOM, la valutazione funzionale visiva tramite VLM (con ricalcolo del TSR) e, infine una nuova sessione di critica costituzionale.

Il punto di partenza di ogni iterazione è lo "stato dell'arte" del sistema, le cui metriche sono mantenute all'interno del registro di tracciamento, il file 'best_state.json'. La pipeline estrae da questo registro il path associato al codice sorgente migliore e lo fornisce in input al Large Language Model. Affinché il modello possa operare una rigenerazione mirata, riceve in input (tramite un prompt strutturato) sia il codice sorgente attuale, sia il feedback correttivo specifico generato durante la precedente fase di autocritica (relativo al principio costituzionale identificato come bottleneck), affiancato dall'istruzione di miglioramento generale di quest'utlimo.

Il codice così raffinato viene istanziato all'interno di un ambiente di staging (una directory temporanea isolata). Questa separazione logica a livello di file system è essenziale per garantire la sicurezza del processo: permette agli script di valutazione funzionale di eseguire il nuovo codice e calcolare il TSR aggiornato senza sovrascrivere l'artefatto stabile. Al termine della valutazione, il sistema applica una rigorosa politica di promozione basata sui risultati funzionali:

- **Rollback e Archiviazione**

  Nel caso in cui il raffinamento comprometta parzialmente l'usabilità o la chiarezza dell'interfaccia, portando a un $TSR_{new} < TSR_{best}$, il nuovo codice viene scartato. Le informazioni relative a questo tentativo fallito vengono comunque registrate nello storico delle iterazioni ("failed_attempts" presente all'interno della struttura del file 'best_state.json') a fini analitici, mentre le metriche dello stato migliore rimangono inalterate.

- **Promozione**

  Qualora le modifiche apportate dall'LLM si traducano in un miglioramento (o in un mantenimento) dell'efficacia funzionale ($TSR_{new} \geq TSR_{best}$), il codice presente nell'ambiente di staging viene validato e trasferito in una nuova directory. I metadati del precedente stato migliore vengono archiviati nello storico dei 'failed_attempts' e il sistema aggiorna il 'best_state.json' per eleggere il nuovo artefatto come stato di riferimento per l'iterazione successiva.

Il processo definisce una condizione di uscita rigida: il ciclo si arresta al completamento esatto di 10 iterazioni (sebbene la logica architetturale supporterebbe un early stopping al raggiungimento del punteggio costituzionale massimo di 40, pari a 5 su tutti i principi). La scelta sperimentale di fissare il limite a 10 iterazioni è stata identificata come condizione sufficiente per garantire un'esplorazione esaustiva: copre ampiamente l'eventualità peggiore (uno score di partenza pari a 0 su tutti e 8 i principi), assicurando almeno un ciclo di aggiornamento mirato per ogni euristica, e offre un margine statistico sufficiente per assorbire eventuali fluttuazioni intrinseche nella fase di valutazione dell'LLM.

3.5 Implementazione Tecnica e Gestione dei Dati

Questa sezione descrive l'infrastruttura tecnica alla base della pipeline, dettagliando l'organizzazione del file system e i meccanismi di mantenimento dello stato necessari a garantire l'esecuzione deterministica e tracciabile della pipeline.

3.5.1 Struttura del File System e Salvataggio degli Snapshot

Tutti i dati generati e manipolati durante l'applicazione del framework sono centralizzati all'interno di una directory principale "Outputs". Durante la fase di Setup, partendo dalle direttive di generazione fornite dal dataset, il modello produce il documento denominato 'artifact.txt'. Questo file viene archiviato in una struttura gerarchica, indicizzata tramite l'identificativo univoco (ID) assegnato a ciascun sito web.

Una funzione di parsing dedicata elabora il file 'artifact.txt' estrapolando i singoli file necessari per il rendering e la logica del sito (ad esempio index.html, app.js, data.js, ui.js), salvandoli nella medesima directory indicizzata.

Analogamente, vengono inseriti in questa locazione anche i test Playwright generati per il controllo della Task Evaluation. Questi sono estrapolati da un'ulteriore funzione di parsing a partire da un altro file denominato 'test_artifact.txt', generato dal LLM basandosi sul file contenente il codice del sito e le direttive di completamento dei task fornite nel dataset. Durante la fase iterativa, al fine di isolare il codice della baseline dalla versione ottimizzata, i file derivanti dalla fase di Refinement vengono inizialmente salvati in una directory temporanea. L'aggiornamento della struttura indicizzata definitiva avviene esclusivamente a seguito di una valutazione del Task Success Rate (TSR) che ne determina la promozione a stato migliore. Al termine del flusso iterativo, il sistema garantisce la persistenza di due alberi di directory simmetrici (baseline e reviewed), ciascuno comprensivo di 'artifact.txt', sorgenti estratte, 'test_artifact.txt', test Playwright, copie in locale delle librerie (es. Tailwind) per il funzionamento offline, e storici dettagliati di esecuzione.

3.5.2 Tracking dello Stato Iterativo

Il controllo del flusso di generazione, valutazione e promozione è demandato a un file di registro denominato 'best_state.json', allocato all'interno della struttura della baseline. Questo file mantiene i metadati dello stato corrente migliore e tiene traccia dello storico dei tentativi scartati, includendo:

- "artifact_path": Il percorso di sistema che punta all'artefatto della versione attualmente considerata. Al termine della fase di Setup, questo punta alla posizione del codice della baseline all'interno del progetto; a seguito di iterazioni migliorative, si aggiorna dinamicamente puntando alla versione revisionata.

- "best_tsr": Il valore percentuale del Task Success Rate calcolato per la versione corrente del codice migliore.

- "min_score": Il punteggio per-principio minimo rilevato a valle del processo di autocritica basato sulla Costituzione.
- "feedback": Le direttive testuali strutturate che LLM dovrà sfruttare per aggiornare il codice nel ciclo di Refinement successivo.
- "id_principle": L'identificativo del principio HCI maggiormente violato (corrispondente al "min_score"), che costituisce il bersaglio principale dell'aggiornamento.
- "failed_attempts": Raccoglie i metadati delle iterazioni subottimali. Contiene gli stati che non hanno registrato un incremento del TSR o quelli precedentemente validi ma superati da promozioni successive. Per le iterazioni scartate preliminarmente, i campi relativi ai punteggi costituzionali assumono valori di default, indicando il bypass della fase di critica (early-exit) al fine di ottimizzare i tempi di calcolo.

All'interno della directory associata al singolo sito web vengono inoltre archiviati, in file specifici denominati "task_X_history.json", i log delle azioni eseguite dal VLM e i relativi esiti, garantendo la tracciabilità necessaria per il calcolo del Task Success Rate (TSR) al termine della fase di Task Evaluation. Per ogni task, il VLM ha a disposizione un budget massimo di 10 iterazioni per interagire con il sito. Al termine di queste iterazioni, o in caso di validazione anticipata da parte del test Playwright relativo al task, viene registrato il flag booleano "dom_test_success" (true o false). Questa architettura dei log permette, in fase di valutazione finale, di calcolare in modo deterministico il TSR globale, registrato poi in 'best_state.json', semplicemente rapportando la somma dei task con esito positivo al numero totale di task assegnati.

3.6 HCI Constitution

Il concetto di Costituzione all'interno di questo framework trae diretta ispirazione dal filone di ricerca della Constitutional AI [13]. Originariamente sviluppato per addestrare modelli di intelligenza artificiale affinché siano utili e innocui (helpful and harmless) riducendo la dipendenza dal feedback umano diretto, in questo lavoro il paradigma viene traslato in un nuovo dominio applicativo: l'ingegneria e l'ottimizzazione delle interfacce utente. Invece di valutare l'allineamento etico di un testo generato, la pipeline utilizza la Costituzione come un set di regole architetturali ad alto livello per far sì che il modello esegua un'autocritica e raffini iterativamente il codice sorgente. La Costituzione qui proposta si configura quindi come un vero e proprio framework normativo, progettato per rendere il codice più fruibile e massimizzare sistematicamente la User Experience.

La sua definizione ha richiesto un processo di selezione e sintesi a partire dai principali standard teorici ed empirici della Human-Computer Interaction. L'obiettivo primario non era replicare la totalità delle normative esistenti, bensì riassumere un insieme di direttive che fossero applicabili a livello operativo, ovvero direttamente interpretabili da un modello generativo e verificabili in modo deterministico tramite l'ispezione automatizzata (ad esempio, analizzando la struttura del DOM).

A tal fine, la selezione si è concentrata su tre pilastri complementari dello sviluppo web:

- **Le 10 Euristiche di Nielsen**: fondamento per l'usabilità qualitativa e la riduzione del carico cognitivo. Forniscono regole per garantire un design coerente, feedback chiari e prevenzione degli errori [27].
- **Web Content Accessibility Guidelines (WCAG)**: integrate per estrarre requisiti strutturali e percettivi. Queste garantiscono che l'interfaccia sia inclusiva a livello semantico e accessibile anche mediante tecnologie assistive (come gli screen reader) [28].
- **La Legge di Fitts**: applicata in quanto standard per valutare l'ergonomia del movimento. Essa definisce dei vincoli per il dimensionamento e il posizionamento degli elementi interattivi al fine di minimizzare il tempo di acquisizione del target e ridurre la frustrazione dell'utente [29].

A partire da questi tre insiemi di norme è stato condotto un processo di sintesi e filtraggio. Sono state mantenute alcune delle direttive direttamente analizzabili da agenti autonomi e che non fossero troppo dipendenti dai diversi contesti applicativi, permettendo di definire dei prompt di critica e di refinement generale che consentano di applicare delle modifiche mirate al codice sorgente.

Il risultato di tale sintesi ha portato alla formulazione di 8 principi costituzionali fondamentali. Ciascuno di essi è stato concepito come una direttiva esplicita e autosufficiente, appositamente strutturata per guidare i cicli di autocritica (Self-Critique) del sistema e permettere una misurazione dei difetti di interfaccia. Di seguito viene riportata la struttura completa dei principi della HCI Constitution.

HCI_1_COGNITIVE_SYSTEM_STATUS

Principle: *"Visibility of System Status (Nielsen #1)"*

Description: *"Users experience anxiety if they don't know if their action was registered. The UI must provide immediate, unmistakable visual feedback during any processing."*

Critique Prompt: *"Analyze the HTML/JS. During simulated delays ('setTimeout'), does the UI visually confirm the processing state? Is a loading spinner rendered, and is the trigger button visually dimmed (e.g., using Tailwind 'opacity-50') to prevent double-clicks?"*

Instruction: *"Provide immediate visual feedback for user actions. During JS 'setTimeout', inject a visible loading spinner into the DOM. Visually disable the active button using Tailwind classes like 'opacity-50' and 'cursor-not-allowed' so the user clearly sees it is processing."*

HCI_2_EXPLICIT_ERROR_TEXT

Principle: *"Error Recognition and Recovery (Nielsen #9)."*

Description: *"Color-blind users or those with cognitive disabilities cannot rely solely on red borders to understand form errors. Errors must be explicit text."*

Critique Prompt: *"If form validation fails, does the UI only change border colors, or does it inject a clear, readable text paragraph explaining the error directly into the DOM?"*

Instruction: *"Implement explicit textual error feedback. On validation failure, write JS to inject a highly visible text paragraph (e.g., Email is required) directly under the input field. Do not rely solely on color changes."*

HCI_3_MOTOR_ACCESSIBILITY_FITTS

Principle: *"Fitts's Law (Pointer Targeting)."*

Description: *"Users with motor tremors (and Agents calculating pixel coordinates) need adequately sized and well-spaced clickable areas to avoid accidental misclicks with the mouse."*

Critique Prompt: *"Are interactive elements too small for comfortable pointer interaction? Are buttons missing generous Tailwind padding (like px-4 py-2) or packed too closely together without margins (gap)?"*

Instruction: *"Maximize hit targets and spacing. Apply generous Tailwind padding (e.g., 'px-4 py-2' or 'min-h-[40px]') to all buttons. Add clear spatial separation between adjacent interactive elements using margins or flex/grid gaps (e.g., 'gap-4') to prevent accidental clicks."*

HCI_4_ESCAPE_HATCHES

Principle: *"User Control and Freedom (Nielsen #3)."*

Description: *"Users often click by mistake and feel trapped in modals or popups. They need a universally recognized, highly visible 'Emergency Exit'."*

Critique Prompt: *"Do modals, popups, or dynamic overlays have a highly visible, explicitly rendered 'Close' or 'Cancel' button on the screen?"*

Instruction: *"Provide obvious 'Escape Hatches'. Ensure every modal or overlay renders a clear, standalone 'Close' button (either a distinct 'X' icon or a 'Cancel' button). Ensure the Vanilla JS completely removes or hides the element when clicked."*

HCI_5_VISUAL_AFFORDANCE

Principle: *"Visual Boundaries and Affordance."*

Description: *"Users must instantly intuitively know what is clickable and what is static text without having to guess or scrub the screen with their mouse."*

Critique Prompt: *"Do buttons look like plain text? Are they missing background colors, clear borders, or visual boundaries that distinguish them from surrounding static content?"*

Instruction: *"Enhance visual affordance so interactive elements are instantly recognizable. Ensure every button has a distinct boundary using Tailwind classes (e.g., background colors like 'bg-blue-600 text-white', or distinct borders like 'border-2 border-gray-300 rounded-md')."*

HCI_6_UNIVERSAL_COMPREHENSION

Principle: *"Clarity over Aesthetic (Text > Icons)."*

Description: *"Icons can be ambiguous across different cultures and age groups. Text labels guarantee universal comprehension."*

Critique Prompt: *"Are there important interactive elements that use ONLY an SVG/icon without any visible explanatory text next to them?"*

Instruction: *"Prioritize clarity over minimalism. For any icon-only button that performs a primary action, append a visible text label next to the SVG (e.g., place the icon and the word 'Search' together inside the button layout)."*

HCI_7_CONTEXT_ISOLATION

Principle: *"Visual Focus and Modality."*

Description: *"When a modal opens, the user's cognitive focus must be forced onto the active window, visually signaling that the background is temporarily disabled."*

Critique Prompt: *"When a modal or popup is active, is the background content still fully bright and visible, causing potential visual distraction?"*

Instruction: *"Visually isolate active modals. Add a backdrop overlay to all modals using Tailwind (e.g., a fixed div with 'bg-black/50' or 'backdrop-blur-sm') to dim the background. This prevents distraction and clearly signals that the background is inactive."*

HCI_8_CUSTOM_DIALOGS

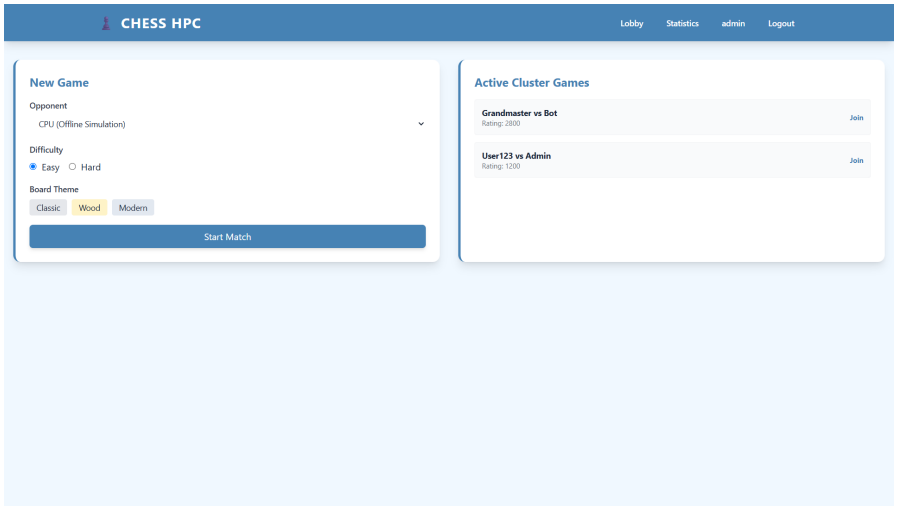
Principle: *"Consistent Brand Experience (No Native Prompts)."*

Description: *"Native browser alerts (alert, confirm) break the visual immersion of the application and cannot be styled, confusing the user."*

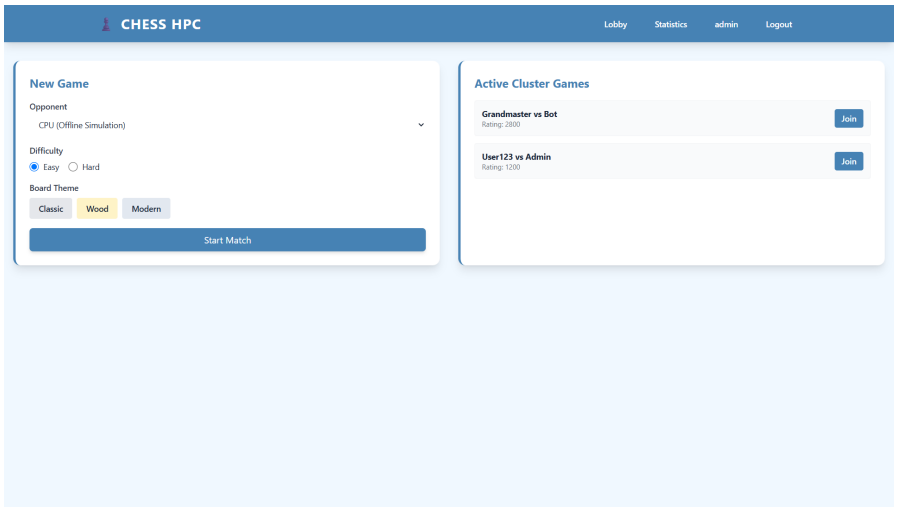
Critique Prompt: *"Does the JavaScript use native browser dialogs like 'alert()', 'confirm()', or 'prompt()' instead of in-app UI?"*

Instruction: *"Never use 'alert()', 'confirm()', or 'prompt()'. Replace all native browser dialogs with custom, styled HTML/Tailwind modals to maintain a consistent user experience and visual design."*

Le Figure 3.3, 3.4 e 3.5 mostrano il risultato dell'applicazione dei feedback di alcuni principi alle UI dei siti web.

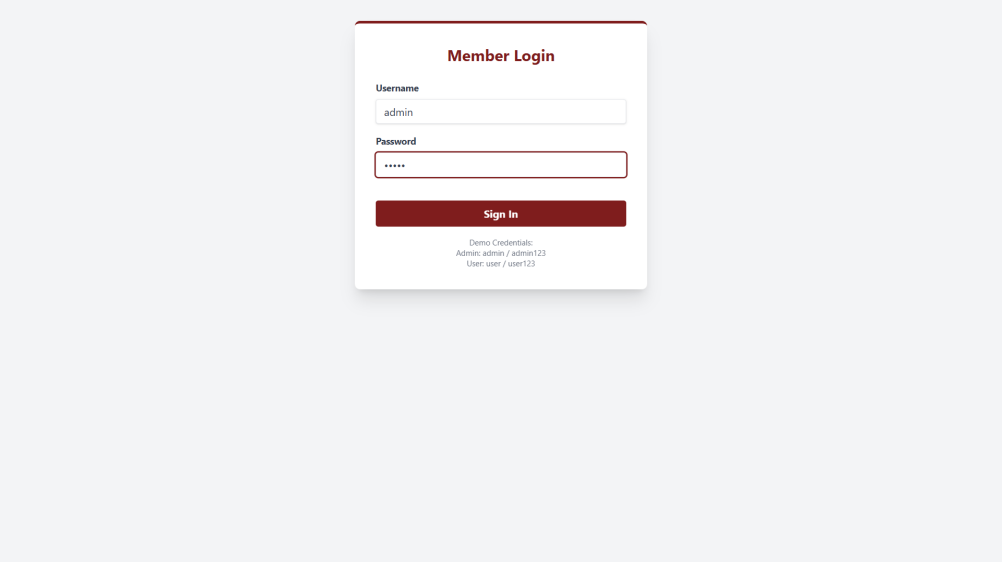


(a) UI prima dell'applicazione.

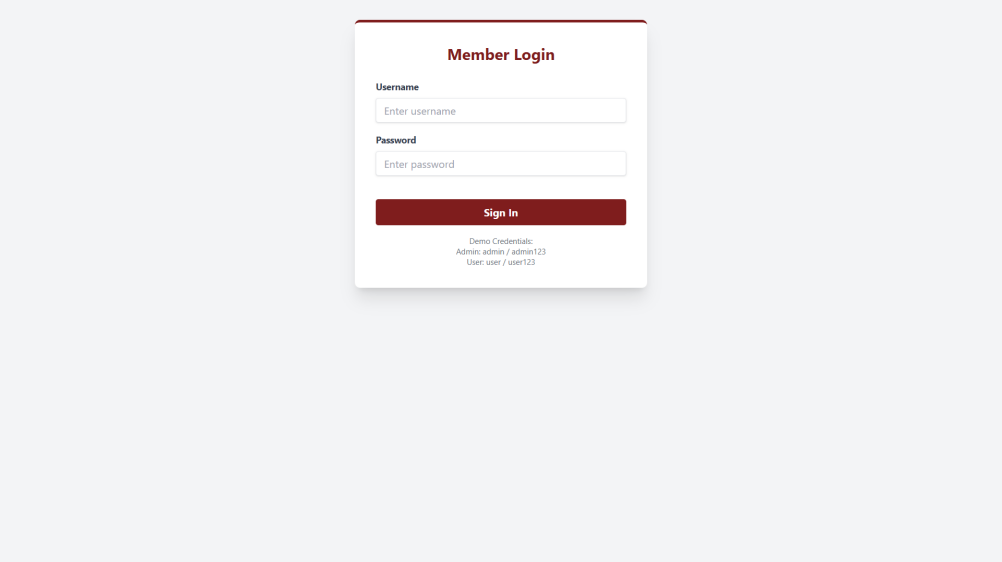


(b) UI dopo l'applicazione.

Figura 3.3: UI prima e dopo l'applicazione dei principi 3 (Fitt's Law) e 5 (Visual Boundaries and Affordance).

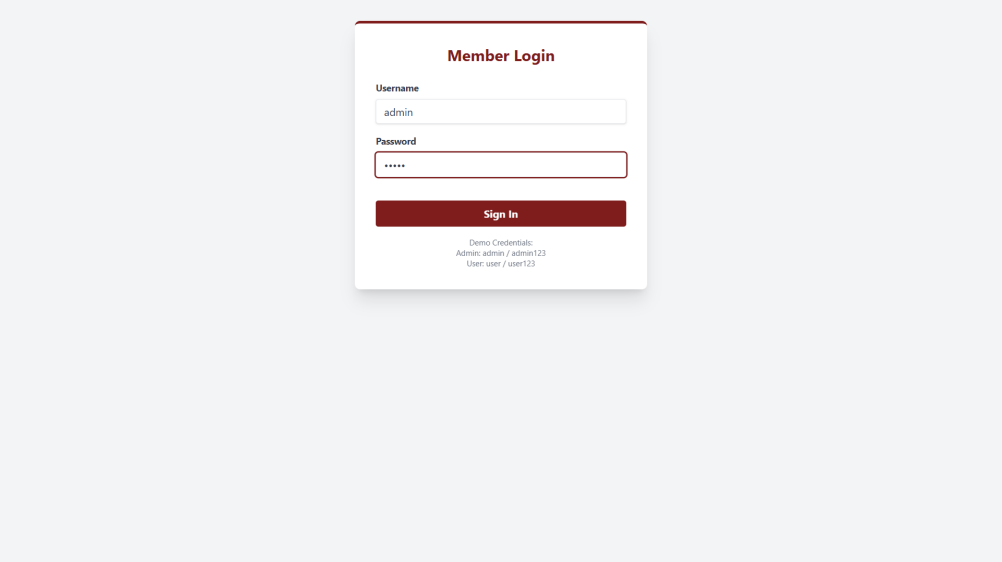


(a) Inserimento errato delle credenziali.

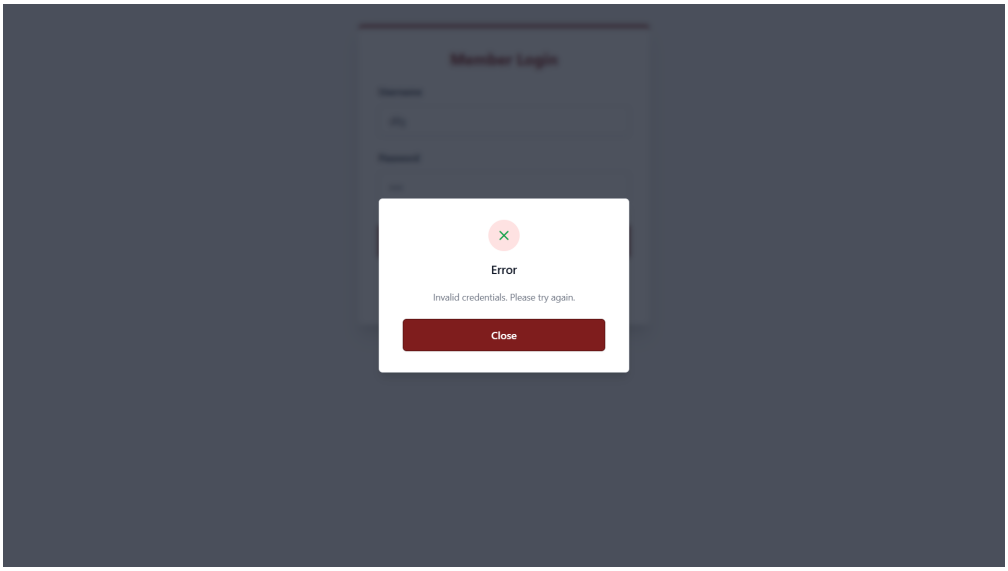


(b) Risultato dell'inserimento errato delle credenziali nella UI prima dell'applicazione dei principi.

Figura 3.4: UI prima dell'applicazione del principio 2 (Error Recognition adn Recovery).



(a) Inserimento errato delle credenziali.



(b) Risultato dell'inserimento errato delle credenziali nella UI prima dell'applicazione dei principi.

Figura 3.5: UI dopo l'applicazione del principio 2 (Error Recognition adn Recovery).

Al fine di sistematizzare l'approccio, i principi sono stati suddivisi in tre macroaree concettuali che affrontano specifiche criticità dell'interazione utente-macchina:

- Macroarea 1: Feedback di Stato e Isolamento Cognitivo
- Macroarea 2: Prevenzione degli Errori e Controllo
- Macroarea 3: Affordance Visiva e Precisione Motoria

3.6.1 Macroarea 1: Feedback di Stato e Isolamento Cognitivo

Questa categoria affronta le problematiche relative alla latenza di elaborazione del sistema e alla focalizzazione dell'attenzione. In particolare, fanno parte di questa sezione i principi:

- HCI_1_COGNITIVE_SYSTEM_STATUS: Derivato direttamente dall'Euristica 1 di Nielsen. Impone che l'interfaccia fornisca un feedback visivo inequivocabile e immediato (ad esempio, l'iniezione di uno spinner di caricamento o la disabilitazione dei bottoni attivi tramite opacità) durante le fasi di elaborazione, riducendo l'ansia da interazione dell'utente.
- HCI_7_CONTEXT_ISOLATION: Stabilisce la necessità di isolare visivamente le viste modali. Richiede l'oscuramento del background tramite backdrop overlay per forzare il focus cognitivo dell'utente sulla finestra attiva, mitigando il rischio di distrazioni visive.
- HCI_8_CUSTOM_DIALOGS: Vieta categoricamente l'utilizzo di prompt nativi del browser (come alert(), confirm() o prompt()), i quali rompono l'immersione visiva dell'applicazione. Promuove invece l'uso di dialoghi personalizzati e stilizzati per mantenere una brand experience coerente.

Analizzando nel dettaglio l'applicazione di questi principi possiamo individuare i benefici sia lato umano che lato Visual Language Model (VLM). Nel primo caso, l'applicazione dei principi riduce l'ansia da interazione fornendo feedback visivi immediati (spinner di caricamento) e forza il focus dell'utente oscurando il background durante l'apertura di modali, evitando l'uso di stranianti pop-up nativi del browser. Nel secondo, quando un VLM esegue un'azione (es. un click), acquisisce un nuovo screenshot per decidere il passo successivo. Se il sistema non mostra un evidente cambiamento di pixel (come un bottone che diventa opaco o un livello di backdrop che scurisce lo sfondo), il VLM non percepisce il cambiamento di stato e rischia di entrare in loop, ripetendo l'azione all'infinito.

3.6.2 Macroarea 2: Prevenzione degli Errori e Controllo

Questo gruppo di principi si concentra sulla gestione proattiva degli imprevisti e sulla sicurezza dei percorsi di navigazione:

- HCI_2_EXPLICIT_ERROR_TEXT: Fondato sull'Euristica 9 di Nielsen relativa al riconoscimento e recupero degli errori e sui requisiti di accessibilità WCAG. Garantisce che la segnalazione degli errori di validazione non conti esclusivamente su variazioni cromatiche (critiche per gli utenti daltonici), ma sia supportata da un testo esplicito, leggibile e iniettato direttamente nel DOM.

- HCI_4_ESCAPE_HATCHES: Diretta applicazione dell'Euristica 3 di Nielsen sul controllo e la libertà dell'utente. Assicura che ogni elemento in sovrimpressione, come modali o popup, presenti un'uscita di emergenza universale e altamente visibile (come una distinta icona "X" o un tasto "Annulla") per prevenire che l'utente si senta intrappolato nell'interfaccia.

Analogamente, anche in questa macroarea è possisbile individuare le due tipologie di benefici. Lato umano garantisce che gli errori non contino esclusivamente sul colore (pratica fondamentale per il daltonismo) ma siano supportati da testo esplicito. Inoltre, assicura che gli utenti abbiano sempre un'uscita di sicurezza visibile ("X" o "Annulla") se cliccano per errore.

I modelli VLM eccellono nell'estrazione del testo tramite OCR (Optical Character Recognition). Un bordo rosso attorno a un input è un segnale visivo debole per un'intelligenza artificiale, mentre una stringa di testo ("Email mancante") viene letta e processata perfettamente, permettendo all'Agente di correggere dinamicamente la propria traiettoria d'azione. Analogamente, le Escape Hatches evitano che l'Agente fallisca il task rimanendo intrappolato in viste modali aperte per errore.

3.6.3 Macroarea 3: Affordance Visiva e Precisione Motoria

Questa categoria colma il divario tra la percezione visiva di un elemento interattivo e l'effettiva interazione fisica:

- HCI_3_MOTOR_ACCESSABILITY_FITTS: Implementazione della Legge di Fitts sull'accessibilità motoria e il puntamento. Richiede che le aree interattive siano massimizzate in termini di dimensioni (tramite padding generosi) e ben distanziate spazialmente, minimizzando l'incidenza di click accidentali dovuti a tremori motori.

- HCI_5_VISUAL_AFFORDANCE: Sancisce l'obbligo di definire confini visivi netti per gli elementi interattivi. I bottoni devono possedere colori di sfondo o

bordi marcati che li rendano istantaneamente riconoscibili e distinguibili dal contenuto testuale statico delle circostanze.

- HCI_6_UNIVERSAL_COMPREHENSION: Stabilisce la priorità della chiarezza semantica rispetto all'estetica minimalista. Richiede che le icone per le azioni primarie, che potrebbero risultare ambigue a livello culturale o anagrafico, siano sempre accompagnate da etichette testuali esplicative che ne garantiscono la comprensione universale.

L'applicazione di questi principi ha come scopo quello di migliorare l'accessibilità motoria dell'utente umano richiedendo target di click ampi e ben distanziati, prevenendo "misclicks". Trasforma interfacce minimaliste e ambigue in sistemi dove i bottoni sono chiaramente distinguibili (sfondi, bordi) e le icone sono sempre accompagnate da testo, eliminando barriere culturali o anagrafiche.

Inoltre, i VLM generano coordinate spaziali (X, Y) per individuare gli elementi. Applicando la Legge di Fitts tramite ampi padding e gap (HCI_3), la tolleranza d'errore spaziale del modello aumenta drasticamente: anche una previsione leggermente imprecisa colpirà comunque il target. L'obbligo di inserire bordi e sfondi netti (HCI_5) permette alla Computer Vision di segmentare correttamente le bounding boxes degli elementi interattivi. Infine, l'aggiunta alle icone ambigue di etichette testuali (HCI_6) si appoggia nuovamente alla potenza dell'OCR, eliminando le allucinazioni semantiche in cui il VLM fraintende il significato di un'icona astratta.

Capitolo 4

Analisi di Affidabilità e Validazione Metodologica

L'applicazione di Large Language Models (LLM) e Visual-Language Models (VLM) all'interno della pipeline autonoma per la generazione e la valutazione di interfacce web introduce una serie di sfide metodologiche. In particolare, il calcolo del Task Success Rate (TSR) e la sua funzione di metrica oggettiva per la valutazione delle funzionalità di un sito web può essere fortemente influenzato e minacciato da controlli di validità. Un incremento del TSR potrebbe di fatto suggerire un miglioramento delle performance a valle del processo di Refinement iterativo ma, in un flusso di lavoro in cui agenti LLM e VLM interagiscono dinamicamente con il codice e l'ecosistema di verifica di completamento, si è retenuto necessario effettuare controlli di validità e affidabilità.

La necessità di condurre queste analisi nasce per rispondere alle criticità dell'architettura proposta, che sono state affrontate in fase di sperimentazione. In particolare, è possibile individuare quattro criticità fondamentali:

- **Integrità della struttura di Testing**: Poichè i test automatizzati Playwright vengono adattati e rigenerati dinamicamente alle variazioni del codice durante la fase di revisione, vi è il rischio che un incremento del TSR sia dovuto ad un indebolimento delle asserzioni e controlli piuttosto che ad un effettivo miglioramento dell'usabilità. È dunque necessario effetturare un'analisi sulla variazione dei controlli effettutati dai Test Playwright per escludere o inserire un parametro di decurtazione dovuto ad una maggiore permissitivà.

- **Causalità dell'HCI Constitution**: In cicli di ottimizzazione del codice, il modello generativo potrebbe tendere a correggere bug funzionali latenti. Risulta dunque fondamentale isolare l'impatto delle correzioni involontarie di bug funzionali rispetto al reale contributo apportato dall'applicazione della

Costituzione, dimostrando tramite un'ablazione che il miglioramento funzionale è dovuto all'utilizzo dei principici di Human-Computer Interaction.

- **Bias di Conferma e Reward Hacking**: Affidare la valutazione del codice allo stesso modello incaricato di generarlo crea un problema di circolarità. Il modello potrebbe tendere a massimizzare lo Score Costituzionale attraverso scorciatoie elusive, effettuando modifiche mirate al solo scopo di massimizzare tale punteggio. L'utilizzo di panel ibridi, comprendenti LLM esterni e valutatori umani, è un passaggio obbligatorio per garantire un'ottimizzazione oggettiva.
- **Varianza di Campionamento**: L'efficacia effettiva del ciclo iterativo di Refinement deve essere confrontata con l'esplorazione stocastica pura. È necessario dimostrare, tramite il confronto con una baseline Best-of-N, che l'architettura sviluppata non stia convergendo verso soluzioni ottime, recuperando semplicemente varianza di campionamento. Confrontando il TSR delle versioni migliori del codice con i migliori risultati di TSR ottentuti da una serie di generazioni di baseline, è possibile analizzare l'effettivo miglioramento dovuto all'applicazione del framework.

Analizzare e affrontare questi problemi di affidabilità e validazione ha come obiettivo quello di conferire maggiore solidità e trasparenza ai risultati presentati.

4.1 Validazione della Permissività dei Test Playwright

L'efficacia della Pipeline di ottimizzazione e il conseguente incremento del Task Success Rate (TSR) richiedono una validazione rigorosa dell'infrastruttura di testing automatizzato. Poiché i Test Playwright vengono rigenerati ad ogni iterazione per adattarsi alle modifiche strutturali e visive del DOM, sussiste il rischio critico che un aumento del TSR non derivi da un reale miglioramento dell'usabilità dell'interfaccia, ma piuttosto da un progressivo rilassamento dei criteri di validazione.

Un set di test eccessivamente permissivo produrrebbe dei falsi positivi, gonfiando artificialmente le metriche di successo del Visual-Language Model (VLM).

Per eseguire quest'analisi è stato preso in considerazione il dataset WebGen-Instruct definito in WebGen-Bench [14]. Un'indagine preliminare condotta su un campione limitato di 47 test aveva evidenziato un tasso di maggiore permissività del 19.1%.

Tuttavia, per garantire una maggiore solidità statistica all'argomentazione ed escludere che il fenomeno fosse limitato esclusivamente all'estremità finale del loop generativo, l'analisi è stata estesa a un campione di 101 coppie di test eseguendo un confronto diretto tra le due versioni distribuite su 19 specifiche di generazione del codice estrapolate dal dataset.

Per mappare in modo accurato l'eventuale deriva progressiva della permissività, il campionamento ha incluso gli step intermedi del ciclo di raffinamento, estraendo i dati dai round 2, 3, 4, 6, 7, 8 e 9 del loop iterativo. La classificazione della permissività (etichettata come test più permissivo, meno permissivo o equivalente) è stata condotta a quattro mani tramite annotazione indipendente.

Per quantificare il grado di accordo tra i giudici umani coinvolti nella valutazione è stato utilizzato il coefficiente Cohen's Kappa, una misura statistica che valuta l'accordo inter-giudice per dati categoriali. Il coefficiente assume valori compresi tra -1 e 1, dove 1 indica un accordo perfetto, 0 un accordo equivalente al caso, e valori negativi un disaccordo sistematico.

Per l'interpretazione dei risultati si fa riferimento alla scala proposta da Landis e Koch [30], che suddivide il coefficiente in fasce di accordo: slight (0.00-0.20), fair (0.21-0.40), moderate (0.41-0.60), substantial (0.61-0.80) e almost perfect (0.81-1.00).

In una prima fase, l'accordo inter-giudice misurato tramite il Cohen's Kappa ha restituito un valore di 0.558. Essendo tale metrica inferiore alla soglia di 0.6, la classificazione iniziale è risultata troppo soggettiva, rendendo necessaria la stesura di una rubrica di analisi formale con criteri quantificabili prima di procedere a una seconda annotazione. La rubrica si è basata su due parametri architetturali:

- Tipo di Check (Copertura Semantica del Task): Valutazione delle fasi essenziali validate dal test, suddivise in Azione (presenza dell'elemento interattivo), Stato/Dati (presenza dei dati attesi) e Feedback (validazione del risultato visuale). L'omissione di una di queste categorie essenziali rispetto al codice definito nell'altro Stato classifica il test come più permissivo. Ad esempio, un test che si limita a verificare il click sul tasto "Save" ignorando la comparsa del container '#toast-success' risulta oggettivamente più debole.

- Numero Logico di Assertion (Equivalenza per Implicazione): Il numero assoluto di asserzioni 'await expect(...)' non determina univocamente la severità del test. Se un'asserzione risulta essere un prerequisito architetturale di una successiva (ad esempio, l'esistenza di un elemento <ul class="cart-list"> implica già semanticamente la navigazione corretta nella sezione carrello), la rimozione del controllo ridondante non rende il test più permissivo.

L'applicazione di questa rubrica rigorosa ha innalzato il Cohen's Kappa a 0.619, indicando un accordo sostanziale tra gli annotatori e validando la metodologia di classificazione.

Dall'analisi congiunta è emerso che 27 test su 101 risultavano effettivamente più permissivi nella versione del codice migliorato. Per valutare l'impatto reale di questo fenomeno sul TSR globale, si è rivelato fondamentale stratificare i test annotati per difficoltà del task. Utilizzando come proxy di difficoltà il TSR baseline,

i task sono stati classificati in 'Easy' (già superati nel codice baseline) e 'Hard' (falliti nel baseline e superati nella versione aggiornata, definiti come 'flipped').

L'obiettivo di questa stratificazione è verificare se la permissività si concentri sui task difficili, dove il peso sui falsi positivi è nettamente superiore. Sui 101 test analizzati, 52 sono stati classificati come 'Hard'. All'interno di questo sottoinsieme critico, 14 test sono risultati più permissivi, 6 meno permissivi, 18 equivalenti e 12 indefiniti a causa di discrepanze non risolvibili nella valutazione.

Per effettuare una valutazione della validità di questa analisi è stato condotto uno studio su una partizione del dataset WebGen-Instruct costituita da 100 siti. Questo sottogruppo è stato sottoposto alla fase di Setup e alla totalità della fase di Refinement iterativo, ottenendo per la baseline un TSR di 30.2% , mentre per la versione migliore un TSR di 68.2%.

Assumendo un worst-case scenario in cui tutti i 27 test più permissivi abbiano generato falsi positivi, applicando i risultati dell'analisi al TSR calcolato sulla partizione del dataset, esso subirebbe una flessione al 49.97%.

Tuttavia, un'analisi manuale mirata della Task Evaluation ha dimostrato che solo 14 di questi test alterano effettivamente la metrica, posizionando il TSR corretto al 58.74%. Per garantire la massima trasparenza scientifica, il risultato reale viene presentato non come stima centrale, ma come intervallo di confidenza: TSR effettivo $\in [49.97\%, 58.74\%]$.

Anche considerando il limite inferiore di tale intervallo, l'applicazione della pipeline conferma un aumento netto e statisticamente significativo del Task Success Rate di circa il +20 punti percentuali rispetto alla baseline.

4.2 Correzione bug indipendente vs. Applicazione della HCI Constitution

Nel quadro della validazione metodologica, un aspetto critico da analizzare riguarda il peso della correzione involontaria di bug funzionali durante i cicli di rigenerazione del codice. Sussiste infatti il rischio che l'aumento del Task Success Rate (TSR) registrato a valle del processo sia il risultato di un refactoring implicito che ripara la logica del codice, rendendo di fatto marginale il reale contributo dei principi di usabilità definiti dalla HCI Constitution.

Se così fosse, i miglioramenti misurati deriverebbero da una maggiore robustezza del codice sorgente e non da un'ottimizzazione dell'esperienza utente. Per isolare e misurare empiricamente l'influenza diretta della Costituzione, è stato condotto uno studio di ablazione su un sottoinsieme composto dai primi 40 siti web generati dal dataset WebGen-Instruct.

L'esperimento ha previsto l'esecuzione di un ciclo di raffinamento completo in cui il feedback mirato della Costituzione (normalmente estratto dal file 'best_state.json') è stato disattivato e sostituito con una direttiva generica di miglioramento del codice. Nello specifico, il prompt fornito al Large Language Model è stato alterato omettendo le valutazioni sui principi e inserendo la seguente istruzione statica:

```
"Your task is to REFACTOR and IMPROVE the existing
HTML+Tailwind+JS web application artifact to enhance
code quality, maintainability, and execution robustness,
following industry best practices."
```

I risultati al termine del ciclo di raffinamento hanno evidenziato una netta divergenza nel comportamento del modello a seconda dell'approccio utilizzato, come sintetizzato in 4.1:

Tabella 4.1: Feedback Genererico vs. Applicazione HCI Constitution

	TSR	**Score Costituzionale Medio**
Baseline	24.23%	23.35
HCI Constitution	57.27%	34.20
Feedback Generico	65.20%	25.77

L'analisi di queste metriche rivela dinamiche fondamentali sul funzionamento dell'architettura. Osservando la conformità all'usabilità, il prompt di refactoring generico produce un aumento del tutto marginale dello Score Costituzionale Medio (+2.42). Questo dato dimostra inequivocabilmente che le istruzioni mirate della Costituzione sono un requisito indispensabile per forzare l'allineamento dell'interfaccia alle euristiche e alle direttive di usabilità e accessibilità, garantendo un incremento significativo della qualità dell'interfaccia (+10.85).

Al contrario, focalizzandosi sul TSR grezzo, si osserva che l'approccio con feedback generico supera quello basato sulla Costituzione (65.20% contro 57.27%). Questa discrepanza si spiega col fatto che un prompt focalizzato sulla robustezza del codice spinge il modello a concentrare le iterazioni sulla correzione dei bug funzionali e sul completamento delle logiche non funzionanti nella prima generazione, trascurando totalmente l'accessibilità e l'usabilità (come testimoniato dal basso incremento dello Score Costituzionale).

La pipeline guidata dalla HCI Constitution, invece, vincola il modello a operare strettamente sulla UI/UX (User Interface / User Experience), inibendo interventi diretti di profondo debug strutturale. L'ablazione conferma pertanto la rilevanza cruciale del sistema di principi: il miglioramento del TSR registrato sotto la guida

della Costituzione non è un artefatto dovuto alla mera riparazione del codice, ma è la diretta conseguenza di un'interfaccia più accessibile e coerente. È proprio questa superiore qualità visiva e interattiva che permette al Visual-Language Model (VLM) di interpretare correttamente i segnali della pagina e completare i task senza incontrare blocchi operativi.

4.3 Mitigazione del Reward Hacking

A valle della fase di rigenerazione del codice, il sistema esegue iterativamente un processo di Task Evaluation e di autocritica basato sulla HCI Constitution. Una criticità metodologica intrinseca a questa architettura riguarda il rischio di Reward Hacking. Poiché il medesimo Large Language Model (Qwen3.5-27B [22]) è incaricato sia della generazione del codice sia della valutazione del rispetto dei principi, sussiste il pericolo che il modello converga verso soluzioni atte a massimizzare artificialmente il punteggio, piuttosto che apportare un reale beneficio qualitativo all'esperienza utente.

Questo fenomeno distorsivo tende a manifestarsi principalmente in due direzioni:

- Ottimizzazione superficiale: Il modello predilige modifiche puramente estetiche o stilistiche che risultano facilmente interpretabili come conformi alla Costituzione, trascurando la solidità dell'architettura logica sottostante.
- Comportamenti elusivi: Il modello opta per l'eliminazione diretta di componenti funzionali (ad esempio, la rimozione totale di un alert anziché la sua conversione in una modale accessibile) al solo scopo di eludere le penalità previste dalle metriche di valutazione.

La circolarità di questo processo - in cui il generatore "anticipa" le preferenze del giudice essendo di fatto la stessa entità - rischia di creare un bias di conferma, producendo un incremento dello score ($\Delta Score$) che non trova riscontro in un effettivo miglioramento del prodotto finale.

Per validare la robustezza della pipeline e respingere speculazioni metodologiche, limitarsi all'utilizzo di un singolo giudice esterno per confermare i risultati non è sufficiente. Modelli diversi possono infatti condividere bias latenti derivanti da fasi di training simili. Per blindare scientificamente l'analisi, è stato pertanto istituito un panel di giudici esterni composto da tre modelli addestrati con tecniche di Reinforcement Learning from Human Feedback (RLHF) e preferenze strutturalmente diverse: Claude Opus 4.7 [31], GPT-4o [32] e Gemini Pro 3.1 [24].

L'indagine condotta su un campioni rappresentativi di siti web estratti da WebGen-Instruct per garantire rilevanza statistica, si è articolata diversificando i formati di valutazione. Oltre alla critica guidata rigidamente dai vincoli della Costituzione, ai giudici è stato sottoposto un Blind A/B Preference Test in formato

free-form, privo di rubriche predefinite, per verificare se il miglioramento venisse percepito in modo netto anche al di fuori del framework teorico di partenza. Infine, per accertare che i guadagni prestazionali non fossero concentrati esclusivamente su principi facili da "gamificare", i risultati sono stati disaggregati analizzando la distribuzione del $\Delta Score$ principio per principio.

4.3.1 Deviazione Standard e Affidabilità Intra-Judge

In una fase preliminare dell'analisi, il modello Qwen3.5-27B e Gemini Pro 3.1 sono stati utilizzati per valutare 20 artefatti relativi al codice originale (Base Code) e al codice migliorato (Best Code). I risultati dell'analisi delle due versioni sono riportati in Figura 4.1, 4.2 e Tabella 4.2.

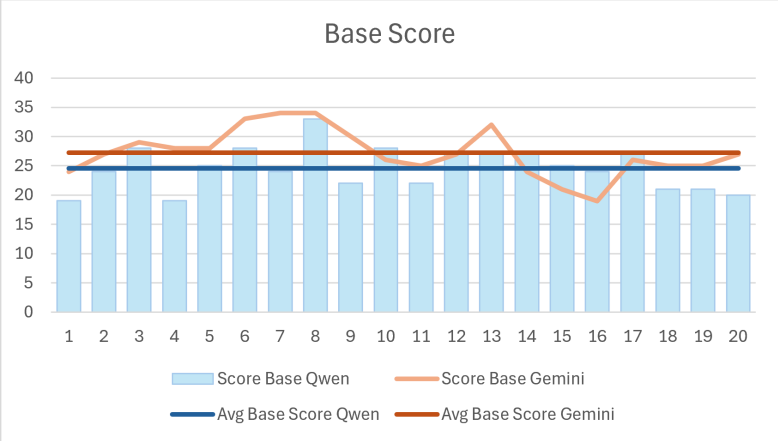


Figura 4.1: Andamento Score Costituzionale versione Base

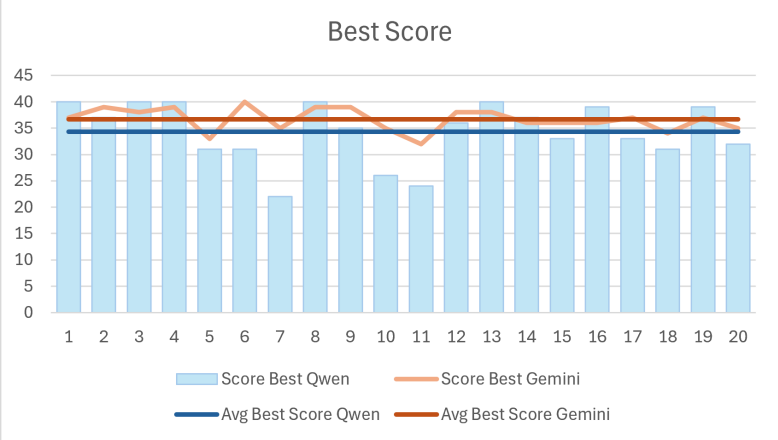


Figura 4.2: Andamento Score Costituzionale versione Best

Tabella 4.2: Score Costituzionale Medio Qwen3.5-27B e Gemini Pro 3.1

Modello	Score Costituzionale Base	Score Costituzionale Best
Qwen3.5-27B	24.55	34.30
Gemini Pro 3.1	27.20	36.65

Il test condotto su singola valutazione ha evidenziato un discostamento di +2.65 punti nelle medie assolute dello Score Costituzionale relative al Base Code rispetto a Qwen3.5-27B e un aumento di +9.45 punti tra le medie degli Score Costituzionali dei codici Base e Best. A fronte di un $\Delta Score$ di +9.75 registrato da Qwen, i risultati preliminari di Gemini suggeriscono un allineamento tra i due valutatori, fornendo una prima indicazione dell'assenza di Reward Hacking.

Tuttavia, per mitigare la presenza di bias legati alle naturali fluttuazioni di output tipiche dei Large Language Models, è stato condotto un esperimento specifico volto a calcolare la varianza e la deviazione standard intra-giudice del modello Gemini Pro 3.1. A tale scopo, il campione di 20 siti estratti dal dataset WebGen-Instruct, è stato sottoposto a 5 valutazioni consecutive. L'analisi sui risultati del Base Code ha prodotto i risultati mostrati in Tabella4.3:

Tabella 4.3: Analisi Varianza, Deviazione Standard e MoE di Gemini Pro 3.1

Avg Score	Avg Variance	Standard Deviation	MoE
30.15	1.73	1.32	3.65

In seguito a questo test ripetuto, l'Avg Score è passato da 27.20 a 30.15, comportando di conseguenza un aumento del bias di Gemini a +5.6 rispetto alla media di 24.55 ottenuta da Qwen.

Per completezza e rigore metodologico, le chiamate API per le 5 valutazioni ripetute sono state rieseguite anche sulla versione Best del codice. Questa operazione ha prodotto un valore medio di 38.4, individuando un incremento di +8.25 rispetto alla media registrata per il codice Base e un discostamento di +4.1 rispetto a Qwen.

Rispetto alla media calcolata in precedenza sui codici Best, si è osservato un incremento di +1.75, un valore che rientra pienamente entro i limiti della deviazione standard e del Margine di Errore massimo appena calcolati per i codici Base.

I valori di varianza media e deviazione standard dimostrano che, valutando il medesimo artefatto in istanze ripetute, il modello tende a restituire punteggi con un'oscillazione media di circa 1.3 punti rispetto alla propria media.

Alla luce di questi dati, è possibile concludere che il salto prestazionale di +8.25 è ampiamente superiore sia alla deviazione standard (1.32) sia al Margine di Errore

massimo (3.65). Questo scarto conferma in modo oggettivo e statisticamente significativo che le modifiche applicate al codice hanno avuto un impatto positivo e misurabile da parte del modello Gemini Pro 3.1. Inoltre, tale analisi fornisce solide basi per escludere l'insorgenza di fenomeni di Reward Hacking all'interno del modello Qwen3.5-27B.

Le valutazioni di quest'ultimo risultano infatti in linea con le misurazioni di Gemini Pro, mostrando una differenza minima nel $\Delta Score$ pari a soli +1.5 (+9.75 per Qwen contro +8.25 per Gemini Pro). Questa lieve divergenza può essere giustificata dal fisiologico raggiungimento di un limite superiore dello Score Costituzionale (effetto di saturazione) da parte di Gemini nella valutazione dei Best Code, fenomeno riconducibile al bias di clemenza individuato per tale modello nell'analisi dei Base Code.

4.3.2 Blind A/B Preference Test tramite giudici LLM

Per consolidare la validità della pipeline al di fuori dei rigidi vincoli imposti dalla Costituzione, è stato eseguito un Blind A/B Preference Test in modalità free-form. Un campione di 50 coppie di artefatti, costituite dalla versione del codice originale (Base) e dalla versione migliorata (Best), è stato sottoposto al vaglio di tre modelli esterni: GPT-4o, Claude Opus 4.7 e Gemini Pro 3.1. Ai modelli è stato richiesto di esprimere una preferenza basata esclusivamente su intuizioni generali relative al design, all'usabilità e alla chiarezza visiva.

Per garantire l'imparzialità dell'esperimento, non è stata fornita alcuna rubrica valutativa né alcuna indicazione in merito alla provenienza del codice generato. Al fine di assicurare il massimo rigore metodologico, sono state implementate due contromisure specifiche:

- Bias di lunghezza: sono state selezionate alcune coppie di siti con una differenza di lunghezza di token inferiore al 10%.
- Bias di posizione: l'ordine di presentazione degli artefatti ai giudici LLM è stato sottoposto a randomizzazione.

I risultati dell'esperimento hanno mostrato una preferenza netta per la versione Best del codice, scelta nel 100% dei casi da GPT-4o e Gemini Pro 3.1, e nel 96% dei casi da Claude Opus 4.7. Questa marcata convergenza dimostra come l'ottimizzazione apportata al codice sia chiaramente percepibile anche da valutatori che operano in cieco rispetto ai principi della Costituzione.

In particolare, il 4% di preferenze accordato all'artefatto originale da parte di Opus 4.7 evidenzia come il modello valuti gli artefatti in modo analitico e imparziale, non essendo affetto da un bias a favore della versione Best. Un'analisi dettagliata delle due casistiche in cui Opus ha preferito il Base Code rivela dinamiche interessanti. Nel primo caso, la variazione dello Score Costituzionale tra le due versioni

risultava pressoché nulla (appena 2 punti di differenza), a indicare che il codice non aveva subito modifiche sostanziali durante il ciclo iterativo di miglioramento. Nel secondo caso, il modello ha eseguito un confronto esplicito dei pro e dei contro delle due versioni, portandolo a ritenere marginalmente superiore il Base Code.

Complessivamente, le evidenze emerse dal Blind A/B Preference Test si pongono come un ulteriore pilastro metodologico: oltre a confermare l'efficacia della pipeline nel migliorare tangibilmente il design e l'usabilità, forniscono un'ulteriore e convergente prova a supporto dell'immunità del modello Qwen3.5-27B da fenomeni di Reward Hacking.

4.3.3 Validazione del Blind A/B Preference Test tramite Panel Umano

Per mitigare ulteriormente i potenziali bias e l'incertezza intrinseca derivante dall'utilizzo esclusivo di Large Language Models come giudici finali, l'esperimento di preferenza blind è stato esteso introducendo un panel di valutatori umani.

Tale step è stato concepito con lo scopo di stabilire una solida Ground Truth in merito alle preferenze di usabilità e design, permettendo di quantificare matematicamente l'allineamento tra il giudizio algoritmico e la percezione visiva umana, escludendo in modo definitivo l'incidenza di fenomeni di Reward Hacking.

Un sotto-campione rappresentativo di 20 coppie di siti web (versione Base e versione Best) è stato sottoposto a un panel composto da 3 giudici umani. Al fine di mantenere un'assoluta coerenza metodologica con l'esperimento automatizzato, i valutatori hanno operato anch'essi in modalità blind, senza conoscere l'origine del codice analizzato, ed esprimendo una preferenza basata esclusivamente su criteri di design generale, usabilità e chiarezza visiva. L'analisi dei 60 voti complessivi ha evidenziato un accordo unanime (3 giudici su 3) per 17 coppie di siti su 20, mentre per le restanti 3 si è registrato un accordo parziale, risolto tramite voto di maggioranza (2 a 1). Aggregando le preferenze per ciascuno dei 20 siti tramite il voto di maggioranza, la Ground Truth risultante ha decretato una schiacciante preferenza per le versioni ottimizzate:

- Versione Best preferita: 19 siti (95% del campione)
- Versione Originale preferita: 1 sito (5% del campione)

Questi risultati sono stati incrociati con le valutazioni precedentemente effettuate dai tre modelli esterni (GPT-4o, Claude Opus 4.7 e Gemini Pro 3.1) sugli stessi 20 siti, calcolando l'Agreement Ratio, ovvero la percentuale di casi in cui la scelta del modello ha coinciso con il consenso umano.

Tabella 4.4: Agreement Ratio tra i tre modelli e il panel umano

Modello	Preferenza Best %	Accordo Consenso Umano
GPT-4o	100%	95%
Claude Opus 4.7	96%	90%
Gemini Pro 3.1	100%	95%

Dall'analisi incrociata emerge che i modelli GPT-4o e Gemini Pro 3.1, che nel test globale avevano un 100% di preferenza, ottengono matematicamente un accordo con gli umani del 95% (avendo individuato tutti i 19 siti in cui anche gli umani hanno preferito la versione ottimizzata). Allo stesso modo la percentuale di Opus 4.7 mantiene un accordo del 90%.

L'elevato Agreement Ratio registrato dimostra che i punteggi e le preferenze assegnate dagli LLM non sono un artefatto dovuto al Reward Hacking. Al contrario, il fatto che i modelli ottengano un accordo superiore al 90% con il panel umano certifica che le ottimizzazioni apportate dalla pipeline su componenti di UI/UX si traducono in un reale e oggettivo miglioramento qualitativo dell'esperienza utente, percepibile e confermato dall'occhio umano.

4.3.4 Bulk A/B Preference Test in modalità Pairwise

Un'ulteriore validazione, mirata a escludere in via definitiva la presenza di Reward Hacking nelle scelte del modello Qwen3.5-27B, è rappresentata dal Bulk A/B Preference Test. Nel dettaglio questa analisi si basa sulla struttura del Blind A/B Preference Test: dati i 3 modelli Gemini Pro 3.1, Opus4.7 e GPT-4o è stata eseguita un'analisi di preferenza sui principi della Costituzione tra gli artefatti Base e Best di 50 siti web in modalità Pairwise.

L'abbandono dello score assoluto in favore di un approccio Pairwise è scaturito dall'analisi del comportamento di GPT-4o e Opus4.7. In fase preliminare, questi modelli hanno mostrato una marcata tendenza accondiscendente, assegnando punteggi quasi massimi (35-40 su 40) già al codice Base, generando un effetto saturazione che annullava matematicamente la possibilità di misurare il miglioramento.

Espandendo l'approccio adottato nel test blind in free-form, questa analisi ha valutato simultaneamente gli artefatti Base e Best di 50 siti web in relazione agli 8 principi specifici della HCI Constitution. I tre modelli impiegati come giudici sono stati istruiti a dichiarare, per ciascun principio, una preferenza netta (Best o Original) oppure un pareggio (Tie). I risultati dell'analisi sono riportati nella Tabella 4.5.

Tabella 4.5: Bulk A/B Prefernce Test

Modello	Preferenza Best	Prefernza Original	Tie
GPT-4o	165	0	235
Claude Opus 4.7	188	1	211
Gemini Pro 3.1	176	1	223

I dati indicano una preferenza per la versione Best nel 44% dei casi per Gemini, nel 47% per Opus e nel 41.25% per GPT-4o, a fronte di preferenze per il Base pressoché nulle.

Tramite l'analisi per-principio dei tre modelli impiegati come giudici i risultati del Bulk A/B Preference Test sono stati aggregati per evidenziare la distribuzione delle preferenze (Best e Tie) su ciascuno degli 8 principi della Costituzione. Il breakdown analitico delle valutazioni è riportato nella Tabella 4.6.

Tabella 4.6: Breakdown analitico delle Preferenze per-principio

Id	Gem B	Gem T	Opus B	Opus T	GPT B	GPT T
1	35	15	35	15	34	16
2	27	23	27	23	26	24
3	37	13	37	13	32	18
4	14	36	22	28	13	37
5	31	19	32	18	23	27
6	15	34	16	33	17	33
7	10	40	10	40	13	37
8	8	42	9	41	9	41

Dall'analisi della distribuzione è possibile isolare i principi che hanno riscontrato una maggiore preferenza nella versione Best del codice generato.

Considerando il campione di 50 coppie di artefatti (codice Base vs codice Best), emerge che per i principi 1, 2, 3 e 5 si è registrata una netta preferenza per la versione aggiornata, superando il 50% delle assegnazioni di vittoria, fatta eccezione per il principio 5 valutato da GPT, che si attesta al 46%. Al contrario, i principi 4, 6, 7 e 8 mostrano una percentuale di preferenza per il Best marginale.In particolare, per i principi 7 e 8 le valutazioni tendono al Tie in circa l'80% dei casi in tutti e tre i modelli.

Per validare questa tendenza, le preferenze per-principio sono state incrociate con gli score medi calcolati nella precedente fase di analisi della Deviazione Standard di Gemini 3.1. Il confronto è stato condotto sul campione dei 20 siti comuni a

entrambe le analisi, garantendo così coerenza metodologica. Nella Tabella 4.7 vengono riportate le medie per-principio degli score, calcolate su 5 iterazioni per studiare le fluttuazioni intrinseche del modello Gemini.

Tabella 4.7: Medie per-principio Gemini Pro 3.1

Principio	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Avg Base	3.05	2.53	4.08	4.34	3.42	3.85	5	3.87
Avg Best	4.67	4.75	4.88	4.95	4.72	4.45	5	5
Avg Diff	1.62	2.22	0.80	0.61	1.3	0.60	0	1.13

L'esame degli score medi per-principio sul codice Base e Best fornisce la seguente chiave di lettura: i principi 3, 4, 6, 7 presentano una differenza tra le medie inferiore ad 1. Nello specifico, il principio 7 registra un punteggio medio perfetto (5 su 5) già nella versione Base. Di contro, i principi 1, 2, 5 e 8 mostrano differenza di media superiore. Per quanto riguarda i principi 4, 6 e 7 il risultato dell'analisi conferma l'andamento delle preferenze: uno scarto minimo sullo score, inferiore alla deviazione standard del modello Gemini (pari a circa 1.3), giustifica la forte presenza di Tie definiti dai tre modelli.

Analizzando le medie degli score dei principi 3 e 8, emerge una tendenza peculiare nella correlazione tra score e preferenze. Si potrebbe erroneamente dedurre che l'aumento dello score medio per il principio 8 derivi da un miglioramento omogeneo del codice su tutti i siti. Tuttavia, disaggregando gli score nel dettaglio, si osserva che nel 75% dei casi analizzati il principio 8 ottiene un punteggio già pari a 5, mentre nei restanti 5 casi lo score crolla a valori pari o inferiori a 1. Questa forte polarizzazione potrebbe essere scambiata per un errore di valutazione delle preferenze da parte del modello; tuttavia, ricordando che la pipeline interviene modificando il codice prioritariamente sui principi con score inferiore, l'esame della baseline rivela che il principio 3 presenta punteggi di partenza mediamente più bassi e più distribuiti rispetto all'8. Di conseguenza, l'elevato numero di Tie per il principio 8 è giustificato dalla massiccia presenza di score perfetti in partenza (effetto tetto), nonostante presenti una differenza di media complessiva alta (causata esclusivamente dal recupero di quei pochi casi prossimi allo zero).

Viceversa, i valori sistematicamente inferiori del principio 3 nella baseline giustificano la presenza di un win-rate maggiore per il codice Best, poiché in quel frangente c'era un margine di ottimizzazione effettivamente sfruttabile.

Alla luce di questi riscontri, si può concludere che l'elevato tasso di Tie registrato (attorno al 50% delle valutazioni complessive) non riflette un'incapacità discriminatoria da parte dei modelli valutatori, bensì un andamento metodologicamente corretto e fisiologico. Esso rappresenta la diretta conseguenza di un effetto di saturazione: i giudici confermano il pareggio sui principi in cui il codice

Base possedeva già uno standard qualitativo elevato (4, 6, 7, 8), concentrando le preferenze per il codice migliorato unicamente sui principi (1, 2, 3, 5) dove il margine di perfezionamento risultava effettivamente sfruttabile dall'architettura di generazione.

Di conseguenza, questi dati continuano ad affermare l'assenza di Reward Hacking nella valutazione del modello Qwen3.5-27B, utilizzato sia come generato del codice che come critico.

4.4 Validazione tramite Baseline Best-of-N vs. Refinement Iterativo

Per completare la validazione metodologica e dimostrare l'effettiva utilità della pipeline, è stata introdotta un'ulteriore metrica di controllo: la baseline Best-of-N.

In letteratura (es. nei lavori su Self-Refine), questa è considerata la baseline più rigorosa e indispensabile da superare nei processi di iterative refinement. L'obiettivo è escludere che il loop iterativo stia semplicemente recuperando varianza di campionamento (sampling variance). Se la generazione di N versioni indipendenti (senza alcuna fase di critica) raggiungesse soglie di Task Success Rate (TSR) elevate (es. attorno al 60%), significherebbe che il loop iterativo non sta apportando un valore aggiunto sostanziale. Al contrario, se il tetto massimo del Best-of-N si arresta a valori nettamente inferiori (es. 45%), si certifica che il lavoro di auto-critica e correzione guidata sta compiendo una reale ottimizzazione logico-strutturale.

L'analisi è stata condotta su un campione di 40 siti web estratti dal set di WebGen-Instruct. Per ciascun sito, il modello Qwen3.5-27B ha generato 10 versioni indipendenti del codice (Best-of-10) partendo dal medesimo prompt iniziale e dalle medesime direttive, estraendo poi la versione, che a termine della fase di Setup, è in grado di massimizzare il TSR.

Il risultato grezzo ottenuto al termine di questa fase esplorativa ha evidenziato un TSR massimo del 54.33%. Come dimostrato dalla precedente indagine sulla permissività dei Test Playwright, per ottenere una valutazione corretta non ci si può limitare al dato puntuale grezzo, in quanto un set di test più permissivo rischia di generare falsi positivi portando a una sovrastima ottimistica del TSR. Pertanto, è metodologicamente più corretto depurare i risultati da questo effetto e presentarli tramite intervalli specifici.

Applicando i parametri di decurtazione derivati dall'analisi della tolleranza dei test, le metriche aggiornate delineano il seguente scenario comparativo:

- Baseline (Singola Iterazione - Base): Il TSR originale si attesta a un valore grezzo del 34.61%, corrispondente a un range reale ricalibrato di [25.26%, 29.41%].

- Baseline Best-of-10 (Solo Sampling): Il picco grezzo del 54.33% si ridimensiona a un range effettivo di [39.66%, 46.18%].
- Pipeline Iterativa (Refinement con Critica - Best): Al termine delle 10 iterazioni di refinement sul codice originale, il TSR raggiunge un range consolidato di [49.83%, 58.03%].

Il confronto tra i range evidenzia come l'approccio Best-of-10 garantisce un fisiologico miglioramento rispetto alla singola generazione (passando da un tetto del 29.41% a uno del 46.18%), giustificato dall'esplorazione stocastica dello spazio latente da parte dei modelli, in quanto anche i risultati del TSR sono calcolati sulla base delle azioni prese dal VLM Holo2-8B. Tuttavia, il dato cruciale risiede nel fatto che il limite superiore del Best-of-10 (46.18%) risulta strettamente inferiore al limite minimo garantito dalla pipeline iterativa (49.83%). Questo divario matematico certifica in modo inequivocabile che il loop iterativo non si sta limitando a estrarre la variante stocasticamente più fortunata, ma elabora un effettivo refinement del codice che aggiunge reale valore, confermando la robustezza dell'approccio.

Capitolo 5

Risultati sperimentali e Valutazioni

Questo capitolo presenta i risultati sperimentali derivanti dell'applicazione della HCI Constitution alla pipeline iterativa di Refinement del codice frontend per siti web. L'analisi valuta l'efficacia dell'applicazione degli 8 principi della Costituzione nelle fasi iterative di critica e generazione, con lo scopo di migliorare le User Interfaces (UI) e la User Experience (UX), valutando le funzionalità del sito ad ogni iterazione.

5.1 Setup degli esperimenti e dataset utilizzati

Come definito nel capitolo relativo a Metodologie e Setup Sperimentale, questo lavoro di tesi è stato condotto utilizzando le infrastrutture di calcolo (cluster) del CINECA e del Politecnico di Torino. L'interto processo di generazione e refinement del codice si sviluppa all'interno di un ambiente chiuso (Sandbox), privo di connessione internet per il download a runtime di librerie o asset esterni. Per ovviare a questa limitazione, i pacchetti necessari al corretto funzionamento della pipeline e allo styling dei siti web (ad esempio, la libreria Tailwind CSS) sono stati pre-scaricati e integrati nativamente all'interno della Sandbox.

È stato eseguito un ciclo completo di aggiornamento del codice per ciascun sito, comprendente la fase iniziale di Setup e le 10 iterazioni di Refinement. Al fine di quantificare il vantaggio apportato dall'architettura proposta, è stato impiegato il dataset di test introdotto in WebGen-Bench [14]. Si ricorda che il dataset è costituito da 101 direttive di generazione del codice per i siti web e 647 task interattivi, con relativi risultati attesi, da eseguire per definire la funzionalità dei siti e il loro aspetto visivo.

5.2 Risultati dell'applicazione del Refinement Iterativo

In questa sezione vengono riportati i risultati della critica Costituzionale effettuata sul codice generato dal modello Qwen3.5-27B [22] e della valutazione funzionale tramite il modello Holo2-8B [23], che interagisce con i siti web come un utente umano.

Per ogni istruzione di generazione del dataset è stata eseguita la fase di Setup della pipeline, come mostrato in Figura 3.1, effettuando la generazione del codice, la Task Evaluation e la prima critica Costituzionale sul codice base (Base Code). Al seguito della fase di Setup, il codice sorgente è stato sottoposto a 10 iterazioni del loop di refinement, ciascuna comprendente la rigenerazione del codice seguendo il feedback costituzionale, la riesecuzione della Task Evaluation e l'eventuale nuova critica seguendo la HCI Constitution nell'eventualità in cui il Task Success Rate (TSR) calcolato del nuovo codice fosse maggiore del TSR dello stato attualmente migliore (Best Code), come mostrato in Figura 3.2.

La Tabella 5.1 mostra i risultati relativi al Base Code, al termine della fase di Setup valutando le 101 istruzioni di generazione e i 647 task del dataset.

Tabella 5.1: TSR Grezzo e Score Costituzionale Medio - Fase di Setup

Task Completati	TSR Grezzo	Score Costituzionale Medio
244	37.71%	26.27

Come sottolineato nel precedente capitolo, la valutazione del TSR non può limitarsi al dato puntuale grezzo, ed è necessario applicare i parametri di decurtazione derivati dall'analisi della tolleranza (permissività) dei Test Playwright. Di conseguenza, al termine della fase di Setup, il TSR $\in [27.52\%, 32.05\%]$.

A seguito dell'esecuzione delle 10 iterazioni di Refinement Costituzionale, si riscontra un netto miglioramento nel TSR e nello Score Costituzione Medio estratti dalle versioni migliori dei siti. La Tabella 5.2 presenta i dati grezzi al termine della valutazione.

Tabella 5.2: TSR Grezzo e Score Costituzionale Medio - Fase di Refinement

Task Completati	TSR Grezzo	Score Costituzionale Medio
421	65.07%	33.36

Analogamente alla fase di Setup, è necessario applicare i parametri di decurtazione e per la versione migliore dei codici il TSR $\in [47.50\%, 55.31\%]$.

Dal confronto dei risultati si evince un aumento del completamento con successo dei task di +177, un aumento dello Score Costituzionale Medio di +7.09 punti e un incremento del TSR di circa 20 punti percentuali. Il confronto tra TSR e Score Costituzionale dei 101 siti tra il Base Code e il Best Code è riportato nelle Figure 5.1 e 5.2.

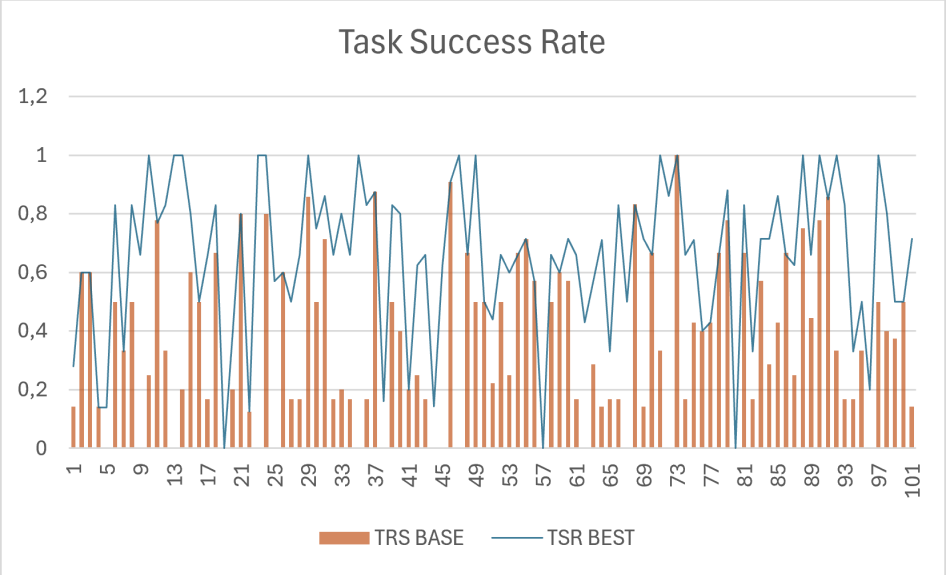


Figura 5.1: Confronto Task Success Rate Base Code - Best Code

A titolo di confronto, il lavoro di WebGen-Agent [4], che ha adottato un miglioramento iterativo basato su feedback funzionali derivati dall'esecuzione di un GUI-Agent e su un'analisi visiva statica effettuata tramite VLM, ha ottenuto un incremento del TSR, sfruttando come modello Claude-3.5-Sonnet, da una baseline del 26.4% al 51.9%. Parallelamente, l'applicazione del loro metodo di training definito Step-GRPO ha dimostrato di poter aumentare l'accuratezza del modello Qwen2.5-7B-Instruct dal 38.9% al 45.4%.

A differenza di tale approccio, il caso di studio oggetto di questo lavoro si è concentrato esclusivamente sull'ottimizzazione dell'usabilità e del design per il raffinamento del codice frontend, escludendo un diretto bug fixing delle funzionalità del codice e un miglioramento visivo fornito dall'analisi di un VLM, ottenendo dei risultati in linea con i lavori precedentemente svolti.

L'analisi di questi risultati evidenzia come l'integrazione della HCI Constitution all'interno della pipeline generativa e di refinement iterativo consenta di innalzare significativamente la qualità del codice prodotto. I siti web finali mostrano miglioramenti sostanziali in termini di funzionalità, usabilità e interattività, confermando la solidità e la robustezza dell'approccio proposto.

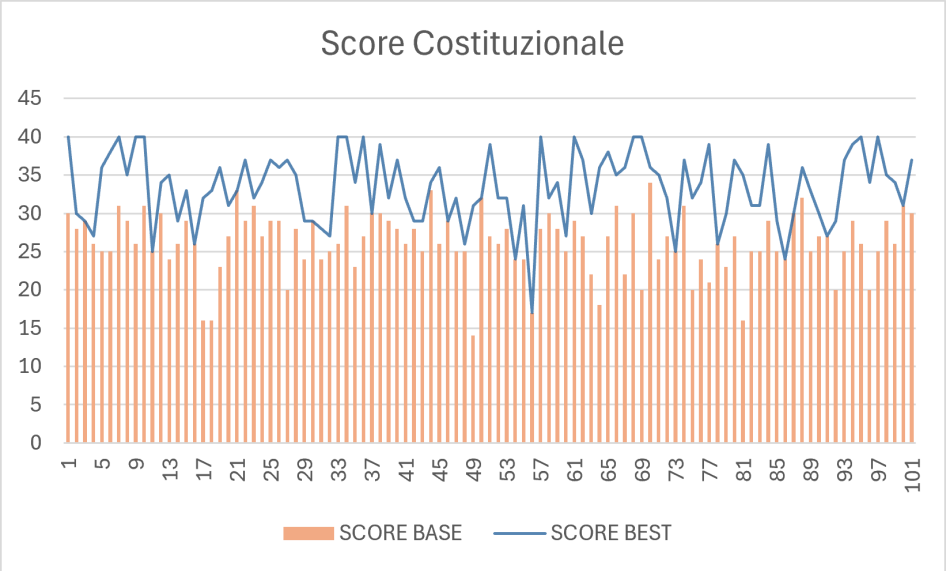


Figura 5.2: Confronto Score Costituzionale Base Code - Best Code

Questi dati dimostrano che la generazione automatica di interfacce a partire da direttive in linguaggio naturale non può limitarsi alla sola correttezza funzionale, ma deve necessariamente integrare criteri di usabilità e di interazione uomo-macchina per garantire un prodotto finale realmente efficace e centrato sull'utente.

5.3 Costo computazionale e guadagno TSR per GPU-h

Gli esperimenti sono stati eseguiti su cluster HPC, sfruttando nodi con 4 GPU NVIDIA A40 o A100. A titolo di calcolo dei costi di computazione si fa riferimento alla partizione gpu_a40 del cluster HPC@PoliTO.

Ogni job ha richiesto l'allocazione di un intero nodo (4 GPU e 48 core CPU) ed è stato configurato per elaborare il refinement iterativo di 8 siti web. Ogni iterazione del ciclo di refinement è stata organizzata per sfruttare il parallelismo hardware.

Il refinement completo di 10 iterazioni su un job da 8 siti richiede un tempo di esecuzione di circa 15,94 ore, equivalenti a 63,74 GPU-hour, 764,82 CPU-hour e 3,714 kWh di energia consumata. Normalizzando tali valori sul numero di siti, il costo medio ammonta a circa 7,97 GPU-hour e 0,46 kWh per sito, portando a una stima complessiva di circa 805 GPU-hour e 47 kWh per l'elaborazione dell'intero dataset di 101 siti del benchmark. La stima dei costi relativi all'esecuzione della pipeline iterativa è riportata in Tabella 5.3.

Tabella 5.3: Stima dei costi computazionali per l'esecuzione della pipeline iterativa.

Metrica	**Per job (8 siti)**	**Dataset (101 siti)**
Tempo di esecuzione (h)	15,94	—
GPU-hour	63,74	≈ 805
CPU-hour	764,82	≈ 9.656
Energia (kWh)	3,714	≈ 47

Al fine di valutare quantitativamente il beneficio introdotto dal processo di refinement rispetto al suo costo computazionale, è stata definita l'efficienza η:

$$\eta = \frac{\mathrm{TSR}_{\mathrm{Best}} - \mathrm{TSR}_{\mathrm{Base}}}{\text{GPU-hour}_{\mathrm{tot}}} \tag{5.1}$$

dove $\text{GPU-hour}_{\mathrm{tot}}$ rappresenta il costo complessivo del refinement sull'intero dataset.

Come descritto nella sezione precedente, il TSR (corretto per la permissività dei test automatizzati Playwright) passa da un intervallo di $[27{,}52\%, 32{,}05\%]$ per il Base Code a un intervallo di $[47{,}50\%, 55{,}31\%]$ per il Best Code, registrando un guadagno medio di 21,62 punti percentuali (dal 29,79% al 51,41%).

In conclusione, con circa 805 GPU-hour il guadagno medio si attesta a $\eta \approx 0{,}027$ punti percentuali per GPU-hour, equivalenti a circa 37 GPU-hour per ogni punto percentuale di TSR guadagnato. Considerando gli estremi dei range del TSR, nel caso più prudente (TSR iniziale massimo e TSR finale minimo) e in quello più favorevole (TSR iniziale minimo e TSR finale massimo), il guadagno oscilla tra 15,45 e 27,79 punti percentuali, con un valore di efficienza η compreso tra 0,019 e 0,035 punti percentuali per GPU-hour.

5.4 Convergence Curve e identificazione dell'Early Stopping

Per valutare l'efficienza reale della pipeline e stabilire se il numero di iterazioni di refinement, fissato a 10, sia proporzionato o eccessivo, è stata condotta un'analisi delle curve di convergenza. Questa indagine prevede il monitoraggio incrementale delle metriche di Task Success Rate (TSR), Score Costituzionale medio e percentuale di giudizi che hanno ottenuto il punteggio massimo (punteggio pari a 5) per ogni singolo step del loop iterativo, anziché limitarsi all'estrazione del solo Best State.

L'obiettivo è individuare il punto di saturazione di ottimizzazione dell'applicazione della Costituzione: il momento in cui i difetti strutturali e visivi principali sono stati risolti e le revisioni successive hanno iniziato a recuperare solo difetti marginali, assottigliando drasticamente i guadagni prestazionali. Identificare questo

plateau permette di ridurre il costo computazionale, ottimizzando l'impiego delle risorse senza sacrificare in modo sensibile il risultato finale.

L'analisi è stata condotta sul test set di WebGen-Bench [14] che conferma un andamento asintotico classico dei processi di ottimizzazione iterativa. Osservando i risultati ottenuti sui 101 siti del dataset presentati nella Tabella 5.4, emerge una chiara decrescita del guadagno marginale (ΔTSR) ad ogni iterazione successiva.

Il TSR riportato in questa sezione è calcolato come media macro sui siti, cioè come media dei rapporti tra task completati e task totali di ciascuno dei 101 siti, così che ogni sito abbia lo stesso peso indipendentemente dal numero di task. Nelle Tabelle 5.1 e 5.2 il TSR è invece calcolato come rapporto tra task completati e 647 task totali (media micro), che assegna lo stesso peso a ogni task. Le due misure differiscono lievemente ma descrivono lo stesso andamento.

Tabella 5.4: Andamento delle metriche prestazionali per iterazione di Refinement

Iterazione	**AVG TSR**	**Avg Score**	**% giuduzi a 5**	ΔTSR	$\Delta Score$
0 (Setup)	0.3670	26.27	37.03	-	-
1	0.4780	29.19	45.48	0.111	2.92
2	0.5343	29.56	46.68	0.0563	0.37
3	0.5613	30.64	50.51	0.027	1.08
4	0.5723	30.88	53.07	0.011	0.24
5	0.5913	31.28	54.27	0.019	0.4
6	0.6048	31.74	57.20	0.0135	0.46
7	0.6172	32.33	60.36	0.0124	0.59
8	0.6258	32.56	61.37	0.0086	0.23
9	0.6458	33.15	62.33	0.0198	0.59
10	0.6562	33.36	63.26	0.0106	0.21

Il comportamento del modello nelle prime tre iterazioni è decisivo per la risoluzione delle criticità più gravi: la percentuale di giudizi al massimo sulla Costituzione passa rapidamente dal 37.03% al 50.51%. In questa ristretta finestra operativa si concentra la stragrande maggioranza dell'incremento del TSR, che sale da 0.3670 a 0.5613, assorbendo circa il 67% del guadagno totale registrato all'iterazione 10, e dell'incremento dello Score che sale da 26.27 a 30.64, assorbendo circa il 62% del guadagno totale.

La lettura di questi dati riflette la saturazione progressiva dei principi valutativi: man mano che le revisioni sistemano i difetti evidenti, il critico LLM trova sempre meno elementi su cui intervenire e l'impatto sul TSR si appiattisce. Analizzando la derivata prima della curva di convergenza, il "ginocchio" si posiziona nettamente attorno all'iterazione 3. Superata questa soglia il punteggio totale medio e l'aumento del TSR subiscono un forte rallentamento, stabilizzandosi su guadagni frazionali.

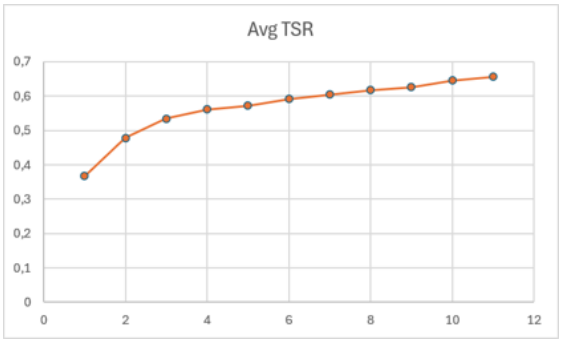


Figura 5.3: Andamento Task Success Rate medio

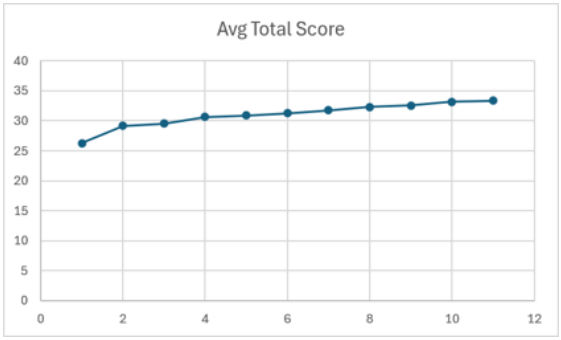


Figura 5.4: Andamento Score Costituzionale medio

Alla luce di questi dati, si può concludere che il limite di 10 iterazioni rappresenta un upper bound utile a fini esplorativi e di ricerca per estrarre il massimo potenziale teorico della pipeline. Tuttavia, in uno scenario di applicazione reale, fissare un early stopping a 3 iterazioni rappresenta il trade-off ottimale. Fermare il loop di autocritica all'iterazione 3 garantirebbe infatti un risparmio computazionale fino al 70%, mantenendo un TSR altamente competitivo (0.5613 contro lo 0.6562 finale). Questo certifica che il nucleo del lavoro logico e strutturale del modello si concentra nelle primissime fasi di refinement, rendendo le iterazioni finali spesso ridondanti a fronte dei costi di computazione richiesti.

Capitolo 6

Discussione e Conclusioni

Questo capitolo conclude la tesi riassumendo i principali contributi proposti nel progetto di Consitutional Web Agent e i relativi miglioramenti dovuti alla sua applicazione per la generazione autonoma e miglioramento di codice frontend per siti web. In aggiunta vengono presentate le limitazioni del corrente approccio e le possibili direzioni per futuri lavori.

6.1 Sintesi dei contributi e dell'efficacia del lavoro

In questo lavoro di tesi è stato sviluppato e presentato un Constitutional Web Agent, un framework innovativo volto a migliorare la generazione autonoma di codice frontend per siti web, sfruttando la cooperazione tra Large Language Model e Visual-Language Model. La soluzione proposta definisce un set di principi, organizzati sotto forma di una Costituzione, e li applica all'interno di una pipeline di Refinement iterativo. L'obiettivo è il miglioramento qualitativo e di usabilità delle interfacce web, ottenuto attraverso un processo in cui il codice viene criticato e aggiornato sulla base di regole consolidate della Human-Computer Interaction (HCI), facendo si che le parti di codice legate alle funzionalità logiche del sito rimangano inalterate. Nello specifico, i contributi introdotti in questo studio si dividono in:

- **Pipeline generativa e di Refinement iterativo**: È stata progettata un'architettura in grado di orchestrare LLM e VLM per l'ottimizzazione e la valutazione iterativa del codice web. La pipeline sfrutta la capacità degli LLM di generare codice e di operare un'autocritica basata su principi HCI. Parallelamente, impiega VLM come agenti autonomi capaci di simulare il comportamento umano sulle interfacce, testando il funzionamento e la navigazione nel DOM. Questo duplice approccio permette di validare l'output

e raccogliere metriche e statistiche affidabili legate al funzionamento del codice e alla sua conformità alla linee guida della Human-Computer Interaction.

- **HCI Constitution**: È stata definita una Costituzione composta da 8 principi fondamentali derivati dalle 10 Euristiche di Nielsen, dalle linee guida sull'accessibilità WCAG e dalla Legge di Fitts, che rappresentano lo standard per lo sviluppo web. Tali principi fungono da direttive cardine per la valutazione e il miglioramento delle User Interface (UI) e della User Experience (UX). L'applicazione di questa Costituzione garantisce che il miglioramento guidato e misurato dai modelli avvenga con vincoli precisi, senza corrompere la logica sottostante. Questa metodologia riduce al minimo l'intervento umano, delegando il processo di iterazione, critica e correzione ad un sistema automatizzato in grado di generare codice frontend più performante in termini di usabilità.

L'applicazione congiunta della pipeline e della Costituzione consente di generare e ottenere feedback relativi all'usabilità e accessibilità dei siti web, la cui applicazione permette di revisionare il codice e fornire una versione migliorata e più performante. Tramite le analisi delle metriche di valutazione, quali lo Score Costituzionale e il Task Success Rate (TSR), si è dimostrato come l'applicazione del framework ha portato ad un notevole incremento del TSR sui siti web analizzati tramite il benchmark WebGen-Bench [14], con risultati indicativamente in linea con quelli riportati da WebGen-Agent [4], pur con un meccanismo di feedback basato sulla sola critica costituzionale e senza analisi visiva tramite VLM. L'analisi delle curve di convergenza mostra inoltre che circa i due terzi del guadagno si concentrano nelle prime tre iterazioni, suggerendo un compromesso favorevole tra costo computazionale e prestazioni con un early stopping precoce.

In conclusione, l'utilizzo di una pipeline di Auto-Refinement iterativo, che sfrutta le capacità degli LLM per la generazione, l'auto-critica e la revisione del codice sotto la guida di una Costituzione fondata sull'HCI, si mostra una soluzione robusta per l'ottenimento di codice frontend significativamente più performante. Questo approccio abbatte gran parte delle criticità legate all'usabilità e alla navigabilità dei siti web generati automaticamente.

Inoltre, richiede un intervento umano limitatissimo, sufficiente a indirizzare le metriche di valutazioni verso un maggiore allineamento con le reali esigenze degli utenti, superando così i limiti di un'ottimizzazione guidata unicamente da logiche puramente algoritmiche o dalla percezione e capacità di critica limitate dei modelli.

6.2 Limitazioni del lavoro

Nonostante i risultati promettenti ottenuti dall'applicazione della HCI Constitution, il lavoro presentato espone alcune limitazioni intrinseche, derivanti sia dalle scelte architetturali sia dalle attuali restrizioni tecnologiche.

In primo luogo, il sistema di Task Evaluation adottato si affida esclusivamente all'analisi strutturale del codice tramite l'interrogazione del DOM con test Playwright autogenerati dal Large Language Model. Questa scelta è stata necessaria in quanto la valutazione visiva del design diretta da parte del modello Holo2-8B si è dimostrata inefficace, arrivando a validare come sufficientemente funzionanti anche pagine web del tutto bianche o destrutturate.

Sebbene l'approccio basato sul DOM si sia dimostrato solido per la verifica logica e funzionale, esso risulta di fatto cieco rispetto al reale rendering visivo dell'interfaccia. Di conseguenza, il sistema non è in grado di rilevare anomalie grafiche, come la sovrapposizione di elementi a schermo, contrasti cromatici inadeguati o malfunzionamenti legati al responsive design.

Inoltre, possono verificarsi situazioni di fallimento di rendering totale in cui il Visual-Language Model si ritrova ad analizzare un'interfaccia vuota, sebbene gli elementi logici risultino formalmente presenti e corretti all'interno del DOM. Ciò si traduce in un inutile consumo di risorse computazionali durante le varie iterazioni del ciclo di refinement, poiché avviene ugualmente l'elaborazione dell'immagine durante i cicli di Task Evaluation.

Queste criticità non possono essere intercettate dalla sola analisi del codice sorgente, richiedendo l'integrazione di capacità multimodali più avanzate.

A questo si unisce la sfida della traduzione dei principi di Human-Computer Interaction in direttive formali. La codifica di concetti complessi e talvolta soggettivi (come la percezione del carico cognitivo, l'estetica o il feedback visivo) in regole testuali all'interno della Costituzione comporta inevitabilmente un processo di riduzione dei concetti. Alcune sfumature dell'interazione umana sfuggono alla rigida categorizzazione dei prompt, limitando parzialmente la capacità del modello di valutare l'interfaccia con la medesima sensibilità di un esperto di usabilità umano.

Dal punto di vista architetturale e prestazionale, la pipeline di self-critique e refinement iterativo introduce un notevole overhead. Il ciclo di correzione richiede, senza considerare l'Early Stopping, multiple chiamate al Large Language Model per la generazione, la critica e la revisione del codice. Anche sfruttando infrastrutture High-Performance Computing (HPC), questo processo comporta elevati costi computazionali e un'alta latenza complessiva. Di conseguenza, il framework nella sua veste attuale risulta poco adatto per scenari di produzione che richiedono la generazione di interfacce one-shot o in real-time.

Infine, un'ulteriore limitazione risiede nella scalabilità dell'approccio rispetto alla complessità delle moderne architetture frontend. La pipeline attuale gestisce in modo eccellente la generazione e l'ottimizzazione di interfacce standard e componenti isolati. Tuttavia, l'estensione di questo approccio iterativo a framework complessi che fanno uso di gestori dello stato globale e logiche di reattività incrociata rappresenta una sfida aperta.

6.3 Sviluppi futuri

A partire dalla progettazione del Constitutional Web Agent e dai risultati ottenuti, emergono diverse direzioni promettenti per i lavori futuri.

Il primo obiettivo consiste nell'eseguire il fine-tuning dei modelli generativi per ottenere agenti capaci di produrre, in modalità one-shot, codice front-end di alta qualità. Considerando l'architettura dell'attuale pipeline, dalla fase di setup a quella terminale di Refinement, risulta particolarmente indicato l'impiego della Direct Preference Optimization (DPO) unita a tecniche Parameter-Efficient come LoRA. In questo scenario, il codice della baseline fungerebbe da output scartato, mentre il codice della soluzione migliore rappresenterebbe la risposta preferita su cui allineare il modello. Tale approccio permetterebbe al sistema di interiorizzare i principi di usabilità, riducendo drasticamente i costi computazionali e i tempi di inferenza grazie alla potenziale eliminazione delle fasi iterative di Self-Critique e di Task Evaluation.

In secondo luogo, la HCI Constitution potrebbe essere ulteriormente espansa per includere nuovi principi, andando a mitigare ulteriori criticità legate all'usabilità e all'accessibilità del codice generato dall'Intelligenza Artificiale. Un'evoluzione interessante consisterebbe nell'introdurre la possibilità di definire e inserire nuove direttive a run-time, permettendo alla fase di critica di adattarsi dinamicamente agli specifici requisiti del dominio applicativo.

Un'ulteriore prospettiva di sviluppo riguarda l'estensione della pipeline alla generazione di interfacce basate su componenti più complessi, come ad esempio React. Questo passo richiederebbe la definizione di nuovi principi dedicati all'interno della Costituzione, progettati per valutare e governare direttive specifiche di accessibilità e usabilità che derivano dall'impiego di framework moderni e architetture modulari.

Queste direzioni di ricerca evidenziano un potenziale di miglioramento ancora ampiamente inesplorato nel campo della generazione e del testing autonomo di siti web. I risultati di questo lavoro confermano, infatti, che un codice perfettamente funzionante non si traduce automaticamente in un prodotto finale ottimale; al contrario, il controllo rigoroso della qualità dell'esperienza utente, garantito dai requisiti di usabilità e accessibilità delle User Interface (UI), rappresenta un fattore imprescindibile.

Bibliografia

[1] Pengcheng Yin e Graham Neubig. «A Syntactic Neural Model for General-Purpose Code Generation». In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 440–450. DOI: 10.18653/v1/P17-1041 (cit. alle pp. 1, 6, 9, 13).

[2] Lei Wang et al. «A survey on large language model based autonomous agents». In: *Frontiers of Computer Science* 18.6 (2024), p. 186345. DOI: 10.1007/s11704-024-40231-1 (cit. a p. 1).

[3] Grégoire Mialon et al. «Augmented Language Models: a Survey». In: *arXiv preprint arXiv:2302.07842* (feb. 2023). arXiv: 2302.07842 [cs.CL] (cit. a p. 1).

[4] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan e Ofir Press. «SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering». In: *arXiv preprint arXiv:2405.15793* (2024). arXiv: 2405.15793 [cs.SE] (cit. alle pp. 1, 2, 9, 13, 51, 57).

[5] Shuyan Zhou et al. «WebArena: A Realistic Web Environment for Building Autonomous Agents». In: *arXiv preprint arXiv:2307.13854* (2023). arXiv: 2307.13854 [cs.AI] (cit. alle pp. 1, 7, 13).

[6] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge et al. «OpenHands: An Open Platform for AI Software Developers as Generalist Agents». In: *arXiv preprint arXiv:2407.16741* (2024). arXiv: 2407.16741 [cs.SE] (cit. alle pp. 1, 13).

[7] Marc Hassenzahl e Noam Tractinsky. «User experience—a research agenda». In: *Behaviour & Information Technology* 25.2 (2006), pp. 91–97. DOI: 10.1080/01449290500330331 (cit. a p. 2).

[8] Chenchen Zhang et al. «ArtifactsBench: Bridging the Visual-Interactive Gap in LLM Code Generation Evaluation». In: *arXiv preprint arXiv:2507.04952* (2025). arXiv: 2507.04952 [cs.SE] (cit. alle pp. 2, 13).

[9] Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan e Hongsheng Li. «WebGen-Agent: Enhancing Interactive Website Generation with Multi-Level Feedback and Step-Level Reinforcement Learning». In: *arXiv preprint arXiv:2509.22644* (set. 2025). arXiv: 2509.22644 [cs.CL] (cit. alle pp. 2, 4, 5, 9, 13, 16, 18).

[10] Florian Bordes, Richard Yuanzhe Pang, Anurag Ajay, Alexander C. Li, Adrien Bardes, Suzanne Petryk, Oscar Mañas, Zhiqiu Lin et al. «An Introduction to Vision-Language Modeling». In: *arXiv preprint arXiv:2405.17247* (mag. 2024). arXiv: 2405.17247 [cs.CV] (cit. alle pp. 2, 5).

[11] Wenyi Hong et al. «CogAgent: A Visual Language Model for GUI Agents». In: *arXiv preprint arXiv:2312.08914* (2023). arXiv: 2312.08914 [cs.CV] (cit. alle pp. 2, 5).

[12] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret et al. «RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback». In: *arXiv preprint arXiv:2309.00267* (2023). arXiv: 2309.00267 [cs.CL] (cit. alle pp. 2, 5, 8).

[13] Yuntao Bai et al. «Constitutional AI: Harmlessness from AI Feedback». In: *arXiv preprint arXiv:2212.08073* (dic. 2022). DOI: 10.48550/arXiv.2212.08073. arXiv: 2212.08073 [cs.CL] (cit. alle pp. 2–5, 8–10, 13, 18, 22).

[14] Zimu Lu, Yunqiao Yang, Houxing Ren, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan e Hongsheng Li. «WebGen-Bench: Evaluating LLMs on Generating Interactive and Functional Websites from Scratch». In: *arXiv preprint arXiv:2505.03733* (mag. 2025). DOI: 10.48550/arXiv.2505.03733. arXiv: 2505.03733 [cs.CL] (cit. alle pp. 4, 9, 14, 35, 49, 54, 57).

[15] Lovable Labs. *Lovable: The Last Piece of Software*. [Online; accessed 16-Sept-2026]. 2024. URL: https://lovable.dev (cit. alle pp. 5, 6).

[16] StackBlitz Labs. *bolt.diy*. GitHub repository. [Online; accessed Jan-2026]. 2024. URL: https://github.com/stackblitz-labs/bolt.diy (cit. alle pp. 6, 16).

[17] Yevgen Borodin, Jeffrey P. Bigham, Glenn Dausch e I. V. Ramakrishnan. «More than meets the eye: a survey of screen-reader browsing strategies». In: *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*. W4A '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 1–10. DOI: 10.1145/1805986.1806005 (cit. a p. 6).

[18] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan e Dong Yu. «WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models». In: *arXiv preprint arXiv:2401.13919* (gen. 2024). DOI: 10.48550/arXiv.2401.13919. arXiv: 2401.13919 [cs.CL] (cit. a p. 7).

[19] Yuxuan Lu et al. «UXAgent: An LLM Agent-Based Usability Testing Framework for Web Design». In: *arXiv preprint arXiv:2502.12561* (2025). DOI: 10.48550/arXiv.2502.12561. arXiv: 2502.12561 [cs.HC] (cit. alle pp. 7, 9).

[20] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei e Paul F. Christiano. «Learning to summarize with human feedback». In: *arXiv preprint arXiv:2009.01325* (2020). arXiv: 2009.01325 [cs.CL] (cit. a p. 8).

[21] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin et al. «Training Language Models to Follow Instructions with Human Feedback». In: *arXiv preprint arXiv:2203.02155* (2022). arXiv: 2203.02155 [cs.CL] (cit. a p. 8).

[22] Qwen. *Qwen3.5-27B*. Hugging Face repository. [Online; accessed Mar-2026]. 2026. URL: https://huggingface.co/Qwen/Qwen3.5-27B (cit. alle pp. 12, 39, 50).

[23] H Company. *Holo2-8B*. Hugging Face repository. [Online; accessed Feb-2026]. 2024. URL: https://huggingface.co/Hcompany/Holo2-8B (cit. alle pp. 12, 50).

[24] Google DeepMind. *Gemini 3.1 Pro*. [Online; accessed Sept-2026]. 2026. URL: https://deepmind.google/technologies/gemini/ (cit. alle pp. 14, 39).

[25] The Chromium Projects. *Chromium: An open-source browser project*. Google e The Chromium Authors, 2008. URL: https://www.chromium.org/chromium-projects/ (visitato il giorno 03/2026) (cit. alle pp. 16, 17).

[26] Microsoft. *Playwright: Fast and reliable end-to-end testing for modern web apps*. https://playwright.dev/. [Online; accessed Mar-2026]. Repository: https://github.com/microsoft/playwright. 2020 (cit. alle pp. 16, 17).

[27] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. [Online; updated 2024, accessed Feb-2026]. 1994. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (cit. a p. 23).

[28] W3C. *Web Content Accessibility Guidelines (WCAG) 2.1*. [Online; accessed Feb-2026]. World Wide Web Consortium (W3C), 2018. URL: https://www.w3.org/TR/WCAG21/ (cit. a p. 23).

[29] Paul M. Fitts. «The information capacity of the human motor system in controlling the amplitude of movement». In: *Journal of Experimental Psychology* 47.6 (1954), pp. 381–391. DOI: 10.1037/h0055392 (cit. a p. 23).

[30] J. Richard Landis e Gary G. Koch. «The Measurement of Observer Agreement for Categorical Data». In: *Biometrics* 33.1 (1977), pp. 159–174. DOI: 10.2307/2529310 (cit. a p. 36).

[31] Anthropic. *Claude Opus 4.7*. [Online; accessed Sept-2026]. 2026. URL: https://www.anthropic.com/news/claude-opus-4-7 (cit. a p. 39).

[32] OpenAI. «GPT-4o System Card». In: *arXiv preprint arXiv:2410.21276* (ott. 2024). DOI: 10.48550/arXiv.2410.21276. arXiv: 2410.21276 [cs.CL] (cit. a p. 39).