Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica

A.a. 2024/2025

Sessione di laurea Ottobre 2025

Generazione automatica di prototipi web da sketch personalizzati

Relatori:

Luigi De Russis

Tommaso Calò

Candidato:

Simone Santarcangelo

Ringraziamenti

A tutti coloro che mi sono stati vicini durante questi anni. Soprattutto ai miei genitori, che sono stati i miei primi sostenitori.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Con il passare degli anni, e con l'integrazione sempre più pervasiva delle tecnologie digitali nella vita quotidiana, ognuno di noi ha avuto modo di interagire con sistemi diversi, dai dispositivi mobili ai siti web. L'accesso a queste tecnologie avviene attraverso le interfacce, ossia quell'insieme di elementi visivi e interattivi che consentono all'utente di comunicare con il sistema e di sfruttarne le funzionalità. La progettazione di tali interfacce non è mai casuale: è il risultato di un processo di studio che mira a renderle intuitive, accessibili ed efficaci. Questo compito è affidato ai web designer, figure specializzate nel tradurre bisogni e aspettative degli utenti in soluzioni visive e funzionali. A partire dalla prima interfaccia grafica moderna, sviluppata nel 1973 presso lo Xerox Palo Alto Research Center [1], il design digitale si è progressivamente orientato verso un approccio user-centered, privilegiando l'usabilità rispetto alla sola estetica. L'evoluzione delle interfacce è stata spinta, da un lato, dalla crescita esponenziale del numero di persone che si avvicinavano a queste tecnologie e, dall'altro, dal rapido avanzamento degli strumenti di progettazione e sviluppo. In questo contesto nuove applicazioni e metodologie stanno emergendo per supportare i designer: strumenti capaci di velocizzare la creazione di layout, ridurre le barriere tecniche e persino adattarsi allo stile individuale di ciascun progettista, aprendo la strada ad un processo di design sempre più personalizzato e collaborativo tra uomo e macchina.

1.1 Prototipazione UI

Nel contesto dello sviluppo web, il processo di progettazione che precede la realizzazione delle interfacce grafiche prende il nome di prototipazione UI, in cui UI sta per *User Interface*, cioè interfaccia utente. La prototipazione dell'interfaccia utente rappresenta una fase fondamentale nello sviluppo di prodotti digitali, poiché consente ai designer di creare versioni preliminari dell'interfaccia, dei veri e propri

prototipi, che servono come base di partenza per il processo di sviluppo successivo. È il primo step, che ha lo scopo di simulare le funzioni che avrà l'applicativo finale e permette di avere un quadro di come l'utente andrà a interagire con esso. Analogamente a quanto accade in altri ambiti progettuali, anche nel settore web il prototipo costituisce la prima concretizzazione dell'idea iniziale, fungendo da ponte tra la fase concettuale e quella di implementazione. I prototipi delle interfacce possono variare da wireframe a bassa fedeltà, che definiscono soltanto la struttura e la disposizione degli elementi, fino a modelli interattivi ad alta fedeltà, in grado di simulare l'esperienza d'uso finale in maniera realistica. Questa pratica riveste un ruolo chiave nella valutazione dell'usabilità e dell'efficacia del design, poiché consente di individuare potenziali criticità prima dell'implementazione definitiva. Attraverso la prototipazione, sviluppatori e designer possono osservare il comportamento degli utenti, raccogliere feedback empirici e apportare modifiche mirate, migliorando così progressivamente l'esperienza utente (UX) e garantendo un prodotto finale più coerente con le esigenze del target di riferimento.

1.1.1 Prototipi in bassa fedeltà

Le fasi iniziali del processo di prototipazione dell'interfaccia utente partono generalmente dalla realizzazione di prototipi a bassa fedeltà. Si tratta di rappresentazioni visive estremamente schematiche dell'applicazione o della pagina web in fase di progettazione, il cui obiettivo principale è definire la struttura logica e funzionale dell'interfaccia piuttosto che il suo aspetto estetico finale. Questi prototipi si concentrano sulla disposizione degli elementi, sulla loro gerarchia visiva e sulle funzionalità chiave che l'interfaccia dovrà offrire. Vengono tipicamente realizzati utilizzando forme grafiche semplici come riquadri, linee e testi segnaposto che consentono di delineare rapidamente il layout, i punti di interazione e la navigazione tra le varie sezioni del sistema. L'attenzione, in questa fase, non è rivolta al design visivo, bensì alla struttura e all'usabilità. Per tale motivo è comune che i prototipi in bassa fedeltà vengano realizzati manualmente, ad esempio tramite schizzi su carta, oppure con l'ausilio di software di disegno digitale generico, senza l'utilizzo di strumenti di prototipazione avanzata. Lo sketching rappresenta infatti il metodo più immediato e semplice per valutare l'efficacia del layout di un'interfaccia poiché consente di ottenere una visione d'insieme delle relazioni spaziali e delle funzioni principali in tempi molto ridotti. Sia il disegno in sè sia eventuali modifiche, correzioni o ridisegni possono essere effettuati in modo rapido e con un dispendio minimo di risorse, evitando di compromettere fasi successive del processo di sviluppo, più complesse e costose da correggere. Iniziare il processo di prototipazione realizzando degli sketch è quindi una pratica molto utilizzata, nonostante possa sembrare obsoleta e anacronistica.

1.1.2 L'evoluzione degli strumenti di prototipazione web

La fase successiva è quella di evolvere lo sketch o il generico prototipo a bassa fedeltà in un prototipo a alta fedeltà. Questi prototipi si distinguono per il loro elevato livello di dettaglio grafico e interattività, includendo elementi visivi completi, transizioni animate e interazioni realistiche, per cercare di simulare con precisione come il prodotto finale apparirà e funzionerà. Grazie a questo grado di realismo, tali prototipi risultano particolarmente efficaci per valutare le reazioni degli utenti a elementi specifici dell'interfaccia e per raccogliere feedback mirati sull'esperienza d'uso. La loro realizzazione richiede però l'impiego di strumenti avanzati di progettazione interattiva. Inizialmente, anche questa fase di prototipazione avveniva attraverso un processo manuale e soprattutto statico, basato su disegni a mano o tramite semplici mockup digitali senza nessuna forma di interattività. Ad esempio uno dei software più usati dai designer per dare vita alle loro creazioni era Adobe Photoshop. Tuttavia, nonostante l'indubbia qualità dell'applicazione nel uo campo di utilizzo, risulta chiaro che questa non è stata creata per progettare un sito web e risulta inadeguato per questo compito. [2] Attualmente, i migliori strumenti di prototipazione seguono dei criteri per poter offrire in primis, a loro volta, un'esperienza utente semplice e intuitiva e che permettano di ottenere risultati efficaci e in modo rapido. Alcune delle funzioni di cui un prototype tool deve disporre per rientrare in questi criteri sono sicuramente:

- Disporre a sua volta di un'interfaccia utente intuitiva.
- Permettere la collaborazione tra i membri del team.
- Integrare funzionalità che consentano di convertire in modo semplice contenuti statici in contenunti interattivi.
- supportare l'integrazione con altre piattaforme, quali possono essere applicazioni per la condivisione di risorse o programmi per l'editing.
- Essere compatibile con il maggior numero di dispositivi possibile.

Con l'avvento di strumenti dedicati come Adobe XD, InVision, Framer, Marvel e Figma [3], la prototipazione è diventata più collaborativa e dinamica, permettendo la creazione di prototipi interattivi senza necessità di scrivere codice, ma in grado di simulare l'esperienza finale dell'utente.

1.1.3 Il ruolo dell'Intelligenza Artificiale nella prototipazione UI

Come in molti dei contesti tecnologici e digitali, anche in quello del web design e conseguentemente della prototipazione web, l'uso dell'IA sta pian piano prendendo

sempre più spazio negli ultimi anni. Oggi progettare un sito web non significa più necessariamente partire da un foglio bianco e costruire tutto manualmente. Grazie al supporto dell'IA esistono strumenti in grado di rivoluzionare il concetto di "design assistito", che esiste da anni, proponendo attivamente layout coerenti o generando testi e contenuti in generale [4]. Questo porta a una progettazione accelerata, personalizzata e intelligente, con strumenti che supportano designer e sviluppatori in ogni fase, dalla generazione del layout ai test, fino all'ottimizzazione continua delle esperienze digitali. Questo strumento si affianca al designer non solo per fornire un supporto alleggerendo il carico di lavoro attraverso l'automazione di task ripetitivi, ma dando assistenza intelligente durante il design. Sono in grado di suggerire alternative di layout, proporre combinazioni di colori o tipografie adatte al contesto o, anche, inserimento degli elementi in determinate posizioni al fine di massimizzare l'engagement dell'utente. Inoltre possono analizzare e simulare i possibili pattern di utilizzo dell'app per identificare punti di attrito e per prevedere le navigazioni più probabili [5], suggerendo addirittura microinterazioni e flussi di navigazione sulla base dei dati comportamentali degli utenti.

1.2 Obiettivi della di tesi

Abbiamo analizzato il quadro generale nel panorama dei sistemi atti alla prototipazione di pagine web, osservando che, attualmente, ne esistono alcuni che danno all'utente la possibilità di integrare in parte il lavoro dell'intelligenza artificiale nel flusso della prototipazione. Ma, attualmente, l'utilizzo dell'IA, grazie alla capacità di generare codice funzionante, si può spingere anche oltre. Non esiste ancora un tool che, sfruttando l'IA, permetta di passare direttamente dalla primissima fase della prototipazione - cioè un disegno - a un prototipo finale.

L'obiettivo di questa tesi è stato quello di progettare e creare uno strumento che permetta di trasformare uno sketch di un'interfaccia disegnata a mano in una vera interfaccia funzionante scritta in linguaggio HTML, sfruttando un modello di intelligenza artificiale generativa che produca il codice corrispondente allo sketch. Il sistema proposto è un'applicazione web che deve essere in grado di riconoscere lo stile di disegno dell'utente al fine di generare un'interfaccia che rispecchi l'intenzione del disegno dell'utente. È stato quindi realizzato un sistema che, da una parte permette di disegnare a mano libera senza limitazioni nel modo più semplice possibile - in linea con le caratteristiche di un software per la prototipazione a basso livello - mentre dall'altra interroga un modello di intelligenza artificiale personalizzato che genera il codice. Sono stati utilizzati i servizi Cloud AWS sia per il lato server backend, sia in parte per il frontend.

1.3 Struttura della tesi

La tesi si articola come segue:

Capitolo 2: Fondamenti teorici. Verranno analizzate le tecnologie di intelligenza artificiale nell'ambito della generazione automatica di codice.

Capitolo 3: Progettazione. Verrà definito il sistema che si vuole realizzare, dai requisiti che deve soddisfare fino a tracciarne uno schema di funzionamento iniziale

Capitolo 4: Implementazione. Si descriverà il processo di sviluppo dell'applicazione, spiegando come vengono utilizzati i servizi AWS lato server, l'implementazione del codice del frontend e le comunicazioni tra questi due componenti.

Capitolo 5: Sperimentazione e risultati. Verrà descritta la fase di test dell'applicazione, e analizzati i risultati ottenuti.

Capitolo 6: Conclusioni: Verranno commentati i risultati del punto precedente sia nell'ottica oggettiva del soddisfacimento dei requisiti iniziali, sia in quella crtica di impatto con l'utente finale, proponendo possibili miglioramenti.

Capitolo 2

Fondamenti teorici

In questo capitolo si andranno ad analizzare quelli che sono i presupposti del progetto. Prima di tutto verrà commentato il concetto di intelligenza artificiale generativa applicata al task della generazione di codice. Seguirà una sezione in cui verranno descritti i metodi di finetuning, nello specifico quelli leggeri.

2.1 Intelligenza Artificiale e generazione di codice

L'avanzamento esponenziale nell'ambito dell'elaborazione del linguaggio naturale (NLP) e dell'apprendimento automatico (Machine Learning) ha catalizzato una profonda trasformazione in numerosi settori. Tra i campi che hanno beneficiato maggiormente di questi progressi vi è quello della generazione automatica di codice, una disciplina emergente che mira a supportare o automatizzare lo sviluppo software attraverso l'impiego di modelli di Intelligenza Artificiale (IA). Nel contesto della presente analisi, l'attenzione è focalizzata in particolare sul codice HTML (HyperText Markup Language). Questo linguaggio rappresenta il fondamento stesso del web, costituendo il punto di partenza essenziale per la creazione strutturale di interfacce, pagine e applicazioni. I modelli di IA che operano in questo settore sono progettati per tradurre specifiche e requisiti in output di codice funzionante. Tali sistemi non si limitano a produrre frammenti di codice, ma ambiscono a generare strutture complete, ottimizzate e conformi agli standard di accessibilità e best practice del web. L'introduzione della generazione automatica del codice apporta benefici sostanziali e trasversali a diverse categorie di utenti. Per la categoria dei programmatori, l'impiego di strumenti di generazione automatica di codice apre nuove e significative prospettive:

- Qualità del codice: l'IA può contribuire a produrre codice più pulito, coerente e meno soggetto a errori.

- Ottimizzazione delle strutture: i modelli avanzati possono suggerire o implementare strutture di codice più efficienti e scalabili.
- Aderenza agli standard del web: l'automazione garantisce una maggiore conformità agli standard e alle specifiche tecniche più recenti.

Il vantaggio più rivoluzionario si manifesta per gli individui con limitate capacità tecniche. I modelli di IA più avanzati hanno raggiunto la capacità di:

- Interpretare richieste in linguaggio naturale: gli utenti possono esprimere le loro esigenze di design o funzionalità tramite semplici descrizioni testuali.
- Contestualizzare le esigenze: l'IA è in grado di comprendere il contesto della richiesta e le specifiche esigenze dell'utente.
- Proporre soluzioni funzionali: il risultato è la produzione di codice funzionante, rendendo possibile la creazione di contenuti web senza che l'utilizzatore finale debba possedere nozioni di programmazione avanzate.

Questa democratizzazione dello sviluppo web attraverso l'Intelligenza Artificiale rappresenta un cambiamento di paradigma che ha il potenziale per accelerare drasticamente i processi creativi e produttivi nel panorama digitale.

2.1.1 Modelli generativi per il design e lo sviluppo web

L'intervento dell'Intelligenza Artificiale nel campo dello sviluppo web si concretizza attualmente attraverso l'impiego di modelli generativi capaci di produrre codice. Vengono utilizzati per debuggare, correggere codice o, in alcuni casi, generarlo da zero a partire da un prompt scritto o da pezzi di codice esistente. L'utilizzo delle IA nell'ambito dello sviluppo web non differisce dunque dall'utilizzo che se ne fa normalmente nel campo della programmazione generica, con le IA che nella maggior parte dei casi si limitano a generare pezzi di codice o a spiegarne il funzionamento.

Tra gli esempi più noti di architetture che eccellono in queste capacità figurano GPT-4.5 Turbo, Gemini 2.5 Pro di Google, Claude 4 di Anthropic per i modelli proprietari, mentre per quelli open source ci sono LLaMA 4 e Mistral Medium 3 [6]. È però molto importante precisare che alcuni modelli non si limitano solo a questo. Esistono modelli multimodali che sono in grado di generare intere pagine web personalizzate, complete di codice CSS, sia a partire da prompt testuali, sia da input visivi. Questo significa che un'immagine di una pagina web, come uno screenshot, viene convertita nel codice che la implementa.

Ovviamente è importante capire con che grado di fedeltà la pagina viene riprodotta: ciò è direttamente proporzionale alla qualità del codice generato e alla capacità del modello di comprendere la struttura della pagina, le relazioni tra

elementi contenuti al suo interno e il layout in generale. La valutazione delle performance di tali modelli è essenziale per capirne la reale portata e la reale usabilità. Ad esempio, nello studio *Design2code* [7] è stato condotto un confronto tra la qualità del codice generato da diversi modelli basato sulla somiglianza visiva tra la pagina web generata e una pagina di riferimento. La valutazione è stata affidata ad annotatori umani, i quali dovevano stabilire se la pagina prodotta dal modello in esame fosse più o meno simile alla pagina di riferimento rispetto a quella generata da un modello scelto come baseline (specificatamente, Gemini Pro Vision con Direct Prompting). I modelli sottoposti a verifica includevano GPT-4V, Gemini, WebSight VLM-8B e Design2Code-18B, testati con diverse strategie di prompting. I risultati di questo studio hanno evidenziato prestazioni notevoli, in particolare per GPT-4V: in ben il 64% dei casi gli annotatori hanno giudicato le versioni generate da GPT-4V migliori dei siti reali in termini di design. Questo livello di performance suggerisce chiaramente che la generazione automatica di pagine web non è più una prospettiva futura, ma che è già possibile adesso creare interfacce utente e layout web senza passare dalla programmazione manuale.

2.1.2 Approcci esistenti per la trasformazione da immagini a codice

La trasformazione di un design visivo in codice sorgente è stata affrontata con una varietà di approcci evolutisi nel tempo. I metodi, inizialmente, non si basavano sul deep learning, ma impiegavano la computer vision classica, utilizzando tecniche geometriche e regole definite manualmente per l'analisi dei wireframe. Questi approcci ricorrevano a strumenti come l'OCR (Optical Character Recognition) per identificare elementi strutturali, blocchi di testo, immagini, bottoni e applicavano regole o template predefiniti per la mappatura verso i componenti HTML/CSS. Uno studio del 2019 [8] ha confrontato l'efficacia della computer vision classica con la segmentazione semantica basata su reti neurali per il riconoscimento di elementi di un'interfaccia web, in particolare partendo da uno sketch disegnato a mano. Sebbene il metodo basato sulle reti neurali si sia dimostrato superiore, nessuno dei due approcci ha raggiunto un livello di accuratezza sufficiente nel task di conversione da uno sketch a codice. L'approccio neurale richiedeva, infatti, una buona qualità del disegno e funzionava meglio con elementi standardizzati e layout semplici, limitandone l'utilizzo in produzione. Tra i primi studi che hanno applicato le reti neurali in modo più sistematico, troviamo *pix2code* [9], che ha tentato un approccio end-to-end. Questo studio ha utilizzato un metodo che prevedeva un encoder di immagini, tipicamente una CNN (Convolutional Neural Network), accoppiato con un decoder basato su uno stack di Long-Short Term Memory (LSTM), per generare codice sorgente (HTML/CSS, Android XML, iOS Storyboard) direttamente da uno screenshot di interfaccia grafica, eliminando la

necessità di progettare manualmente i descrittori. Tuttavia, i limiti di questo primo approccio neurale risiedevano principalmente nella scarsa qualità del codice prodotto, dovuta alla dimensione limitata del modello. Gli approcci più moderni si avvalgono invece di grandi modelli multimodali, come i già citati GPT-4V o Gemini, integrandoli con tecniche di fine-tuning su dataset di coppia screenshot-codice per generare file HTML/CSS. Questi modelli sono supportati da strategie di prompting intelligenti, tra cui direct prompting, text-augmented prompting e self-revision. È importante precisare che questi due studi hanno un'utilità esclusivamente di ricerca sulle potenzialità dei moderni modelli di IA nel riconoscere da un'immagine la struttura di un'interfaccia, ma per un utilizzo pratico ha poco senso replicare il codice di una pagina web di cui in partenza si ha già il codice. Invece, un concetto recente e particolarmente innovativo, che già trova applicazioni pratiche, è quello del blending concettuale interattivo introdotto dal sistema *Misty*[10]. Molti software di design grafico permettono di aggiungere esempi preimpostati nel progetto in corso. Misty, tramite il blending concettuale interattivo, fa la stessa cosa, ma non a livello grafico, bensì a livello di codice: consente di unire parti di interfacce di esempio con il codice dell'interfaccia che si sta sviluppando, preservando sia il contenuto originale che lo stile del riferimento. Questo permette, ad esempio, di inserire nel codice dei componenti presi da una lista di esempio, trascinandoli direttamente dagli esempi all'interfaccia grafica, o di applicare lo stile o il layout di un'intera UI di esempio o di una sua sezione specifica al progetto corrente, generando il codice corrispondente e integrandolo nel codice di base dell'interfaccia iniziale. Alla fine del processo, si avrà il codice funzionante dell'interfaccia modificata, il tutto ottenuto usando un nuovo modo esclusivamente grafico di modifica. Questo è un sistema che può trovare già un'applicazione effettiva nel flusso di creazione di una pagina web.

2.1.3 Limiti e potenzialità di tali approcci

L'evoluzione dei metodi Image-to-Code, da quelli basati sulla computer vision classica ai moderni modelli multimodali, ha portato a delle potenzialità funzionali innegabili. Essi aprono la strada a una prototipazione rapida e a una significativa accelerazione del workflow di sviluppo. Tuttavia, è cruciale sottolineare che, allo stato attuale, non esiste un sistema completamente "plug and play" e user-friendly che l'utente medio, un programmatore o un designer possa utilizzare in modo semplice e completo senza richiedere conoscenze pregresse o interazioni complesse, che generi codice a partire da uno sketch. A eccezione del sistema Misty, che richiede come base di partenza un codice HTML già funzionante, si percepisce una netta mancanza in questo panorama di un sistema che, nelle prime fasi di prototipazione, permetta a chiunque di trasformare uno sketch concettuale in un'interfaccia funzionante.

I limiti attuali degli approcci di web design basati sull'intelligenza artificiale risiedono soprattutto nella mancanza di comprensione semantica profonda del layout, nella difficoltà di interpretare correttamente il contesto e le intenzioni progettuali e nella scarsa coerenza tra la struttura visiva generata e il codice sottostante. Inoltre, molti modelli soffrono di problemi di generalizzazione: funzionano bene su esempi tipici o dataset specifici, ma falliscono quando si trovano di fronte a design non convenzionali o a stili grafici innovativi e diversi dallo stile standard su cui sono stati allenati. Non esiste la possibilità che la generazione del codice si adatti allo stile di chi disegna, ma deve essere chi disegna a adattarsi allo stile che il modello sa riconoscere. Questo potrebbe risultare un problema che si traduce in scarsa qualità del codice dovuta alla mancata comprensione dell'input. Anche la qualità del codice prodotto rappresenta un ostacolo: spesso esso risulta ridondante, poco ottimizzato o non conforme alle migliori pratiche di accessibilità e mantenibilità.

In realtà il modello GPT-4V ha dimostrato prestazioni di elevata qualità nella generazione automatica di codice a partire da input visivi. Tuttavia presenta alcune limitazioni strutturali legate alla sua integrazione nei flussi di lavoro reali di designer e sviluppatori. In particolare, l'utilizzo di GPT-4V richiede l'impiego di strumenti esterni per la fase di disegno. L'utente deve infatti realizzare lo sketch dell'interfaccia in un editor grafico separato, per poi caricare l'immagine su ChatGPT e fornire un prompt testuale per ottenere la corrispondente versione in codice. Questo approccio, pur efficace in termini di qualità dell'output, risulta poco efficiente dal punto di vista operativo, poiché frammenta il processo di progettazione tra ambienti e strumenti diversi e risulta scomodo nel caso in cui il disegno vada rifatto più volte o corretto volta per volta.

Lo strumento che si vuole creare mira invece a superare tale discontinuità, integrando in un unico ambiente digitale tutte le fasi del processo: la creazione e la modifica dello sketch, la generazione automatica del codice e la visualizzazione del risultato. In questo modo, l'utente può intervenire in tempo reale sul disegno, osservare immediatamente la traduzione del proprio sketch in codice e confrontare il prototipo generato con l'interfaccia originale, senza dover ricorrere a software esterni o passaggi intermedi.

La disponibilità di un tale strumento è fondamentale per fornire un riscontro diretto e immediato tra il prototipo iniziale e il suo sviluppo futuro come interfaccia completa e funzionale, un passo che al momento resta il principale ostacolo per la piena adozione di queste tecnologie nella pratica quotidiana.

2.2 Tecniche di Fine-Tuning per modelli di AI

Il fine-tuning dei modelli di intelligenza artificiale rappresenta una fase cruciale nel processo di adattamento di un modello pre-addestrato a specifici compiti o

domini applicativi. Partendo da un modello già addestrato su un ampio dataset generale, il fine-tuning consiste nel riaddestrare il modello su un set di dati più ristretto e specifico, ottimizzando i suoi parametri per migliorare le performance nel contesto target. Tale procedimento consente di sfruttare efficacemente le conoscenze pregresse del modello, riducendo notevolmente il tempo e le risorse computazionali necessarie rispetto all'addestramento da zero. Abbiamo visto nella sezione precedente come alcuni modelli venissero finetunati su dataset `Immagini - Codice`, per aumentare la specializzazione del modello su quel task specifico, con conseguente aumento delle performance e della qualità del codice generato, in pulizia e correttezza.

2.2.1 Fine-Tuning tradizionale vs tecniche leggere

Esistono diversi tipi di finetuning, che si adattano a esigenze diverse e richiedono costi diversi. Le tecniche possono essere suddivise in approcci che coinvolgono l'addestramento dell'intero modello, o di gran parte di esso, e approcci più efficienti dal punto di vista dei parametri, chiamati PEFT.

- L'approccio più diretto è il finetuning completo, o full finetuning, in cui utilizzando i pesi del modello preaddestrato come punto di partenza, si ri-addestrano tutti i parametri del modello sul nuovo dataset specifico. Si usa tipicamente un tasso di apprendimento (learning rate) più basso rispetto al pre-addestramento, per evitare di sovrascrivere drasticamente le conoscenze apprese in precedenza. Questo è il metodo che offre le migliori performance in senso assoluto se si dispone di dati a sufficienza, ma è estremamente costoso in termini computazionali, di memoria e tempo e aumenta il rischio di overfitting sul dataset più piccolo.

Le altre tecniche invece prevedono un training parziale del modello. Per ottenerlo si congelano alcuni layer del modello, concentrando l'addestramento solo su una parte.

- Partial fine-tuning: Si procede con il congelamento dei livelli iniziali, o i livelli importanti, cioè quei layer che hanno appreso caratteristiche generiche di base utili per molti compiti. Questi layer, per preservare le conoscenze generali vengono congelati, ossia i loro pesi non vengono aggiornati durante il processo. Vengono invece addestrati solo i livelli successivi, cioè quelli tipicamente più vicini all'output, che apprendono caratteristiche più specifiche per il compito. In particolare viene addestrato il livello di output, che viene spesso sostituito e inizializzato casualmente per adattarsi al numero di classi o al formato del compito specifico. Questo metodo porta a risultati peggiori di un finetuning completo, ma richiede meno dispendio computazionale.

2.2.2 Low-Rank Adaptation (LoRA): principi e applicazioni

L'ultimo metodo di finetuning è chiamato PEFT, che sta per Parameter-Efficient Fine-Tuning. Per PEFT si intende un insieme di tecniche progettate per ridurre drasticamente il numero di parametri addestrabili, rendendo il fine-tuning di modelli molto grandi più accessibile ed economico. I pesi del modello base rimangono congelati e solo pochi pesi (detti *adapters*) vengono aggiornati. Un caso interessante di PEFT è il *LoRA* (Low-Rank Adaptation) fine-tuning. Si tratta di una tecnica innovativa per adattare modelli di intelligenza artificiale pre-addestrati in modo efficiente, particolarmente utile per modelli di grandi dimensioni o per ridurre al minimo le risorse computazionali. Il principio alla base di LoRA è l'aggiunta degli adapters nella forma di piccole matrici a basso rango (low-rank matrices) che vengono aggiunte nei layer ai livelli di attenzione del modello pre-addestrato. Vengono addestrati solo i pesi di queste matrici di adattamento, che sono una frazione minima dei pesi totali, senza toccare nessun altro parametro dell'intero modello. Questo approccio è molto più rapido ed efficiente, perché si concentra sull'addestramento solo del nuovo livello e preserva la conoscenza completa del modello pre-addestrato. In pratica, LoRA introduce delle nuove matrici di adattamento nel modello, che combinate linearmente con i pesi fissi del modello permettono di adattare efficacemente la rete a nuovi compiti senza sovrascrivere la conoscenza preesistente, in quanto la nuova conoscenza ottenuta dal finetuning viene assorbita solo da queste nuove matrici aggiuntive. Questa metodologia si presta particolarmente bene al fine-tuning di modelli di linguaggio di grandi dimensioni o di modelli multimodali, consentendo un addestramento più veloce e meno dispendioso rispetto alle tecniche tradizionali. Facilita inoltre anche la condivisione e il riutilizzo degli adattamenti specifici a domini particolari. LoRA rappresenta quindi una svolta nell'efficienza del fine-tuning, bilanciando flessibilità e contenimento delle risorse, e viene sempre più adottata in scenari applicativi avanzati di intelligenza artificiale.

Data la sua efficienza e il tempo estremamente ridotto per eseguire un finetuning, LoRA sarà l'approccio utilizzato nel sistema nella fase di personalizzazione

Capitolo 3

Progettazione

3.1 Descrizione generale del sistema

In questa sezione verrà presentata una descrizione del sistema proposto all'interno di questa tesi. Dopo aver delineato nei capitoli precedenti il contesto teorico in cui tale progetto si colloca - con particolare attenzione all'evoluzione del web design contemporaneo e al ruolo sempre più centrale dell'intelligenza artificiale - risulta ora necessario entrare nel merito della soluzione che si intende sviluppare. La seguente analisi ha lo scopo di definire le principali caratteristiche che il sistema deve possedere per essere effettivamente funzionale, usabile e in linea con le esigenze del contesto applicativo. La prima parte del capitolo sarà dunque dedicata all'individuazione delle funzionalità attese, nonché alla chiarificazione delle aspettative e dei bisogni degli utenti finali. Questa riflessione consentirà di stilare una lista di requisiti, distinguendo tra requisiti funzionali e non funzionali, che costituirà la base progettuale per le fasi successive. Infine, verrà proposto uno schema architetturale del sistema, concepito per tradurre i requisiti individuati. Tale rappresentazione architetturale fungerà da punto di riferimento per lo sviluppo e l'implementazione, offrendo una panoramica delle componenti principali, delle interazioni tra di esse e delle tecnologie che ne supporteranno il funzionamento.

Nel Capitolo 1 è stata analizzata la situazione attuale dei metodi di prototipazione web basati sull'intelligenza artificiale. È emerso come gli strumenti disponibili si limitino principalmente a fornire assistenza nella modifica del codice, nella generazione di porzioni di codice o nella spiegazione e risoluzione di problemi relativi a codice già esistente. Si tratta, dunque, di un approccio non specificamente orientato alla programmazione web e al web design , ma piuttosto applicabile in modo generale all'ambito della programmazione generica. Tuttavia, è stato anche osservato che, in questo contesto, la possibilità di creare pagine web interamente generate dall'IA è ormai concreta e, in diversi casi, produce risultati di buona qualità.

Nonostante ciò, la generazione automatica di codice per interfacce web risulta ancora poco integrata all'interno dei consueti flussi di prototipazione: attualmente non esiste infatti uno strumento che, all'interno di tale flusso, sfrutti pienamente le potenzialità dell'IA per produrre un prototipo funzionante in linguaggio HTML. Le cause di questa limitata adozione possono essere individuate in diversi fattori:

- È possibile ottenere ottimi risultati dai miglior modelli generativi, ma attraverso i prompt testuali si perde il controllo completo sul layout finale della pagina generata.

- Per ottenere una maggiore precisione e controllo sulla struttura dell'interfaccia è possibile generare codice a partire da immagini di siti web esistenti. Tuttavia, questa modalità risulta poco coerente con il processo di design, poiché implicherebbe la creazione di un'interfaccia di partenza già definita, che verrebbe solo modificata o migliorata, perdendo di senso nel'ottica della creazione di una nuova pagina web.

- Una soluzione più coerente con il processo di design consisterebbe nel generare codice a partire da sketch realizzati a mano. Il disegno a mano libera rappresenta infatti uno step quasi imprescindibile nella fase di prototipazione a bassa fedeltà; poter convertire direttamente questi sketch in codice costituirebbe quindi un supporto concreto e immediato per ottenere rapidamente un prototipo funzionante. Tuttavia, questa possibilità è ancora limitata dalla qualità del codice generato. Abbiamo visto nel capitolo precedente come alcuni studi hanno evidenziato che la generazione di codice di interfacce complete a partire da input visivi, quali sketch disegnati a mano, non raggiunga lo stesso livello qualitativo di quella basata su immagini di interfacce reali. Tale difficoltà deriva principalmente dalla complessità del riconoscimento accurato della struttura intrinseca della pagina e delle relazioni spaziali tra i vari elementi. Le immagini di pagine web reali presentano, infatti, caratteristiche più standard, e i modelli di IA risultano generalmente meglio addestrati al loro riconoscimento. Gli sketch, al contrario, presentano una variabilità molto elevata nello stile grafico e nella rappresentazione degli elementi, rendendo più difficile per il modello generalizzare e interpretare correttamente la struttura della pagina. Gli studi mostrano, inoltre, che l'accuratezza nel riconoscimento dei componenti dell'interfaccia utente dipende fortemente dalla qualità del disegno e risulta maggiore in presenza di layout semplici e stili di disegno standardizzati.

- Un ulteriore limite, approfondito nella Sezione 2.1.3, riguarda la frammentazione del processo di generazione di codice a partire da immagini. Attualmente, un flusso di lavoro che consenta di generare codice utilizzabile in modo diretto

durante la fase di prototipazione risulta, infatti, poco integrato e disarticolato. Indipendentemente dal tipo di immagine di partenza — che si tratti di uno sketch disegnato a mano o di un'immagine di un'interfaccia realistica — è naturale che nel corso della prototipazione questa venga modificata o corretta più volte. Tuttavia, tali modifiche richiedono l'utilizzo di editor grafici esterni, dai quali è necessario esportare un nuovo file da analizzare manualmente mediante un modello di intelligenza artificiale. Questo procedimento comporta una discontinuità operativa significativa poiché costringe il progettista a passare continuamente tra strumenti differenti. Attualmente, infatti, non esiste una soluzione integrata che combini in un unico ambiente un editor grafico e un modello generativo di IA in grado di produrre e restituire direttamente il codice corrispondente all'interfaccia disegnata.

Queste considerazioni, soprattutto l'ultima, hanno portato allo sviluppo dell'idea di realizzare uno strumento integrabile nel processo di prototipazione, capace di generare codice a partire dagli sketch prodotti nella fase di design a bassa fedeltà. L'obiettivo è quello di progettare un sistema semplice da utilizzare, che consenta il disegno a mano libera direttamente all'interno dell'applicazione, evitando così l'uso di software esterni e riducendo la frammentazione del flusso di lavoro. Inoltre, il sistema dovrebbe essere in grado di riconoscere e adattarsi allo stile specifico del disegnatore, superando la limitazione degli approcci attuali che si basano su stili standardizzati. In questo modo, il modello sarebbe in grado di tradurre fedelmente l'intento progettuale dell'utente, mantenendo coerenza tra sketch e prototipo finale.

L'obiettivo principale del sistema è quindi quello di permettere all'utente di disegnare sketch di interfacce e trasformarli in vere interfacce funzionanti, scritte in HTML. Il focus va quindi dato al disegno e alla visualizzazione dell'interfaccia generata. Il sistema dovrà consentire all'utente di rappresentare liberamente, attraverso il disegno a mano libera, l'interfaccia che intende progettare, senza imporre vincoli che possano limitare la traduzione del pensiero progettuale in uno sketch digitale. Il disegno assume dunque un ruolo centrale all'interno del sistema, ma di pari rilevanza deve essere la capacità di trasformare tali rappresentazioni grafiche in codice eseguibile e, conseguentemente, di visualizzare l'interfaccia prodotta. Queste due funzionalità devono essere a stretto contatto, in modo da consentire all'utente di confrontare in maniera immediata il prototipo generato con lo sketch originale, verificandone coerenza e corrispondenza.

Rimane tuttavia un problema intrinseco legato all'uso degli sketch come input visivo per la generazione. Per generare correttamente il codice a partire da un disegno, il sistema deve essere in grado di riconoscere e interpretare lo stile di disegno dell'utente specifico, così da estrarre tutte le informazioni necessarie senza allucinare, cioè introdurre elementi inesistent, né alterare la disposizione spaziale dei componenti dell'interfaccia. Il principale ostacolo risiede nel fatto che nessun

dataset di addestramento esistente contiene esempi di sketch realizzati in tutti i possibili stili di disegno. Di conseguenza, un modello generalista non è in grado di garantire un riconoscimento accurato per ogni utente. La soluzione a questa problematica viene affrontata nella sezione seguente, in vui viene brevemente introdotta la logica per conseguire tale obiettivo.

3.1.1 Personalizzazione

La soluzione proposta consiste nel non utilizzare un modello pre-addestrato su una molteplicità di stili, ma piuttosto nell'adattare un modello generale al bisogno, addestrandolo specificamente sullo stile dell'utente. Questa strategia, tuttavia, introduce due ulteriori problematiche:

- l'addestramento di un modello di IA richiede notevoli risorse computazionali e tempi di elaborazione elevati, difficilmente compatibili con un'applicazione web che deve operare in modo rapido e interattivo.

- tale approccio presuppone la disponibilità di un dataset personalizzato per ciascun utente, contenente una serie di sketch realizzati con il suo stile e il codice corrispondente per l'addestramento.

Finetuning

Per superare questi limiti è necessario adottare un metodo di training che consenta di ottenere risultati di qualità in tempi ridotti. La soluzione individuata è l'impiego di una tecnica di fine-tuning leggero basata su LoRA. Questo approccio presenta un duplice vantaggio: in primo luogo, il tempo di addestramento risulta notevolmente ridotto. LoRA infatti non modifica i parametri di base del modello, ma genera parametri aggiuntivi, gli adapters, che vengono addestrati separatamente. In secondo luogo, questi parametri aggiuntivi catturano e conservano la conoscenza specifica dello stile dell'utente, potendo essere salvati, riutilizzati o caricati all'occorrenza. In tal modo, il modello principale mantiene invariati i propri pesi, mentre gli adapters vengono caricati dinamicamente in base allo stile richiesto. Questo permette di eseguire numerose sessioni di fine-tuning personalizzate, ciascuna delle quali genera soltanto pochi megabyte di dati, senza compromettere l'integrità del modello originale. I dati così ottenuti possono essere salvati in un modo tale da ricordare a quale stile sono relativi, e poi essere integrati nel modello ogni qualvolta si desideri generare codice conforme a quello stile di disegno.

Dataset Personalizzato

Per ottenere i dati personalizzati necessari alla fase di training, è indispensabile disporre di sketch realizzati secondo lo stile specifico dell'utente. Poiché tali interfacce non sono disponibili a priori, è necessario che vengano disegnate direttamente dall'utente. L'idea alla base di questo processo consiste nel far disegnare all'utente alcune interfacce di cui si possiede già il codice sorgente. In questo modo è possibile costruire un dataset personalizzato composto da coppie sketch–codice, dove lo sketch rappresenta la versione disegnata a mano e il codice corrisponde alla struttura HTML dell'interfaccia di riferimento da cui deriva lo sketch. Questo dataset così generato potrà successivamente essere utilizzato per eseguire il fine-tuning del modello, permettendo al sistema di apprendere lo stile di disegno dell'utente e di riprodurlo fedelmente durante la generazione automatica di nuove interfacce, dopo aver caricato gli adapters corrispondenti.

Le modalità operative e gli aspetti tecnici relativi a tale procedura saranno approfonditi nel Capitolo 4. In questa fase, ci limiteremo ad aggiungere alla progettazione questa funzione, insieme alla possibilità di salvare i nuovi parametri derivanti dal processo di fine-tuning, garantendo così la persistenza e il riutilizzo delle personalizzazioni effettuate.

3.1.2 Requisiti funzionali e non funzionali

Questa analisi ha portato a creare la seguente lista di requisiti funzionali:

- **Disegno a mano libera.** Il sistema deve consentire all'utente la realizzazione di schizzi dell'interfaccia in modalità manuale, mettendo a disposizione strumenti adeguati per la creazione e la modifica del disegno. In particolare, devono essere garantite funzionalità di cancellazione, annullamento e ripristino delle azioni, spostamento o copia di porzioni selezionate, nonché la possibilità di eseguire operazioni di zoom.

- **Generazione automatica di codice HTML.** Il sistema deve permettere la conversione dello sketch in un corrispondente codice HTML, che risulti compilabile ed eseguibile.

- **Visualizzazione dell'interfaccia generata.** Deve essere possibile per l'utente visualizzare l'interfaccia derivata dal codice HTML generato, così da consentire un confronto diretto con lo sketch di partenza e verificarne coerenza e fedeltà.

- **Riconoscimento dello stile utente.** Questa funzionalità costituisce uno degli aspetti cardine del progetto. L'IA utilizzata dal sistema deve essere in grado di riconoscere lo stile di disegno proprio di ciascun utilizzatore,

generando interfacce che riflettano quello che l'utente intendeva nello sketch. Questo va implementato con un finetunin PEFT usando LoRA.

- **Gestione degli stili di disegno.** In aggiunta, grazie all'utilizzo di LoRA che permette di effettuare più finetuning senza alterare il modello, è possibile personalizzare il modello su più stili. Ogni utente deve quindi poter definire, salvare e utilizzare diversi stili di disegno, con la possibilità di selezionare quello desiderato in fase di generazione. Il sistema dovrà inoltre ovviamente consentire la creazione e l'aggiunta di nuovi stili personalizzati.
- **Gestione di più layout visivi.** Al fine di garantire la massima accessibilità e adattabilità, il sistema deve prevedere la disponibilità di più layout visivi, così da risultare utilizzabile da un'utenza eterogenea e in diversi contesti operativi.

Per quanto riguarda i requisiti non funzionali, è importante aggiungere che il sistema debba risultare accessibile e utilizzabile anche da utenti privi di esperienza specifica nei settori dell'intelligenza artificiale e della programmazione web. Ci si aspetta che l'utilizzatore medio sia un designer, o un programmatore web, due categorie che non sono da considerare necessariamente esperte nell'ambito dell'IA. L'obiettivo è garantire un'interazione intuitiva, rapida ed efficace, in grado di ridurre al minimo le barriere di accesso tecnologiche.

- **Facilità di utilizzo.** Il sistema deve presentare un'interfaccia chiara e immediata, evitando l'inclusione di funzionalità superflue che possano generare confusione o distrarre l'utente dalle attività principali.
- **Velocità.** L'elaborazione e la generazione dell'interfaccia devono avvenire in tempi ragionevoli, così da non compromettere l'esperienza d'uso né ostacolare il flusso di lavoro dell'utente.

3.1.3 Schema architetturale di alto livello

I requisiti elencati ci aiutano a formulare uno schema di quello che potrebbe essere l'architettura del sistema. Come discusso in precedenza, il sistema deve essere accessibile e di facile utilizzo per un'ampia tipologia di utenti. In quest'ottica, la scelta più appropriata risulta essere quella di sviluppare una web application. L'opzione di realizzare un'applicazione mobile è stata scartata, poiché il sistema richiede la possibilità di disegnare a mano libera, e l'esperienza di disegno su schermi di piccole dimensioni, come quelli di uno smartphone, risulterebbe poco agevole e imprecisa. La soluzione basata su un'applicazione web, invece, garantisce maggiore accessibilità e flessibilità, consentendo l'utilizzo sia da computer sia da tablet. Questi ultimi, in particolare, rappresentano gli strumenti più adatti alla progettazione e al disegno digitale a mano libera, offrendo un buon compromesso

tra precisione e comfort d'uso. La parte più complessa è invece la parte di backend: è necessario trovare una piattaforma che possa occuparsi dell'hosting e del training del modello IA, fino ad arrivare alla comunicazione tra questo e il client. Serve che questa piattaforma abbia la potenza di calcolo sufficiente per ospitare il modello e ottenere inferenze da questo in tempi ragionevoli. Lo stesso vale per il finetuning: nonostante il finetuning con LoRA si molto più rapido di uno tradizionale, è comunque richiesta una buona potenza di calcolo per ultimarlo in tempi che si concilino con i tempi di attesa che un utente si aspetti. Terzo aspetto fondamentale da considerare è la comunicazione tra il modello e il frontend e viceversa. In particolare il frontend deve poter mandare gli sketch al modello per l'elaborazione, che termina con un codice generato, e il modello deve poi comunicare questo codice al frontend, che visualizza la pagina relativa. Dividiamo per ora questa architettura in tre parti: la parte Client, la parte del modello e la parte di comunicazione tra i due, che rimane ancora non ben definita.

Figura 3.1: Bozza di Architettura

Client

Occupiamoci inizialmente del Client.

Per quanto riguarda il lato client, come già detto, si è ritenuto opportuno orientarsi verso la realizzazione di una web application, in quanto tale soluzione garantisce un'elevata accessibilità sia da computer sia da tablet. La scelta del framework è ricaduta su React, considerato uno dei framework JavaScript più diffusi e consolidati a livello internazionale. Questo per la sua facilità di utilizzo, la sua ampia comunità di sviluppatori e l'ecosistema di librerie disponibili che consentono non solo di beneficiare di soluzioni già validate per problematiche comuni, ma anche di ridurre sensibilmente i tempi di sviluppo e di manutenzione del progetto. L'intero frontend verrà implementato in javascript. La struttura dell'app conterrà una pagina in cui disegnare e visualizzare l'interfaccia generata, mentre in una pagina a parte sarà implementata la meccanica di raccolta degli sketch per la personalizzazione. L'utente avrà unicamente a disposizione queste due pagine e nient'altro, a parte gli strumenti di navigazione e ovviamente la schermata di login, per concentrare il focus sul compito principale del sistema.

Backend

Per quanto riguarda l'architettura server, è innanzitutto necessario disporre di una risorsa in grado di ospitare il modello di intelligenza artificiale responsabile della generazione di codice. Tale risorsa deve garantire una capacità di calcolo sufficiente a svolgere questa attività in tempi ragionevoli, assicurando efficienza e prestazioni adeguate. Inoltre, deve poter permettere il finetuning personalizzato del modello, precisamente un finetuning LoRA, con la possibilità di gestire e salvare i parametri ottenuti. Ciò significa che sarebbe sufficiente utilizzare un API di un provider di servizi AI, come potrebbe essere OpenAI. Questa soluzione risolverebe il problema dell'inferenza, ma non soddisferebbe tutto l'aspetto di gestione del finetuning personalizzato. È necessario un ambiente che permetta la gestione completa del modello. Risulta indispensabile anche un'infrastruttura che consenta la comunicazione tra il frontend e la piattaforma su cui il modello di IA è ospitato: il frontend deve poter inviare gli sketch, che rappresentano immagini — un tipo di dato ad alta intensità di elaborazione — al modello per la loro elaborazione. A seguito di questa, il modello restituisce il codice generato, che deve essere inviato nuovamente al frontend affinché venga visualizzato all'utente. Queste esigenze possono essere soddisfatte tramite l'impiego di molteplici servizi distinti, sebbene ciò comporti la complessità di integrare diversi sistemi esterni tra loro. In alternativa, è possibile usufruire di servizi integrati come Amazon Web Services (AWS), una piattaforma di cloud computing tra le più complete ed efficienti disponibili, che fornisce strumenti e infrastrutture capaci di rispondere in modo unificato a tutte queste necessità.

Nel contesto analizzato, AWS presenta la soluzione a ogni punto sopra elencato. In particolare:

- **Hosting e training del modello: AWS Sagemaker**

  Sagemaker è una piattaforma di machine learning basata sul cloud e completamente gestita. È progettata per semplificare e accelerare l'intero processo di sviluppo del machine learning, consentendo di creare, addestrare e implementare rapidamente modelli ML. Contiene ambienti di sviluppo integrati, come SageMaker Unified Studio o SageMaker notebook, che supportano IDE popolari (JupyterLab, VS Code, RStudio). Sarà proprio su un notebook SageMaker che ospiteremo il nostro modello. Questo ambiente permette la gestione e l'hosting del modello di intelligenza artificiale, insieme alla possibilità di training personalizzato. SageMaker oltre all'hosting offre anche altre funzionalità aggiuntive che attualmente esulano dalle nostre necessità, ma porebbero tornare utili per sviluppi futuri come distribuire e mettere in produzione modelli di machine learning. Il motivo principale per cui si è scelto SageMaker non è tanto la possibilita di hosting, in quanto altri servizi (come gia solo JupyterLab, su cui sagemaker si basa) sarebbero stati sufficienti, ma

l'integrazione nativa con tutto l'ambiente AWS, che permette di utilizzare e intrecciare in modo semplice gli altri servizi AWS, come si vedrà in seguito.

- **Comunicazione client-server: AWS Lambda e Amazon API Gateway**

  Per gestire la comunicazione tra la web application e il modello ospitato su SageMaker in maniera flessibile e scalabile, vengono utilizzati AWS Lambda e Amazon API Gateway. Le funzioni Lambda consentono di eseguire codice serverless in risposta a eventi specifici, come le richieste provenienti dal frontend, senza la necessità di gestire direttamente un'infrastruttura server. API Gateway, invece, agisce come punto di ingresso per le richieste HTTP provenienti dalla web app, instradandole verso le funzioni Lambda, le quali si occupano di interfacciarsi con SageMaker. Questo approccio consente di implementare una comunicazione efficiente, sicura e altamente scalabile tra l'interfaccia utente e il modello di intelligenza artificiale, garantendo al tempo stesso una separazione chiara dei compiti tra le varie componenti del sistema. Questo approccio funziona però in una sola direzione, cioè dal frontend al modello. Questo perchè la lambda function non può collegarsi con il frontend. Anche se venisse triggerata da un evento, quale la fine della generazione del codice o la fine del finetuning, la lambda function non ha modo di collegarsi con il frontend per passargli dei dati. O meglio, ha la possibilità di passare dei dati al frontend solo come risposta all'evento che il frontend stesso genera, comunicando e passando i dati attraverso l'API Gateway, a cui il frontend si è collegato inizialmente. In caso di operazioni asincrone, poichè la lambda è di tipo serverless - ossia l'ambiente di esecuzione si attiva solo in risposta a un evento e si spegne a esecuzione terminata - c'è bisogno che sia il notebook sagemaker a generare l'evento per collegarsi alla lambda. In questo caso, la lambda non ha modo di collegarsi con il frontend e comunicare i dati. Per la comunicazione in questo senso, c'è bisogno di inserire un'altro elemento.

- **Storage e comunicazione modello-client: Amazon S3**

  Il servizio Amazon S3 (Simple Storage Service) è solitamente impiegato per l'archiviazione e la gestione dei dati. S3 presenta un integrazione nativa con sagemaker e, come vedremo, è possibile collegarlo direttamete con il frontend attraverso delle librerie. All'interno dell'infrastruttura viene utilizzato per salvare i parametri ottenuti dal finetuning relativo a uno stile e gli sketch sui quali il finetuning è effettuato. Proprio grazie all'integrazione con il notebook è molto semplice sia caricare da questo i parametri su S3, che leggerli per utilizzarli quando necessario. Ma la seconda funzione cardine per cui è utilizzato S3 è la comunicazione: funge da canale per il passaggio delle immagini dal client al modello. Ad esempio, lo sketch generato dall'utente può essere caricato su un bucket S3 dalla web app tramite le API delle librerie

apposite. Da qui SageMaker può poi recuperarlo facilmente per l'elaborazione e restituire a sua volta il risultato nello stesso bucket. Questa operazione non può essere fatta attraverso la lambda perchè il passaggio delle immagini è un'operazione onerosa e la lambda function non è pensata per compiti pesanti. Infine S3 è utilizzato per la comunicazione modello-client. L'unico modo per poter passare i dati elaborati dal modello al frontend è usare S3 come tramite. Il frontend non può recupererare i dati dal notebook sagemaker, ma può farlo del bucket S3 specifico, su cui sagemaker li ha caricati in precedenza .

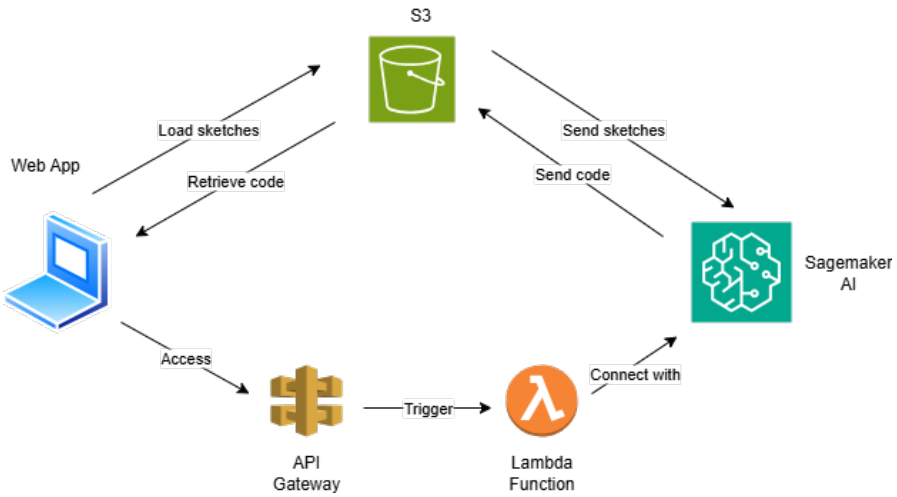

Figura 3.2: Architettura utilizzando AWS

L'immagine mostra l'architettura del sistema utilizzando i servizi di cloud computing di AWS. L'intera architettura proposta è serverless e si basa su tariffe di pagamento a consumo per i vari servizi.

AWS amplify

La scelta di utilizzare Amazon Web Services non è motivata unicamente dalla possibilità di disporre, in un unico ambiente, di tutti i servizi backend necessari per l'applicazione. AWS offre infatti anche AWS Amplify, una libreria javascript che semplifica notevolmente l'integrazione diretta tra il frontend e i servizi cloud di AWS. Amplify mette a disposizione API che consentono di collegare l'applicazione ai principali servizi come Amazon S3, database DynamoDB, e Amazon Cognito per l'autenticazione degli utenti, eliminando la necessità di mantenere un server dedicato costantemente in ascolto per la gestione delle richieste.

Nella comunicazione con il modello infatti, non essendoci nessun server, è necessario che sia direttamente il frontend a mandare gli sketch al notebook

sagemaker. Questo viene fatto attraverso le API della libreria Amplify Storage, che permettono di caricare su S3 gli sketch dell'utente: uno nel caso della generazione di codice, molteplici nel caso si debba effettuare un finetuning. Viceversa, quando il codice è stato generato, il frontend lo legge direttamente da s3. Una volta completata la generazione, il codice prodotto dal modello viene letto direttamente dal bucket S3 dal frontend, garantendo così un flusso di comunicazione semplice e privo di server intermedi.

In seguito all'esecuzione di un fine-tuning, i parametri addestrati vengono salvati su S3 all'interno di un path specifico per ciascun utente e ciascuno stile creato. Poiché tali parametri devono essere recuperabili per generazioni future, è necessario memorizzare questi path in un database. Anche in questo caso, AWS Amplify fornisce un'integrazione diretta con Amazon DynamoDB, consentendo la creazione e la gestione del database attraverso le API di Amplify direttamente dal frontend. All'interno di questo database vengono quindi associati a ogni utente registrato uno o più percorsi relativi ai file dei parametri salvati su S3, consentendo al sistema di recuperare rapidamente i dati necessari per eseguire generazioni personalizzate coerenti con lo stile di disegno di ciascun utente.

Capitolo 4

Implementazione

4.1 Setup

Nel capitolo precedente sono state introdotte le tecnologie abilitanti alla realizzazione del sistema proposto. In questa sezione si descrivono invece le fasi di implementazione, articolate in tre componenti principali, strettamente interconnesse tra loro, ma non necessariamente realizzate in modo isolato l'una dall'altra:

- L'applicazione frontend, sviluppata in React.js, concepita come interfaccia attraverso la quale l'utente può disegnare e generare codice;
- L'infrastruttura AWS, incaricata di gestire l'elaborazione non visibile all'utente, ma centrale per il corretto funzionamento del sistema;
- L'integrazione tra infrastruttura e frontend, necessaria a garantire la comunicazione bidirezionale e la coerenza del sistema nel suo complesso.

Si è iniziato concentrandosi sullo studio dell'infrastruttura AWS e della documentazione relativa. Questa scelta è stata dettata dal fatto che l'ambiente in questione, oltre a rappresentare la componente più complessa e al tempo stesso la più critica per la riuscita del progetto, fosse anche una piattaforma nuova e mai utilizzata prima. Per questo è stato necessario un periodo preliminare di studio approfondito e mirato dell'ambiente AWS e delle sue specificità.

L'implementazione dell'infrastruttura è stata condotta in due passaggi distinti. In un primo momento, terminata la fase di studio, come test è stata realizzata un'infrastruttura di prova: si è fatto ricorso a un modello di intelligenza artificiale temporaneo, leggero e a bassa complessità, un **LLaVA-OneVision**, ospitato su Amazon SageMaker. Questo ha consentito di testare e verificare il funzionamento della comunicazione tra il frontend, le funzioni AWS Lambda e SageMaker, senza introdurre da subito la complessità del modello definitivo. In questa fase

anche l'applicazione client si presentava come una struttura essenziale, progettata esclusivamente per verificare la correttezza delle interazioni tra le componenti.

Successivamente si è passati allo sviluppo reale del progetto, iniziando dal frontend vero e proprio, dotato dei vari moduli e componenti necessari a garantire un'interazione efficace con l'utente finale. Una volta completato il frontend, l'infrastruttura AWS è stata implementata in modo completo, integrando il modello finale, adattando il codice nelle lambda functions e impostando il trasferimento delle immagini tramite S3.

Nei paragrafi successivi verranno analizzati in modo dettagliato i singoli passaggi di questo processo e i servizi AWS effettivamente impiegati, con l'obiettivo di mettere in evidenza il contributo specifico di ciascun componente al funzionamento complessivo della soluzione proposta.

Ruoli e Policy

Per quanto riguarda l'implementazione pratica dei servizi AWS, la fase iniziale dello sviluppo è stata dedicata prevalentemente alle attività di configurazione preliminare. L'architettura di AWS, infatti, si fonda su un sistema di ruoli e policy che regolano l'accesso e le autorizzazioni dei vari servizi. Ciascun servizio richiede specifici permessi per poter eseguire determinate operazioni ed è, pertanto, necessario predisporre in anticipo tali configurazioni nonostante, al bisogno, tali autorizzazioni siano comunque modificabili. Ad esempio, per poter accedere da un notebook SageMaker a un bucket S3 per scaricare degli sketch - che poi verranno elaborati - o per caricare il codice generato, è necessario che il ruolo associato all'utilizzo di SageMaker abbia a sua volta associata la policy che permette la lettura e scrittura su quel determinato bucket S3. Lo stesso vale per tutti gli altri servizi (Lambda Function, S3, ecc). In una prima fase si è dunque provveduto alla creazione dei ruoli destinati ai diversi servizi da impiegare, ai quali sono state successivamente associate le relative policy, contenenti le autorizzazioni necessarie per il loro corretto funzionamento. Solo a valle di questa attività di setup è stato possibile avviare l'implementazione vera e propria delle componenti del sistema.

4.2 Frontend

La programmazione pratica del frontend è stato il primo step dell'implementazione del sistema. Dopo aver testato l'infrastruttura Gateway-Lambda-SageMaker con un frontend di prova e aver assodato il funzionamento, è iniziata la parte di programmazione dell'app in linguaggio javascript.

4.2.1 Design del Frontend

Il processo di progettazione del frontend è stato avviato attraverso la creazione di un prototipo iniziale utilizzando il software Figma. Si è iniziato a prototipare la pagina adibita al compito principale, cioè quello di disegnare a mano libera lo schizzo dell'interfaccia da trasformare. L'idea è quella di dare all'utente la possibiltà di avere nella stessa schermata sia la zona di disegno (il *canva*) sia quella di visualizzazione (il *viewer*). Questo perchè l'obiettivo progettuale era quello di permettere all'utente di confrontare in modo rapido e diretto il risultato generato con lo sketch di partenza. A tal fine, le due sezioni dell'interfaccia destinate a questi compiti sono state disposte affiancate, privilegiando un design minimale, caratterizzato esclusivamente dagli strumenti essenziali per il disegno e da un numero limitato di pulsanti dedicati alla configurazione del layout.

Figura 4.1: Design iniziale della pagina di disegno

Come illustrato in Figura 4.1, l'interfaccia si compone principalmente di un canvas dedicato al disegno e di una sezione in cui viene mostrata l'interfaccia generata, accompagnata da una barra contenente gli strumenti di disegno. Per non occupare spazio destinato al disegno, la barra è stata pensata per essere mobile, trascinabile dall'icona della mano in alto. Il tasto *generate* in alto a sinistra funge da trigger per la conversione da sketch a codice. È inserito in una barra in cui compare anche il tasto per scambiare di posizione canva e viewer, usato per non svantaggiare eventuali utenti mancini. Questa funzione è utile anche nel caso in cui l'utente preferisca un diverso posizionamento dei due componenti. Per poter visualizzare

meglio l'interfaccia generata c'è la possibilità di espandere a schermo intero la preview di quest'ultima dal tasto in alto a sinistra nella zona del viewer. La barra superiore è stata invece pensata in modo più standard, includendo solamente il tasto per il login e la selezione dello stile. Questo perchè tali azioni sono considerate esterne al disegno e conseguentemente sono state posizionate nella barra esterna.

La pagina di personalizzazione è stata concepita nel seguente modo: poichè l'utente deve riprodurre interfacce già esistenti secondo il proprio stile grafico, si è ritenuto utile consentire la visualizzazione in background dell'interfaccia da riprodurr. Questa scelta consente di avere riferimento immediato per il disegno, agevolando l'utente in termini di proporzioni e distribuzione spaziale degli elementi.

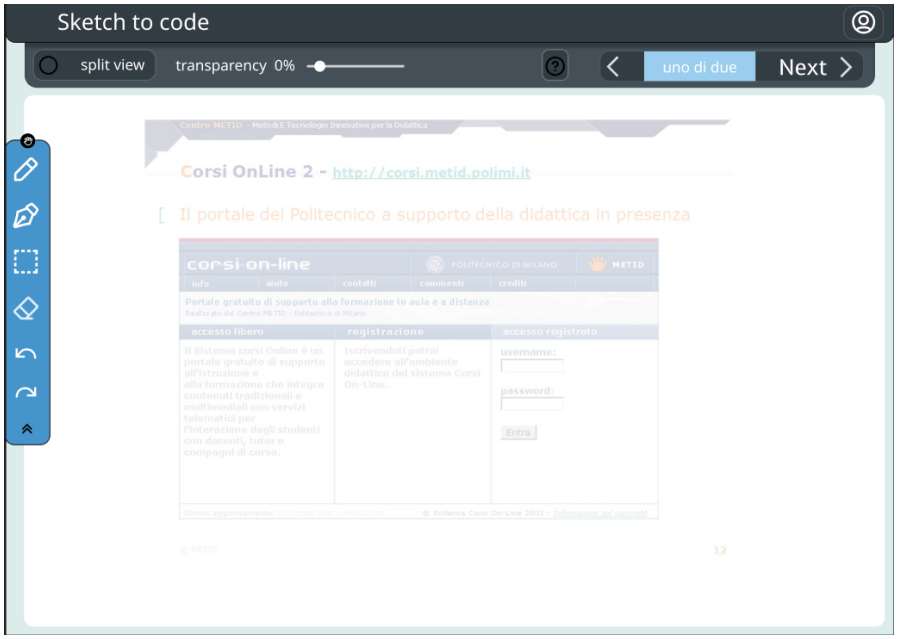

Figura 4.2: Design iniziale della pagina di personalizzazione del modello

La Figura 4.2 mostra la pagina di personalizzazione. Il canva rappresenta quasi l'intera pagina, essendo il disegno il focus principale. In questa pagina la barra inferiore presenta altri strumenti utili:

- la possibilità di regolare il livello di trasparenza dell'immagine di background. Ciò permette all'utente di concentrarsi alternativamente sul solo sketch o sul confronto diretto con l'interfaccia di riferimento.
- il pulsante per abilitare la split view per avere un confronto tra sketch e interfaccia fianco a fianco, da un'altra prospettiva.
- le frecce posizionate in alto a destra nella barra inferiore, per navigare tra le varie interfacce da disegnare.

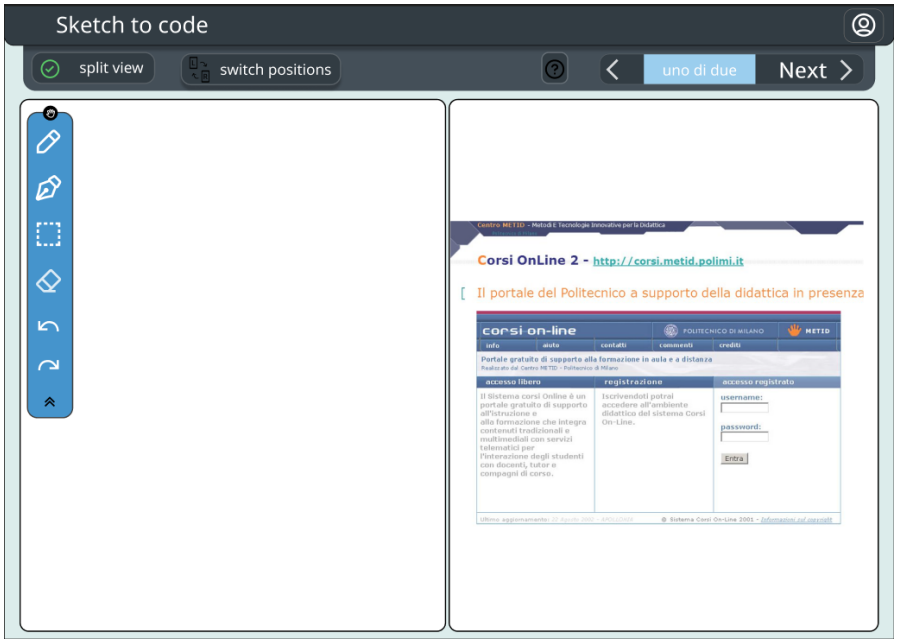

Figura 4.3: Visualizzazione in split view

In Figura 4.3 viene mostrata la modalità di visualizzazione in cui il canvas e l'interfaccia affiancati: si nota come questa modalità offra un altro modo rispetto all'immagine in background per confrontare lo sketch che si sta disegnando con l'interfaccia da replicare.

Design dei componenti

- La **barra degli strumenti** è stata pensata per essere trascinabile e posizionabile ovunque nello schermo al fine di non coprire in modo definitivo sezioni di canvas. Le icone relative agli strumenti scelti sono molto riconoscibili e omologate, tranne il pin per il trascinamento che mantiene uno stile proprio. Inizialmente in questa fase sono stati pensati due strumenti per tracciare le linee, la penna e la matita, ma dopo alcune riflessioni in fase di implementazione è stata eliminata la penna in quanto non ritenuto necessario avere più tratti in uno sketch. Stesso discorso per l'icona di accoricamento della barra, pensata come funzione per ridurre al minimo lo spazio occupato, ma eliminata nell'implementazione finale in quanto risultava macchinoso riaprirla per selezionare uno strumento.

- La **Barra superiore** funge da elemento estetico e contiene il nome dell'applicazione. Ha come funzione quella di contenere il menù dropdown da cui selezionare gli stili di disegno e passare dalla pagina di disegno a quella di

personalizzazione. Questo menù è stato posizionata in questa barra perchè esterna al contesto di disegno, coerentemente con le sue funzioni. In più è presente il pulsante adibito al login e logout collocato all'estrema destra come da design standard.

- La **Barra inferiore** è stata disegnata come elemento parte del contesto di disegno, in quanto contiene i pulsanti per modificare il layout e la visualizzazione e per generare il codice.

4.2.2 Implementazione del frontend e componenti

Dopo la fase di progettazione, si è passati alla programmazione effettiva del frontend, mediante l'utilizzo del framework React. In questa sezione vengono analizzati i componenti principali dell'applicazione, con particolare attenzione al canvas di disegno, alla barra degli strumenti trascinabile e al visualizzatore dell'interfaccia generata.

Canvas

Il canvas rappresenta l'elemento centrale dell'intera applicazione, poiché svolge un ruolo cruciale sia nella fase di sketching sia in quella di personalizzazione. La sua implementazione parte da un semplice componente HTML `<canvas>`, opportunamente esteso e racchiuso in un componente personalizzato denominato `<SimpleCanvas>`, che funge da contenitore per tutte le logiche e funzionalità necessarie. Il componente è stato implementato in modo da poter essere utilizzato sia nella pagina principale sia nella pagina di personalizzazione, in modo da avere un componente unico per il disegno, condiviso tra le due sezioni. Questo presenta tutte le funzionalità necessarie per entrambi gli usi: qualora una funzionalità non fosse necessaria in una delle due pagine, viene semplicemente disattivata.

All'interno di questo componente sono state implementate le seguenti funzionalità:

- **Logica di disegno**: Il meccanismo di disegno è stato sviluppato da zero, sfruttando le funzioni native di HTML per la gestione degli eventi del mouse. Questi eventi consentono di rilevare i movimenti e le pressioni dei tasti che sono stati tradotti in tratti grafici sul canvas.

- **Logica di selezione**: tra gli strumenti di disegno è stata introdotta la possibilità di selezionare una porzione dell'elaborato per spostarla, copiarla o eliminarla. La funzionalità è stata pensata sopratutto per alleggerire il lavoro nella fase di personalizzazione, in cui l'utente dovrà disegnare lo stesso componente più volte. In questo modo, dopo aver disegnato un componente a

```jsx
{/* Contenitore per i canvas */}
<div className='w-full h-full'
  style={{ position: 'relative', opacity: 0.5 + transparency }}
  onMouseDown={handleGlobalMouseDown}
  onMouseMove={handleGlobalMouseMove}
  onMouseUp={handleGlobalMouseUp}
  onTouchStart={handleGlobalMouseDown}
  onTouchMove={handleGlobalMouseMove}
  onTouchEnd={handleGlobalMouseUp}
  onMouseLeave={handleGlobalMousLeave} // Resetta lo stato quando il mouse esce dal canvas
  onTouchCancel={handleGlobalMouseUp}
>

  {/* Canvas principale per il disegno */}
  <canvas
    ref={canvasRef}
   width={canvasDimensions.w}
   height={canvasDimensions.h + 10}
    className="border border-gray-300 rounded-md shadow-md bg-white"
    style={{
      position: 'absolute',   //queste tre righr cambiate attenzione
      top: 1,                 //
      left: 3,                //
      cursor:
        tool === 'pen' || tool === 'eraser'
          ? 'crosshair' // Cursore a croce per penna/gomma
          : tool === 'crop'
          ? 'crosshair' // Cursore a croce per ritaglio
          : 'default', // Cursore predefinito
    }}
  />
```

Figura 4.4: Canvas principale e gestione eventi mouse

mano, può copiarlo o spostarlo dove serve. Anche in questo caso la logica è stata implementata da zero, utilizzando un canvas aggiuntivo sovrapposto a quello principale per visualizzare l'area di selezione.

```jsx
{/* Canvas sovrapposto per la selezione del ritaglio */}

<canvas
 ref={selectionCanvasRef}
 width={canvasDimensions.w}
 height={canvasDimensions.h  + 10}
 className="border-2 border-gray-700 rounded-md shadow-md"

   style={{
   position: 'absolute',
   top: 1,
   left: 3,
   pointerEvents: 'none', // Impedisce a questo canvas di interferire con gli eventi del canvas principale
  }}
/>
```

Figura 4.5: Canvas di selezione

- **Zoom e spostamento**: la funzionalità di zoom e pan è stata implementata tramite la libreria esterna **react-zoom-pan-pinch**. Essa mette a disposizione i componenti **TransformWrapper** e **TransformComponent**, che creano un ulteriore livello di wrapping e consentono, mediante la funzione **useControls**, di gestire ingrandimenti, riduzioni e spostamenti all'interno dell'area zoomata.

- **Undo/Redo**: L'undo/redo è un funzionalità fondamentale che alleggerisce molto il processo di disegno e abbassa il senso di frustrazione, nel caso di un operazione sbagliata. La logica è stata implementata in maniera ibrida. Il componente padre mantiene, tramite una `useState`, la lista ordinata degli stati del canvas, mentre il componente canvas è incaricato di salvare e aggiornare tali stati, oltre che a ridisegnare il contenuto corretto sul canvas, relativo allo stato attuale, permettendo così di gestire le operazioni di annullamento e ripristino.

```
useEffect(()=>{
  restoreCanvasFromDataURL(history[historyPointer])

}, [historyPointer])
```

Figura 4.6: Logica undo/redo nel canvas

```
//stati per undo/redo e cronologia
const [history, setHistory] = useState([]);
const [historyPointer, setHistoryPointer] = useState(-1);
```

Figura 4.7: Stati del canvas nel componente padre per undo/redo

- **Immagine in background**: questa funzionalità è utilizzata solamente nella fase di personalizzazione dello stile, ma è comunque integrata nel canvas. L'immagine in background viene gestita come un layer sottostantevis ualizzato solo nella pagina di personalizzazione. Il layer è sottoposto al canvas perchè, se fosse sovrapposto (cioè avesse un coordinata z più alta), oscurerebbe l'interazione del mouse con il canvas, non permettendo il disegno. La trasparenza è realizzata andando ad assegnare un valore dinamico all'opacità del ** ** in cui è contenuta l'immagine, utilizzando del CSS inline.
- **Bounding Boxes** Anche questa funzionalità è esclusiva della parte di personalizzazione. Non presente inizialmente nella fase di design, è stata aggiunta in corso di sviluppo. Durante l'implementazione ci si è resi conto che serviva un modo per indicare con precisione all'utente dove fossero i componenti e che limiti avessero, in modo da poter disegnare uno sketch con la stessa distribuzione spaziale dell'interfaccia di base. Questo aiuta il processo di fientuning perchè il codice utilizzato come label nell'addestramento sarà più coerente con lo sketch. Le bounding boxes vengono disegnate su un canvas sovrapposto e fungono da aiuto per l'utente, indicando i riquadri in cui devono essere disegnati i vari componenti da riprodurre

```
{/*Immagine in background */}
{isImageBackgrounds &&
<div className="absolute bg-gray-600"
    style={{width: canvasDimensions.w, height: canvasDimensions.h,
    backgroundImage: `url("/src/assets/sample_${page}.png")`, backgroundSize: 'contain',
    backgroundRepeat: 'no-repeat', backgroundPosition: 'center',  opacity: 1-transparency}}>
</div>}
```

Figura 4.8: Layer dell'immagine in background

- **Scrolling**: per la sezione di personalizzazione, in cui le interfacce da disegnare possono avere dimensioni variabili, è stato necessario garantire l'adattamento del canvas per mantenere le proporzioni originali delle immagini. A tale scopo è stato introdotto un ulteriore livello di wrapping attraverso il componente **ScrollShadow** della libreria **@heroui/react**: cio consente lo scorrimento verticale, in modo da non alterare le dimensioni dell'immagine.

Draggable Bar

È la barra che contiene tutti gli strumenti di disegno ed è mobile. Per renderla trascinabile è stato fatto uso della libreria **react-draggable** opportunamente modificata, che permette di trascinare un componente all'interno dello schermo. La possibilità di trascinamento è stata limitata al solo canvas di disegno, per non coprire altri pulsanti e/o elementi importanti.

Code viewer

Per la visualizzazione della pagina relativa al codice generato è stato usato un elemento `iframe`, un elemento standard HTML utilizzato per incorporare un altro documento all'interno del documento HTML corrente. Nel nostro caso incorpora un'altra pagina creata leggendo il codice restituito dal modello. Il modello produce esclusivamente codice HTML senza nessun CSS: per questo motivo il codice CSS è fisso e contenuto in una directory all'interno del file system dell'app. Quando l'`iframe` renderizza il codice HTML generato, automaticamente questo recupera il codice CSS.

Pagina di disegno

Alla fine della fase di implementazione del frontend, la main page, ossia la pagina di disegno dello sketch, si mostra come in figura. Per motivi di coerenza estetica, la barra degli strumenti è stata implementata in orizzontale 4.10. Il resto dell'estetica e il layout dei pulsanti sono rimasti invariati. Una volta che l'utente ha ultimato il suo sketch e procede con la generazione, viene visualizzata una progress bar all'interno del viewer. Questa progress bar ha una durata fissata di 140 secondi

```
<iframe
  ref={iframeRef}
  title="Codice Preview"
  style={{
    width: '100%',
    height: '600px',
    border: 'none',
    backgroundColor: 'white'
  }}
  onLoad={handleLoad}
  srcDoc={`
    ${generatedHTML.html}
  `}
></iframe>
{!iframeLoaded && <p>Caricamento anteprima...</p>}
</>
```

Figura 4.9: Componente iframe per la visualizzazione del codice

che è il tempo massimo misurato impiegato per la generazione. Questa scelta è stata fatta perchè il notebook non può comunicare in tempo reale con il frontend per notificare la progressione dell'elaborazione. Per questo motivo è stato fatto un mockup.

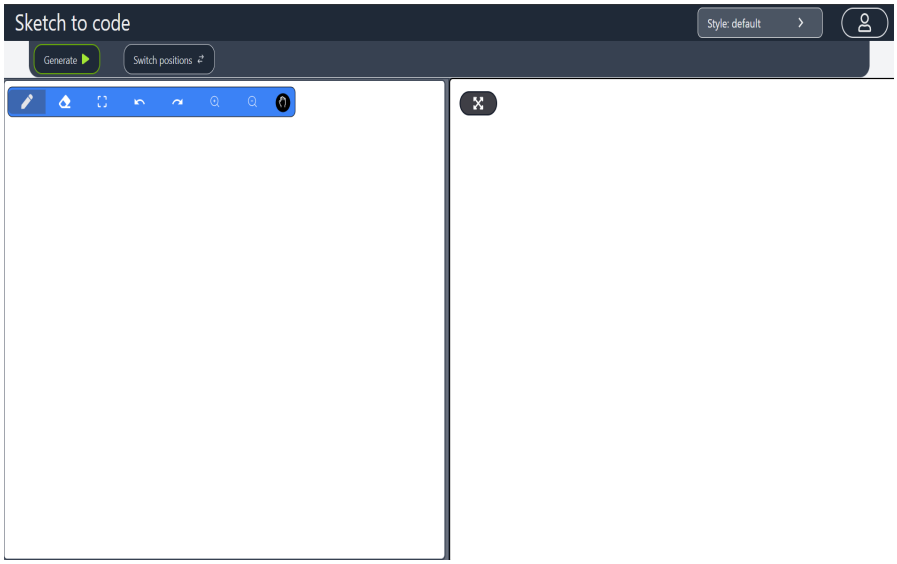

Figura 4.10: Sketch page

4.2.3 Pagina di personalizzazione

La pagina di personalizzazione è stata resa accessibile in due modi:

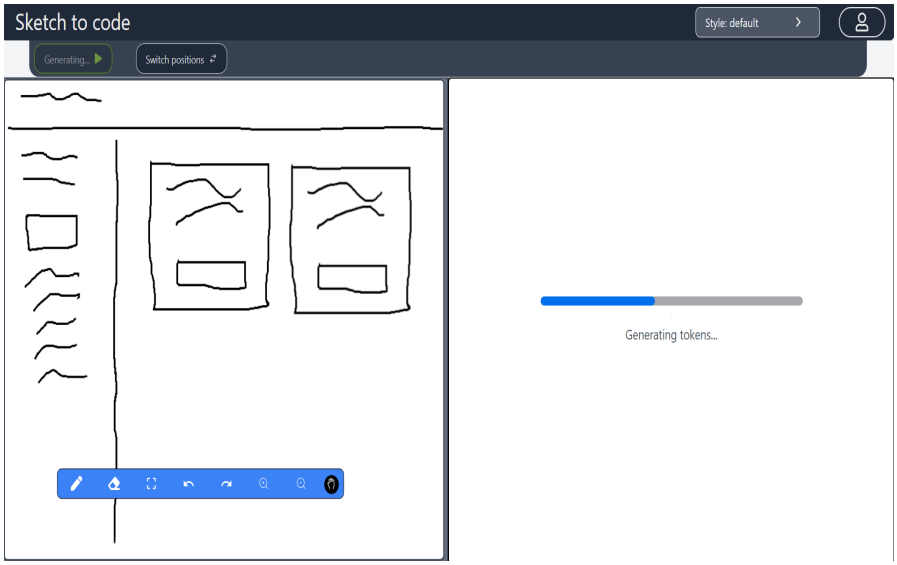

Figura 4.11: Progress bar mockup

- Dalla main page, attraverso il menù a tendina contenente l'elenco degli stili disponibili, l'utente può avviare la procedura per crearne uno nuovo venendo indirizzato alla pagina di personalizzazione.

- tramite un pop-up di benvenuto presentato automaticamente ai nuovi utenti al momento del primo accesso all'applicazione. Questo pop-up illustra il funzionamento generale dell'app e fornisce l'opzione per accedere alla personalizzazione.

In questa pagina l'utente è tenuto a disegnare le 5 interfacce note, secondo il suo stile di disegno. Per il primo accesso è presente un tutorial che spiega il funzionamento del processo di personalizzazione. Infatti questo potrebbe non risultare chiaro a tutti, soprattutto a utenti senza nessuna conoscenza nel campo dell'intelligenza artificiale. Non è immediato capire che è necessario ridisegnare le interfacce proposte per dare al modello la possibilità di allenarsi su tali disegni per poter riconoscere lo stile specifico dell'utente. Questo tutorial è comunque disponibile per essere rivisto in qualunque momento, cliccando sul tasto con l'icona del punto interrogativo nella barra inferiore.

In Figura 4.12 viene mostrato il pop-up di benvenuto, visualizzato esclusivamente al primo accesso. Per gli utenti già registrati, invece, il messaggio non ricompare, al fine di evitare ridondanze.

Nella sezione di personalizzazione, l'utente è chiamato a riprodurre una serie di interfacce presentate in trasparenza sul canvas, disegnandole secondo il proprio stile, ma mantenendo la disposizione spaziale originale dei componenti. Affinché il

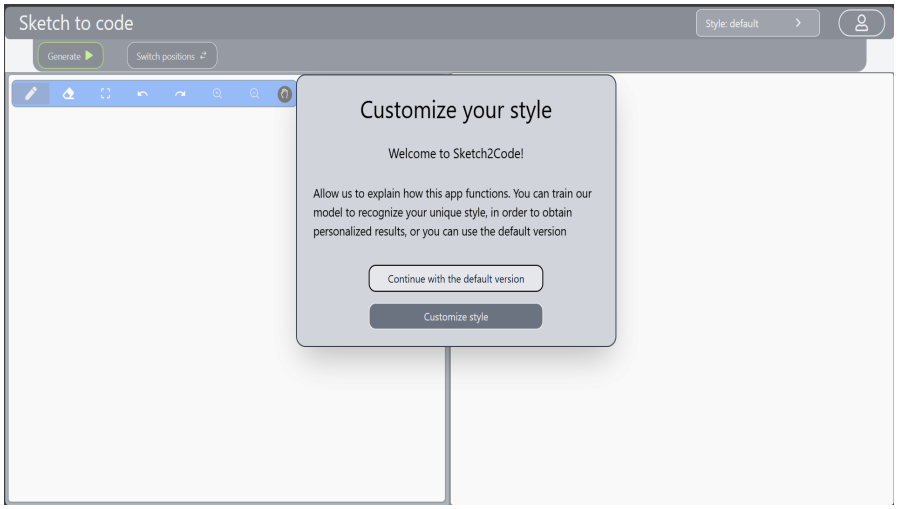

Figura 4.12: PopUp iniziale per i nuovi utenti

fine-tuning risulti efficace è necessario che lo stesso componente venga rappresentato sempre in maniera costante. Tuttavia, distinguere visivamente i diversi componenti non è sempre immediato: per agevolare questo compito è stato implementato un sistema di bounding boxes che evidenzia le aree in cui l'utente deve collocare ciascun componente. Inoltre, i riquadri appaiono con differenti colori in base alla tipologia di componente, semplificandone l'identificazione.

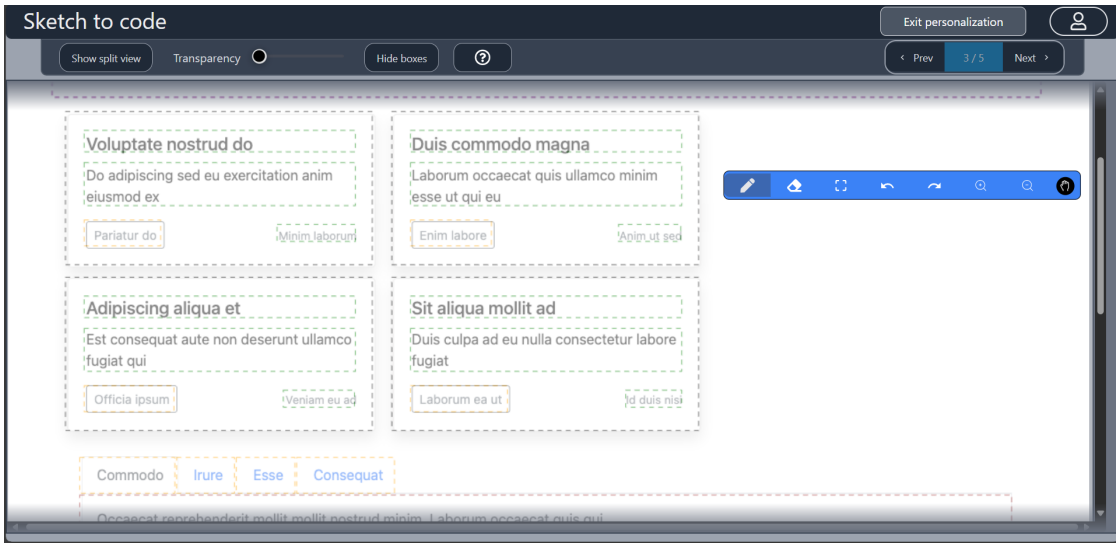

Figura 4.13: Bounding boxes di colori diversi per componenti diversi

Come illustrato in Figura 4.13, i colori associati alle bounding boxes rimangono

gli stessi per ciascun tipo di componente. L'utente può attivare o disattivare la visualizzazione di queste guide tramite l'apposito pulsante `Hide boxes / Show boxes` posizionato nella barra superiore. È inoltre possibile navigare tra diverse interfacce, così da verificare la coerenza del proprio stile di disegno in tutti gli sketch. Nelle dimensioni le interfacce sono adattate allo schermo in larghezza, per una migliore visualizzazione, quindi è possibile scrollare in alto e in basso il canva di disegno. Il numero di sketch richiesti per la creazione di un nuovo stile è fissato a cinque. Questi sketch, insieme al codice corrispondente, costituiscono il dataset necessario per il processo di fine-tuning. Una volta completato il disegno dei cinque esempi, all'utente viene richiesto di assegnare un nome allo stile creato. Infine, gli sketch vengono caricati su S3 e, tramite API Gateway, viene invocata la Lambda Function che si occupa di comunicare con l'istanza SageMaker per avviare il processo di fine-tuning.

4.2.4 AWS Amplify

Prima di passare ad analizzare i servizi backend utilizzati e all'analisi delle due meccaniche principali, cioè la generazione di codice e la personalizzazione, analizziamo brevemente alcune funzioni realizzate tramite AWS Amplify. Amplify è una libreria lato frontend che contiene un insieme di strumenti e servizi offerti da Amazon Web Services che semplifica il collegamento con i servizi cloud backend. È stata introdotta per gestire la comunicazione con il servizio S3 e per semplificare le seguenti funzioni:

Autenticazione

La gestione dell'autenticazione utente, quindi dei login/logout e della sicurezza, è stata gestita utilizzando il servizio `Amplify Auth`. Sia la gestione degli utenti e delle rispettive password sia il database in cui sono contenute, sono tutte completamente gestite dal servizio Amplify Auth basato sul sistema `Amazon Cognito`. Quindi non è stato necessario creare un database apposito per la sicurezza in cui salvare utenti e hash delle password.

Database

È necessario che ogni utente che effettua il login venga registrato in un database, la cui funzione è mantenere una lista dei path in cui sono salvati su S3 gli adapters dei suoi stili. Per fare questo è stato creato un database DynamoDB, generato usando il servizio Amplify GraphQL, che fornisce anche le API per interagire con esso. Appena un utente effettua il login, viene interrogato il database per controllare se quell'utente sta visitando l'app per la prima volta, se è gia registrato o se ha già degli stili salvati. Il database ha come chiave primaria l'username dell'utente

e possiede un campo in cui è presente la lista dei path S3 relativi ai parametri. Se un utente ha già creato uno stile allora avrà una directory su S3 chiamata con il suo username, e in cui ci sarà una sottodirectory con il nome del suo stile, in cui sono salvati i parametri del finetuning associato a quello stile. Il path di questi parametri dentro S3, è invece salvato parallelamente nel database. Questa operazione è effettuata alla fine del finetuning. Nel caso in cui l'utente non abbia ancora creato alcuno stile, il campo nel database rimane vuoto. In caso l'utente abbia già stili salvati, essi verranno recuperati dal database e appariranno nel menù dropdown in alto a destra. Se invece l'utente non appare nel database, questo significa che ha effettuato l'accesso per la prima volta. In questo caso l'utente viene aggiunto al DB, e vengono visualizzati dei PopUp di benvenuto in cui viene brevemente spiegato il funzionamento dell'app.

Storage

Il collegamento tra frontend e S3 avviene tramite il servizio Amplify storage, con cui è possibile caricare, scaricare e leggere file su un determinato bucket S3. Il bucket poi, con gli adeguati permessi può interagire con altri servizi AWS.

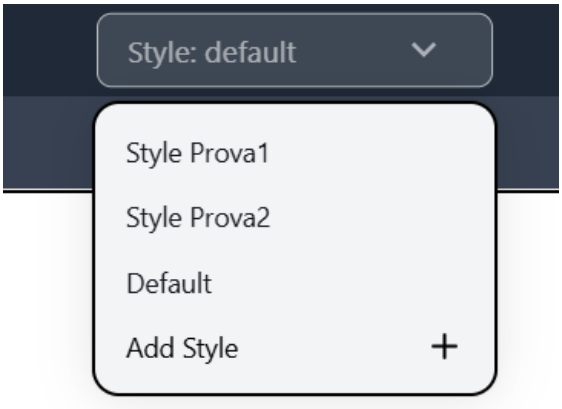

Figura 4.14: Stili di disegno salvati

4.3 Servizi AWS utilizzati

4.3.1 SageMaker: Hosting del modello

Scelta del notebook

Il servizio offre l'accesso a diversi tipi di notebook Jupyter, ottimizzati per l'utilizzo di modelli di IA, e completamente gestiti, il che significa che rispetto a un Jupyter Notebook standard AWS si occupa dell'intera gestione dell'istanza di calcolo, della sicurezza e dell'installazione del software e dei pacchetti, senza che l'utente debba preoccuparsi di configurare server o applicare patch. Inoltre come già detto sono profondamente integrati con il resto dell'ecosistema AWS. Possono accedere in modo sicuro ai dati archiviati in Amazon S3 e interagire direttamente con altri servizi SageMaker. Questo nel nostro caso può semplificare molto il lavoro. I diversi tipi di istanze notebook dipendono dalle risorse disponibili, nello specifico da quante CPU e GPU sono utilizzate, insieme alla quantità di RAM disponibile per entrambe. A seconda del tipo di istanza, queste presentano un costo basato su tariffe di pagamento a consumo. È stato necessario quindi trovare un compromesso tra prestazioni e costo. Dopo aver testato vari notebook la scelta è ricaduta sull'istanza ml.g4dn.xlarge, che comprende 4 CPU e 1 GPU, con 16 Gb di memoria ram ciascuna. L'uso di una sola GPU è stato ritenuto sufficiente per lo scopo.

Modello utilizzato

Il modello utilizzato nel sistema è una versione pre-addestrata di un modello sketch to code, basata sull'architettura `pix2struct-base`. Quest'ultima è progettata per compiti di *visual language understanding*, ovvero la capacità di generare testo a partire da immagini che combinano informazioni visive e linguistiche. In particolare, il modello è costituito da un'architettura encoder–decoder, dove l'encoder elabora l'immagine in input e il decoder produce la sequenza testuale corrispondente. Nel caso d'uso proposto, le immagini di input corrispondono agli sketch delle interfacce utente, mentre l'output testuale generato dal decoder è rappresentato dal codice HTML relativo all'interfaccia disegnata. La variante `base` del modello, adottata in questo progetto, presenta circa 282 milioni di parametri, suddivisi in 12 strati di encoder e 12 strati di decoder. Pur offrendo prestazioni inferiori rispetto alla versione `large`, questa configurazione risulta più leggera e veloce, caratteristiche che la rendono più adatta a un contesto applicativo web, dove è fondamentale garantire tempi di generazione contenuti e un consumo ridotto di risorse computazionali.

Utilizzo

Il notebook viene utilizzato come ambiente di esecuzione per gli script che gestiscono i processi di generazione del codice e di fine-tuning del modello. Il nucleo operativo del sistema è costituito da due script principali in Bash, ognuno dei quali si occupa di uno dei due compiti. In particolare, ciascuno di questi script richiama una serie di script di setup dedicati alla preparazione del file system del notebook (ad esempio la creazione delle directory necessarie e la gestione dei percorsi di input e output), per poi eseguire lo script Python contenente il codice vero e proprio: nel primo caso quello relativo al fine-tuning, e nel secondo quello dedicato alla generazione del codice HTML. Gli input necessari all'elaborazione vengono copiati da Amazon S3 nel file system locale del notebook, all'interno di directory predefinite che gli script andranno a leggere per avviare il processo. La stessa logica è stata applicata alla gestione dei risultati prodotti: gli output vengono prima salvati localmente nel file system del notebook, quindi copiati nuovamente su Amazon S3, da dove il frontend può successivamente recuperarli.

4.3.2 Applicazione del fine-tuning leggero

Il processo di fine-tuning è notoriamente caratterizzato da un elevato consumo di tempo e risorse computazionali, caratteristiche che mal si conciliano con le esigenze di un'applicazione web che richiede elaborazioni rapide e tempi di risposta contenuti. Per questo, la personalizzazione del modello è stata implementata attraverso una tecnica di parameter-efficient fine-tuning, nello specifico con l'approccio LoRA (Low Rank Adaptation). Come discusso nel Capitolo 2, questa metodologia consente di effettuare un fine-tuning estremamente rapido, generando un insieme di parametri aggiuntivi, detti adapters. Tali parametri vengono salvati su S3, seguendo il path `public/Username/style_StyleName/params`, e possono essere semplicemente caricati e aggiunti al modello al momento della generazione con lo stile corrispondente. Durata del finetuning e risultati ottenuti sono due caratteristiche che vanno di pari passo: più epoche dura il training, più il modello impara a riconoscere lo stile dell'utente

Per determinare il numero di epoche di training più adeguato, sono stati condotti test sperimentali mirati a bilanciare tempi di addestramento ed efficacia del modello. Durante questi test è stato utilizzato il portale `WandB` per monitorare l'andamento della *loss*. Dai risultati è emerso che, dopo circa 25 epoche, la *loss* tendeva a stabilizzarsi, con un tempo medio di addestramento pari a circa 1 minuto e 30 secondi. Alla luce di tali osservazioni, si è optato per fissare il numero di epoche a 25, ritenendolo un compromesso adeguato tra velocità e qualità del fine-tuning.

4.3.3 API Gateway e Lambda Function: trigger

La comunicazione all'interno del sistema, come descritto in precedenza, può essere distinta in due tipologie principali:

- la comunicazione relativa al passaggio dei dati (immagini, sketch e codice)
- la comunicazione relativa al trigger delle operazioni di generazione del codice e di fine-tuning da parte del client.

Mentre la gestione del trasferimento dei dati verrà approfondita nella sezione successiva, in questa parte si analizza il meccanismo di comunicazione tra il frontend e l'istanza notebook.

Come primo passo è stata creata un'`API` di tipo REST tramite il servizio API Gateway di AWS. All'interno di essa sono stati definiti due metodi `POST`, uno per ogni lambda function, che consentono l'invio di dati strutturati in formato JSON all'interno del body della richiesta. Una caratteristica fondamentale di API Gateway è la possibilità di collegare direttamente una `Lambda Function` come trigger, la quale viene eseguita automaticamente al momento della richiesta e riceve i dati trasmessi dal frontend. Il passo successivo è stato quindi la creazione delle funzioni Lambda.

Per una maggiore chiarezza nella separazione delle responsabilità, sono state sviluppate due diverse Lambda Functions: una dedicata alla generazione del codice e l'altra al fine-tuning personalizzato. Ciascuna di esse contiene sia il codice necessario per stabilire la connessione con il notebook, sia le istruzioni specifiche per eseguire il rispettivo compito. La comunicazione con il notebook non è stata implementata mediante comandi built-in di AWS, ma attraverso il protocollo `WebSocket`. Questa scelta è stata dettata dall'esigenza di adottare un approccio più flessibile e personalizzabile, che consentisse di interagire direttamente con il file system del notebook (ad esempio per salvare temporaneamente i parametri del fine-tuning e copiarli successivamente su S3), nonché di eseguire più script in un ordine specifico. Tale livello di controllo non sarebbe stato ottenibile con metodi di comunicazione standard. A tal fine, utilizzando la libreria `websocket`, la funzione Lambda apre una connessione verso il notebook e può eseguire comandi da linea di comando in modo diretto.

Una volta stabilita la connessione, la funzione Lambda legge l'username dell'utente e lo stile di disegno associato dal body della richiesta ricevuta tramite API Gateway e invia al notebook una sequenza di comandi personalizzati, dopo aver inviato il comando per installare tutte le dipendenze necessarie. Tali comandi servono per copiare da S3 i dati necessari - che possono consistere in un singolo sketch nel caso della generazione di codice -, gli adapters relativi allo stile nel caso di generazione personalizzata oppure i cinque sketch personalizzati nel caso di fine-tuning. Successivamente, la Lambda invia il comando per eseguire lo script

```
sm_client = boto3.client('sagemaker')
notebook_instance_name = 'Sketch2codeGPU'
url = sm_client.create_presigned_notebook_instance_url(NotebookInstanceName=notebook_instance_name)['AuthorizedUrl']
print(url)


url_tokens = url.split('/')
http_proto = url_tokens[0]
http_hn = url_tokens[2].split('?')[0].split('#')[0]
print(http_hn)

s = requests.Session()
r = s.get(url)
cookies = "; ".join(key + "=" + value for key, value in s.cookies.items())

ws = websocket.create_connection(

    "wss://{}/terminals/websocket/1".format(http_hn),
    cookie=cookies,
    host=http_hn,
    origin=http_proto + "//" + http_hn
)
```

Figura 4.15: Codice nella lambda per interfacciarsi con il notebook

corrispondente al compito richiesto, seguito da quelli necessari a caricare i risultati su S3. Infine, vengono trasmessi i comandi per eliminare i file temporanei generati sul file system del notebook. I comandi vengono inviati in un'unica stringa, separati dai caratteri ˚, che simulano la pressione del tasto invio. Una volta inviati i comandi, la funzione Lambda non attende il completamento dell'elaborazione. Infatti, sebbene la connessione `WebSocket` sia tecnicamente bidirezionale, il notebook non può utilizzare questo canale per restituire dati alla Lambda. Di conseguenza, appena invia la sequenza di comandi, la funzione Lambda termina la propria esecuzione, restituendo al client un valore booleano che indica semplicemente l'avvio dell'elaborazione da parte del modello.

Lambda layers

È importante precisare il contesto operativo e le implicazioni architetturali derivanti dall'utilizzo delle Lambda Functions. Queste funzioni sono state progettate per l'esecuzione di codice leggero e transitorio, ottimizzato per la rapidità di esecuzione e l'efficienza. Il loro ruolo primario nel serverless consiste nell'interfacciare servizi diversi e nell'inoltro di richieste. Nonostante sia tecnicamente possibile estendere il tempo massimo di esecuzione di una Lambda fino a un limite di 15 minuti, esse rimangono inadatte all'esecuzione di carichi di lavoro computazionali intensivi o processi prolungati. Il serverless disincentiva l'utilizzo delle Lambda per attività che richiedono risorse consistenti e persistenti. Un'altra limitazione significativa riguarda la gestione delle dipendenze e delle librerie esterne. L'ambiente di esecuzione serverless non consente l'installazione di pacchetti e librerie tramite metodi standard di gestione dei package (e.g., pip, npm) direttamente all'interno della funzione. Per superare questa restrizione, si ricorre all'uso dei Lambda Layers. Un Layer rappresenta un meccanismo che permette di aggiungere pacchetti esterni, chiamati

layer, contenenti librerie, moduli e codice riutilizzabile, all'ambiente di esecuzione della funzione, senza modificarne il deployment package principale. Altro concetto importante è il riutilizzo: un Layer può essere condiviso da funzioni Lambda multiple, e ogni Lambda Function può integrare un massimo di cinque Layers associati. Esistono Layers pubblici, già pre-forniti dal provider cloud (come gli SDK di AWS o librerie di uso comune), oppure è possibile ricorrere alla creazione di Layers personalizzati per includere librerie specifiche per il progetto. Nella nostra implementazione, l'adozione dei Lambda Layers è stata usata per l'installazione della libreria websocket.

4.3.4 S3: storage dei parametri e canale per i dati

Dopo l'esecuzione del processo di fine-tuning, risulta fondamentale preservare i parametri aggiornati del modello al fine di poterli successivamente ricaricare ed utilizzare per la generazione di codice personalizzato. A tal proposito, si è scelto di adottare Amazon S3 come sistema di archiviazione centralizzato e scalabile. Prima di tutto è stato creato un bucket S3 e sono stati forniti al ruolo collegato al bucket tutti i permessi necessari per interfacciare il bucket all'istanza notebook di SageMaker. All'interno del bucket S3 per ciascun utente viene creata una directory dedicata; in essa vengono salvati i parametri derivanti dal fine-tuning, organizzati in sottocartelle distinte, una per ogni stile di disegno creato. Questa struttura consente di mantenere separati e facilmente recuperabili i diversi set di personalizzazioni.

L'utilizzo di S3 si estende, inoltre, allo scambio dei dati necessari al funzionamento del sistema. Sia gli sketch destinati alla generazione del codice sia quelli impiegati per il fine-tuning vengono caricati dal frontend direttamente su S3, grazie al servizio **Amplify Storage**. Successivamente, tali file vengono copiati nell'ambiente locale del notebook per consentirne l'elaborazione. Lo stesso flusso viene adottato in direzione opposta: al termine dell'elaborazione, quando il codice generato o i parametri risultanti dal fine-tuning vengono caricati su S3, non esistendo un meccanismo diretto per notificare tale evento dal notebook al frontend, questo adotta una strategia basata sull'attesa attiva, o polling, in cui disattiva tutte le funzioni per l'utente e fa partire una progress bar. Durante questa fase, l'applicazione effettua periodicamente dei controlli sul bucket S3 per verificare la presenza dei risultati generati. I periodi medi di attesa tra un poll e l'altro sono stati così definiti:

- circa 30 secondi per il **fine-tuning**. 30 secondi è una scelta arbitraria, data dal fatto che mediamente il finetuning dura di più di una normale generazione di codice (di solito intorno ai 4 minuti), ed è un operazione che si esegue molte meno volte. Per questo motivo si è scelto un intervallo più ampio. Nel caso peggiore, l'utente aspetterà 30 secondi in più dei 4 minuti medi. Non appena i nuovi parametri vengono caricati, il frontend aggiorna il database, aggiungendo al record dell'utente il percorso S3 dei parametri generati

- circa 10 secondi per la **generazione di codice**. L'intervallo è stato scelto più corto, perchè la generazione solitamente impiega meno di 2 minuti. Rispetto al finetuning, l'utente si aspetta di attendere meno per ottenere il codice, perciò nel caso peggiore l'attesa sarà di 10 secondi in più rispetto a quando il codice è effettivamente pronto. Una volta che il codice è disponibile su S3, il frontend lo recupera e lo carica all'interno dell' `iframe` per consentirne la visualizzazione immediata.

```
3  const UploadSketchesOnS3 = async (imageBLOB, path) =>{
4      try {
5      const result = await uploadData({
6          path: path,
7          data: imageBLOB,
8          options: {}
9      }).result;
10     } catch (error) {
11     console.log('Error : ', error);
12     return false;
13     }
14     return true;
15 }
```

Figura 4.16: Codice dell'API per caricare uno sketch su S3

```
35  const downloadHTML = async (path) =>{
36      try {
37      const downloadResult = await downloadData({
38          path: path,
39      }).result;
40      const text = await downloadResult.body.text();
41      console.log('HTML download succeed: ', text);
42      return text;
43      } catch (error) {
44      console.log('Error : ', error);
45      }
46  }
```

Figura 4.17: Codice dell'API per scaricare da S3 il codice HTML generato

Capitolo 5

Sperimentazione e risultati

Il capitolo seguente parla del processo di verifica che è seguito all'implementazione. Prosegue poi con l'analisi dei risultati conseguiti e dell'analisi di punti di forza e criticità del sistema. Tali risultati costituiranno la base per valutare in che misura sono state esaudite le specifiche iniziali, distinguendo tra obiettivi raggiunti in toto, quelli con spazio di affinamento e quelli che, in alcuni casi, non hanno trovato piena realizzazione.

5.1 Setup dei test

L'applicazione sviluppata presenta una chiara vocazione applicativa rivolta principalmente ai web designer. Per questo motivo, la selezione dei candidati destinati alla fase di test è stata condotta seguendo linee guida definite. Come più volte ribadito nel corso di questa tesi, il sistema è stato progettato per essere utilizzabile anche da utenti con competenze limitate nel campo del web design, ma non da persone completamente prive di conoscenze in materia. Pertanto, è stato stabilito come requisito minimo che i candidati avessero sostenuto almeno un esame universitario inerente alla programmazione web o al web design, anche di livello introduttivo. Parallelamente, si è cercato di costruire un campione eterogeneo, includendo sia studenti sia professionisti del settore dello sviluppo software, con diversi gradi di esperienza. Tale scelta è stata dettata dalla necessità di ottenere un quadro rappresentativo di tutte le possibili categorie di utenti che potrebbero utilizzare il sistema. Il gruppo finale dei candidati selezionati per la fase di testing è risultato così composto:

- 5 studenti universitari nel settore dell'ingegneria, che hanno sostenuto almeno un esame in programmazione o web design;
- 3 junior developer;

- 2 senior developer.

Il processo di selezione è stato condotto tramite canali social, attraverso i quali gli interessati potevano manifestare la propria disponibilità. La prenotazione agli slot temporali previsti per i test è avvenuta mediante la compilazione di un foglio Excel condiviso o, in alternativa, tramite comunicazione diretta. Gli esperimenti sono stati condotti in presenza ove possibile. In caso di impossibilità nello svolgere il test in presenza, sono stati svolti a distanza tramite il supporto dell'applicazione AnyDesk, e in videoconferenza. Il test prevede che il candidato esegua tre task:

- **Task 1**: Il candidato deve disegnare nella sezione di disegno uno sketch di un interfaccia a piacere, e deve trasformarla in un prototipo funzionante in HTML generando il codice

- **Task 2**: Il candidato deve personalizzare il sistema, creando uno stile di disegno

- **Task 3**: Il candidato deve disegnare un'altro sketch a piacere, e generare il prototipo corrispondente, ma utilizzando lo stile che ha appena creato

I tre task descritti rappresentano il nucleo operativo del protocollo di test. Per tutta la durata della sessione, ai candidati è stato richiesto di adottare la metodologia del think aloud, esprimendo ad alta voce i propri ragionamenti, dubbi e osservazioni durante l'esecuzione dei compiti. Questo approccio consente di raccogliere dati qualitativi utili a comprendere non solo le azioni svolte, ma anche i processi cognitivi che hanno guidato le decisioni dell'utente. Ciascun partecipante ha dovuto portare a termine i task in completa autonomia, senza ricevere aiuti esterni, al fine di evitare possibili bias che avrebbero potuto compromettere l'oggettività dei risultati. Tuttavia, è stato previsto un ruolo di facilitatore, che ha consentito di intervenire in situazioni di blocco totale o di difficoltà insormontabili al fine di riuscire a completare ugualmente il test. Tali interventi sono stati accuratamente annotati per essere presi in considerazione nell'analisi dei dati raccolti. Durante le sessioni di test, lo schermo del computer è stato registrato, e sono stati archiviati sia gli sketch realizzati dai candidati, sia i prototipi generati dal sistema, così da garantire una base completa e tracciabile per l'analisi successiva.

Metriche di valutazione

Data la valutazione del progetto orientata all'esperienza d'uso del sistema interattivo, sono state impiegate metriche standardizzate come il NASA-TLX, il System Usability Scale (SUS) score e il Creativity Support Index (CLI) score.

NASA-TLX

Il NASA-TLX, sviluppato dalla NASA, consente di misurare il carico di lavoro percepito attraverso sei dimensioni, quali domanda mentale, domanda fisica, domanda temporale, prestazione, sforzo e frustrazione, fornendo così un quadro multidimensionale della fatica cognitiva e operativa. Nello specifico le domande sono:

- *Richiesta Mentale:* Quanto sforzo cognitivo e percettivo è stato richiesto? Il compito era facile o difficile, semplice o complesso?
- *Richiesta Fisica:* Quanto sforzo fisico è stato richiesto? Il compito era leggero o faticoso?
- *Richiesta Temporale:* Quanto senso di urgenza e pressione del tempo è stato percepito? Il ritmo del compito era lento o frenetico?
- *Performance:*Quanto successo pensi di aver avuto nel raggiungere gli obiettivi del compito? Quanto sei soddisfatto del tuo livello di prestazione?
- *Sforzo:* Quanto hai dovuto lavorare sodo (mentalmente e fisicamente) per raggiungere il tuo livello di prestazione?
- *Frustrazione:* Quanto ti sei sentito insicuro, scoraggiato, stressato o, al contrario, sicuro e soddisfatto durante il compito?

Il punteggio assegnabile dal candidato per ogni domanda va da 1 a 100, a intervalli di 5. Poichè la valutazione è relativa al carico di lavoro, punteggi più bassi rappresentano un carico più leggero. Solitamente questo punteggio si assegna tracciando un segno su una linea, con estremi appunto 1 e 100, oppure in modo diretto assegnando il valore esatto. In questo caso si è utilizzato il metodo diretto. Per quanto riguarda il calcolo del punteggio finale, si è calcolata semplicemente la media tra i punteggi. Questo metodo prende il nome di *RAW NASA-TlX* e mantiene un'affidabilità dei punteggi più alta.

CSI Score

Il Creativity Support Index è uno strumento di valutazione concepito per misurare in che misura uno strumento o un sistema tecnologico supporta o ostacola il lavoro creativo di un utente, misurando la percezione soggettiva dell'utente riguardo al supporto fornito dallo strumento durante un compito creativo. È un test composto da 5 domande:

- *Esplorazione:* In che misura lo strumento ti ha permesso di esplorare nuove idee, concetti o soluzioni?

- *Espressività:* Quanto facilmente sei riuscito a tradurre le tue idee in un prodotto o output con questo strumento?
- *Immersione:* Quanto sei stato in grado di immergerti completamente nel compito senza distrazioni o interruzioni causate dallo strumento?
- *Piacere:* Quanto ti sei divertito a usare lo strumento per il compito creativo?
- *Risultati che valgono lo sforzo impiegato:* Hai sentito che lo sforzo che hai messo nel compito ha prodotto risultati che valevano la pena?

Anche qui il punteggio assegnabile va da 1 a 100, con i punteggi più vicini al 100 che indicano un maggior supporto da parte del tool alla creatività.

SUS Score

Il SUS score, invece, è uno strumento per valutare l'usabilità complessiva di un sistema tramite un questionario da dieci domande, che si basa sulla percezione di soddisfazione e usabilità dell'utente durante l'utilizzo del sistema. Le risposte vanno date in base a quanto si è d'accordo con la domanda, in un range da 1 a 5, in cui 1 sta a indicare un forte disaccordo, e 5 che si è fortemente d'accordo. Le domande sono le seguenti:

- *Penso che utilizzerei questo sistema frequentemente.*
- *Ho trovato il sistema inutilmente complesso.*
- *Penso che il sistema fosse facile da usare.*
- *Penso che avrei bisogno del supporto di un tecnico per poter usare questo sistema.*
- *Ho trovato le varie funzioni di questo sistema ben integrate.*
- *Penso che ci siano troppe incoerenze in questo sistema.*
- *Credo che la maggior parte delle persone imparerebbe a usare questo sistema molto rapidamente.*
- *Ho trovato il sistema molto macchinoso da usare.*
- *Mi sentivo molto sicuro nell'usare il sistema.*
- *Ho dovuto imparare (o chiedere) molte cose prima di poter iniziare a usare questo sistema.*

Il punteggio del *System Usability Scale* si ottiene attraverso una formula: le domande dispari (Q_1, Q_3, Q_5, Q_7, Q_9) hanno punteggio positivo, mentre le pari ($Q_2, Q_4, Q_6, Q_8, Q_{10}$) sono invertite. Il punteggio parziale di ciascun item è calcolato come:

$$S_i = \begin{cases} Q_i - 1, & \text{se } i \text{ è dispari} \\ 5 - Q_i, & \text{se } i \text{ è pari} \end{cases}$$

Il punteggio complessivo (SUS) è quindi dato da:

$$SUS = \left(\sum_{i=1}^{10} S_i \right) \times 2.5$$

Il risultato finale è espresso su una scala da 0 a 100, dove valori superiori a 68 indicano un livello di usabilità superiore alla media.

Descrizione degli esperimenti condotti

La struttura del test è stata articolata in due macro-fasi distinte. La prima fase ha riguardato l'interazione diretta dell'utente con l'applicazione, durante la quale il partecipante ha potuto sperimentarne concretamente le funzionalità utilizzando un computer dotato di browser, mouse e tastiera. Questa fase ha permesso di osservare in modo diretto il comportamento dell'utente e le eventuali difficoltà riscontrate durante l'utilizzo del sistema. La seconda fase è stata invece dedicata alla raccolta di informazioni e feedback soggettivi, mediante la compilazione dei questionari e una sessione di domande aperte relative all'esperienza appena conclusa.

In apertura, al candidato viene fornita un'introduzione generale, nella quale vengono illustrati il contesto di sviluppo dell'applicazione, i suoi obiettivi principali e il target funzionale a cui essa è rivolta. Nella stessa fase vengono inoltre presentati in maniera sintetica i tre task che costituiranno il cuore della prova. Successivamente, si procede con l'avvio della registrazione dello schermo e viene ribadito al candidato il primo task da svolgere, che segna l'inizio effettivo del test. A questo punto il candidato inzia il test, navigando nell'app e cercando autonomamente gli strumenti per completare il task. Intanto vengono presi appunti sul comportamento dell'utente durante l'utilizzo del sistema, al fine di trovare eventuali elementi non chiari. Al completamento del primo task, vengono salvate le schermate delle interfacce generate e si passa alla spiegazione del secondo compito. Anche qui l'utente procede autonomamente, mentre vengono presi appunti. Una volta che l'utente ha creato lo stile personalizzato, viene introdotto il terzo task, che conclude la fase operativa. Terminato quest'ultimo, vengono salvate nuovamente le interfacce generate, così da poterle analizzare in seguito per misurare la qualità del sistema in questo senso.

Alla fase di test pratico dell'applicazione è seguita la sessione di valutazione mediante i questionari NASA-TLX, SUS Score e CSI. Infine, il percorso si è concluso con una fase di debriefing, durante la quale sono state rivolte domande aperte al candidato. L'obiettivo è stato quello di approfondire eventuali punti non chiari dell'applicazione, raccogliere feedback sugli aspetti più apprezzati e discutere

criticità emerse. Le domande sono state personalizzate in base all'andamento del test specifico, così da ottenere osservazioni contestuali e pertinenti all'esperienza di ciascun utente.

5.2 Analisi dei risultati

Una volta raccolti tutti i dati, si è passati all'analisi di questi identificando i punti di forza del sistema, i requisiti soddisfatti insieme a quelli non soddisfatti e a eventuali punti critici.

5.2.1 Funzionalità

Le funzionalità implementate nel sistema risultano coerenti con i requisiti funzionali definiti nella fase di analisi e rispondono pienamente alle esigenze del contesto di riferimento. Dal confronto tra i risultati ottenuti e gli obiettivi progettuali, emerge infatti che tutti i comportamenti attesi sono stati realizzati con successo. Il sistema consente all'utente di disegnare a mano libera in modo fluido e senza difficoltà, offrendo strumenti adeguati per rendere l'esperienza di disegno intuitiva e non frustrante, nonostante l'eventuale limitazione derivante dall'uso del mouse come periferica di input. Inoltre, l'applicazione è in grado di convertire lo sketch disegnato in una vera interfaccia web funzionante, generando automaticamente il codice corrispondente, e di visualizzare questa interfaccia per permettere all'utente di confrontarla con lo sketch iniziale in modo diretto. L'utente ha anche la possibilità di addestrare il sistema sul proprio stile di disegno, creando uno o più stili personalizzati che possono essere selezionati liberamente durante le successive generazioni di interfacce. L'applicazione offre la possibilità di gestire differenti layout visivi, adattandosi alle preferenze dell'utente sia nella sezione di generazione sia in quella di personalizzazione, garantendo così un'esperienza d'uso flessibile e personalizzabile.

Nonostante le funzionalità implementate riescano a soddisfare i requisiti progettuali, il sistema presenta ancora alcune criticità. La principale limitazione dell'applicazione, dal punto di vista funzionale, riguarda la qualità del codice generato dal modello. Le interfacce prodotte automaticamente risultano, nella maggior parte dei casi, non pienamente conformi alle aspettative dell'utente in termini di somiglianza visiva con lo sketch originale e di qualità estetica complessiva. In diversi casi, infatti, il design dell'interfaccia generata si discosta in modo significativo dall'input di partenza, perdendo parte della coerenza strutturale. In altri casi ancora, il modello produce output fortemente semplificati o del tutto errati, evidenza del fatto che non è riuscito a interpretare correttamente la struttura dello sketch.

Dopo la fase di personalizzazione, la generazione dell'interfaccia utilizzando lo stile personale mostra, in alcuni casi, lievi miglioramenti rispetto alla generazione standard. In particolare, alcuni componenti dell'interfaccia che in precedenza non venivano riconosciuti correttamente vengono, dopo la personalizzazione, identificati e generati in modo più accurato. Tuttavia, permane il problema relativo al layout complessivo dell'interfaccia. Il modello, infatti, non riesce a interpretare correttamente la struttura dello sketch, restituendo un'interfaccia che, pur contenendo alcuni elementi riconosciuti correttamente, risulta errata nella disposizione spaziale o nel numero dei componenti.

5.2.2 Usabilità del sistema

Dal punto di vista dell'usabilità, il sistema ha evidenziato prestazioni complessivamente molto positive. I risultati emersi dai questionari somministrati agli utenti indicano che l'applicazione è percepita, in media, come semplice, intuitiva e di facile apprendimento. Analizzando i punteggi ottenuti nelle diverse metriche di valutazione, si osserva un andamento coerente tra tutte e tre le scale utilizzate: i valori più elevati sono stati registrati tra i partecipanti con maggiore esperienza nel web design, un risultato prevedibile in quanto tale gruppo presenta una più ampia familiarità sia con il contesto applicativo in cui il sistema si inserisce, sia con l'utilizzo di strumenti di prototipazione e design interattivo. Seguono i junior developer, che mostrano risultati leggermente inferiori ma comunque positivi, e infine gli studenti, che pur registrando punteggi più bassi, hanno comunque dimostrato di poter utilizzare il sistema con successo. Questo andamento suggerisce che l'esperienza pregressa nel settore rappresenta un fattore facilitante nell'interazione con l'applicazione, ma non costituisce una condizione necessaria: il sistema risulta infatti accessibile anche a utenti con conoscenze limitate, confermando così la validità dell'obiettivo di progettare un'interfaccia intuitiva e utilizzabile da un ampio spettro di utenti.

Andando a analizzare i punteggi ottenuti nei singoli questionari notiamo che, per quanto riguarda il NASA-TLX (che misura il carico di lavoro mentale, fisico e temporale), i punteggi medi indicano un livello di impegno complessivo moderato. Le dimensioni legate allo sforzo mentale e alla performance hanno mostrato valori più elevati rispetto a quelle relative alla frustrazione e allo sforzo fisico, suggerendo che l'utilizzo del sistema richiede concentrazione ma non comporta un sovraccarico cognitivo significativo. Il punteggio medio generale si attesta attorno a 35–40 punti, valore che rientra nei limiti di un'interazione fluida e gestibile.

Passando alla valutazione del grado di engagement e supporto fornito all'attività creativa, i risultati mostrano anche qui punteggi mediamente alti, partendo dai 68 punti medi per gli studenti, fino agli 88 punti per i senior developer. Questo suggerisce che gli utenti hanno percepito il sistema come stimolante, in grado di

	NASA-TLX							
Ruolo	mentale	fisica	temporale	perform.	sforzo	frustaz.	P. medio	Media per ruolo
Junior dev	40	20	55	70	30	10	37,50	39,78
Junior dev	60	20	10	90	75	10	44,17	
Junior dev	25	30	45	95	30	1	37,67	
Senior dev	10	1	10	100	10	10	23,50	27,67
Senior dev	40	1	10	90	40	10	31,83	
Studente HCI	50	1	5	50	30	30	27,67	35,37
Studente HCI	30	20	5	65	100	5	37,50	
Studente HCI	60	60	35	85	20	20	46,67	
Studente HCI	70	20	10	75	25	20	36,67	
Studente HCI	20	50	10	70	10	10	28,33	
Punteggio medio per categoria	40,5	22,3	19,5	79	37	12,6		

Figura 5.1: Punteggi Task Load Index

supportare la fase di progettazione e di favorire l'espressione personale nel disegno delle interfacce. I valori più elevati si registrano per i candidati con maggiore esperienza nel web design, a conferma della coerenza tra familiarità con il dominio e percezione di controllo e immersione.

	CSI						
Ruolo	expl.	espress.	immers.	piacere	risultati	P. medio	Media per ruolo
Junior dev	55	75	70	70	60	66	76
Junior dev	80	60	90	75	75	76	
Junior dev	100	70	100	90	70	86	
Senior dev	100	80	100	100	100	96	88,5
Senior dev	90	55	100	90	70	81	
Studente HCI	50	30	70	80	60	58	68,2
Studente HCI	75	55	85	80	70	73	
Studente HCI	35	85	75	100	90	77	
Studente HCI	80	75	85	100	25	73	
Studente HCI	50	40	90	90	30	60	
Puntegggio medio per categoria	71,5	62,5	86,5	87,5	65		

Figura 5.2: Punteggi Creativity Support Index

Infine, il SUS Score, indice finale per valutare la usabilità percepita. La tabella 5.3 mostra una media complessiva di circa 78,7 punti, un risultato che, secondo la scala standard di interpretazione SUS, corrisponde a una valutazione di Good–Excellent. Ciò indica che gli utenti hanno trovato l'applicazione intuitiva, coerente e semplice da utilizzare, indipendentemente dal livello di esperienza. Anche qui si può osservare una discrepanza tra i punteggi rispetto ai ruoli, suggerendo un margine di miglioramento nella chiarezza di alcune funzionalità per gli utenti meno esperti.

	SUS											
Ruolo	1	2	3	4	5	6	7	8	9	10	Punteggio	Media per ruolo
Junior dev	3	2	4	2	3	1	3	1	4	2	72,50	84,17
Junior dev	3	1	4	1	4	1	5	1	5	3	85,00	
Junior dev	5	1	4	1	4	1	5	1	5	1	95,00	
Senior dev	5	1	5	1	5	1	5	1	5	2	97,50	91,25
Senior dev	2	2	5	1	4	1	5	1	5	2	85,00	
Studente HCI	3	3	3	2	4	2	4	1	3	3	65,00	70,50
Studente HCI	2	1	5	1	1	1	5	1	5	3	77,50	
Studente HCI	3	1	5	1	4	2	4	4	4	2	75,00	
Studente HCI	2	2	2	4	3	4	5	4	2	3	42,50	
Studente HCI	3	1	5	1	5	1	5	1	4	1	92,50	
										Media	78,75	

Figura 5.3: Punteggi Sstem Usability Scale

In particolare, l'analisi dei test utente e la successiva fase di debriefing hanno messo in chiaro alcuni punti critici che hanno contribuito ad abbassare la percezione di semplicità nell'utilizzo da parte di utenti meno esperti.

La fase di debriefing, accompagnata da un'analisi dei test focalizzata a cercare i punti più critici per gli utenti nel flusso di lavoro — in particolare quelli in cui si è reso necessario l'intervento diretto del facilitatore — ha messo in evidenza una criticità legata alla semplicità d'uso del sistema. Gli utenti meno esperti, in particolare gli studenti, hanno mostrato difficoltà nel comprendere al primo impatto il funzionamento della fase di personalizzazione. Nonostante la presenza di un tutorial introduttivo, non risultava immediatamente chiaro che le interfacce visualizzate in trasparenza fossero quelle da riprodurre attraverso il disegno. Al contrario, questo elemento visivo ha spesso generato confusione, rallentando l'interazione con il sistema. In alcuni casi, gli utenti hanno impiegato alcuni minuti per comprendere le operazioni da svolgere, mentre in altri è stato necessario un intervento esplicativo da parte del facilitatore. Questa sezione è risultata invece chiara fin da subito solo per i senior developer e per due ulteriori candidati, rispettivamente uno studente e uno sviluppatore junior.

Inoltre, il processo di disegno manuale delle cinque interfacce è stato percepito dagli utenti come la fase più impegnativa dell'intero test. Tale percezione è comprensibile se si considera il contesto sperimentale: nell'ambito di una sessione di test, questa attività può risultare ripetitiva e faticosa. Tuttavia, è importante sottolineare che, in un contesto d'uso reale, questa operazione verrebbe eseguita solo occasionalmente, in proporzione molto ridotta rispetto all'utilizzo complessivo dell'applicazione. Di conseguenza, l'impatto di questa fase risulterebbe quasi trascurabile nell'esperienza d'uso totale del sistema.

Velocità

Le tempistiche relative alle attese durante l'utilizzo dell'app sono essenzialmente due:

- il tempo di attesa per la generazione del codice si assesta mediamente intorno ai 2 minuti
- il tempo di attesa per il completamento del finetuning nella fase di personalizzazione è intorno ai 4 minuti

Durante la fase di debriefing, nessun partecipante ha manifestato insoddisfazione riguardo ai tempi di attesa del sistema, né durante la fase di generazione del codice, né in quella di personalizzazione tramite finetuning. Quando agli utenti è stato chiesto esplicitamente se ritenessero i tempi di elaborazione adeguati, la risposta media è risultata positiva: la maggior parte dei partecipanti ha dichiarato che i tempi di attesa erano pienamente in linea con le proprie aspettative. Questo dato è particolarmente significativo, soprattutto nel caso del finetuning del modello, in quanto grazie alle ottimizzazioni introdotte nel sistema, come l'impiego di tecniche di finetuning leggero (LoRA),i tempi di risposta si sono mantenuti entro soglie accettabili per un utilizzo interattivo. Può quindi considerarsi soddisfatto il requisito funzionale relativo all'ottenimento dei risultati in tempi compatibili con una buona esperienza d'uso.

5.3 Considerazioni finali

Il sistema sviluppato, nell'ottica di realizzare un'applicazione in grado di consentire il disegno a mano libera di interfacce utente e, al tempo stesso, di generare automaticamente il codice corrispondente ha effettivamente raggiunto il risultato. L'applicazione si è dimostrata funzionale, accessibile e coerente con i requisiti definiti in fase di analisi. Gli utenti hanno potuto sperimentare un flusso di lavoro fluido, nel quale la fase creativa del disegno e la fase automatizzata di generazione del codice coesistono in un unico ambiente, riducendo la frammentazione tipica dei processi di prototipazione tradizionali. Questo approccio ha evidenziato come l'integrazione tra tecniche di IA generativa e strumenti di design interattivo possa offrire un reale valore aggiunto nei contesti di sviluppo web. Tuttavia, nonostante il successo architetturale e funzionale del sistema, i risultati sperimentali hanno messo in luce limiti importanti legati alla qualità del codice generato. Tale aspetto è riconducibile principalmente alle prestazioni del modello di intelligenza artificiale adottato. In particolare, il modello ha mostrato difficoltà nel riconoscimento dei componenti e nella disposizione spaziale degli elementi dello sketch, dimostrando gli stessi limiti - analizzati nel capitolo 2 - ritrovati in altri approcci e sistemi che

miravano a trasformare uno sketch in codice. Questa criticità, pur non inficiando la validità del progetto, evidenzia la necessità di un ulteriore miglioramento del modello di generazione, sia attraverso dataset più ampi e diversificati, sia mediante tecniche di fine-tuning più mirate. In prospettiva, l'integrazione di modelli di nuova generazione, addestrati specificamente sul dominio degli sketch di interfacce, potrebbe migliorare sensibilmente la qualità e la fedeltà del codice prodotto. In conclusione, il sistema può essere considerato un esperimento riuscito dal punto di vista progettuale e tecnico, capace di dimostrare la fattibilità e il potenziale dell'integrazione tra disegno libero e generazione automatica di codice. Al contempo, i risultati ottenuti offrono spunti concreti di miglioramento per sviluppi futuri, in particolare nella direzione di una maggiore accuratezza e coerenza semantica nella fase di generazione del codice.

Capitolo 6

Conclusioni

Il lavoro svolto in questa tesi ha rappresentato un percorso di ricerca e sviluppo volto a esplorare le potenzialità dell'intelligenza artificiale applicata alla prototipazione web. L'obiettivo principale è stato quello di realizzare un sistema capace di unire il disegno a mano libera e la generazione automatica di codice, con l'intento di ridurre la distanza tra le fasi di ideazione visiva e di implementazione tecnica di un'interfaccia.

La realizzazione del sistema ha richiesto un processo articolato, che ha combinato progettazione concettuale, sviluppo tecnico e sperimentazione empirica. Nella fase iniziale è stata condotta un'analisi approfondita dei metodi attuali di prototipazione basati sull'IA, evidenziando come la maggior parte degli strumenti oggi disponibili offra solo assistenza parziale e frammentata nella scrittura del codice, senza integrarsi realmente nel flusso di design. Su questa base, si è progettato e implementato un prototipo di web application che consente all'utente di disegnare interfacce a mano libera direttamente nel browser e di ottenere automaticamente il relativo codice HTML, generato da un modello di intelligenza artificiale. La scelta di un'architettura serverless basata su AWS ha permesso di ottenere un sistema scalabile, modulare e facilmente accessibile, in grado di gestire processi complessi come la generazione e il fine-tuning del modello. Il risultato è un'applicazione che integra in un unico ambiente la creatività del disegno e la precisione della generazione automatica, dimostrando la fattibilità tecnica di un approccio di questo tipo.

L'introduzione di un sistema capace di trasformare sketch in codice funzionante rappresenta un cambio di paradigma nel modo di concepire la prototipazione web. Tradizionalmente, il passaggio dal disegno concettuale al codice avviene attraverso una serie di passaggi intermedi — wireframe, mockup, implementazione manuale — che frammentano il processo e ne aumentano i tempi. Il sistema sviluppato, invece, propone una visione più integrata e immediata, in cui la creazione visiva e la realizzazione tecnica diventano parte di un unico flusso continuo. Questa

prospettiva potrebbe avere un impatto significativo nelle pratiche di design digitale contemporanee, fornendo agli sviluppatori e ai designer uno strumento in grado di ridurre il tempo tra l'idea e il prototipo funzionante, favorendo una maggiore iterazione e sperimentazione creativa. Nonostante la qualità del codice generato non permetta di utilizzare il sistema in uno scenario reale, il sistema rappresenta un passo verso una prototipazione semi-automatica, in cui l'intelligenza artificiale funge da assistente creativo e tecnico.

Sviluppi Futuri

I risultati raggiunti aprono numerose direzioni per futuri sviluppi e approfondimenti. In primo luogo, sarà importante migliorare la qualità della generazione del codice, attraverso l'utilizzo di modelli più avanzati e specializzati nella comprensione della struttura visiva degli sketch. Un'ulteriore evoluzione potrebbe riguardare la costruzione di dataset personalizzati, basati su schizzi reali di interfacce e codici corrispondenti, per incrementare la capacità del modello di generalizzare a diversi stili di disegno.

Parallelamente, potranno essere introdotte funzionalità di feedback interattivo, che consentano all'utente di intervenire sul processo di generazione per correggere o guidare il risultato in tempo reale. Dal punto di vista dell'esperienza utente, invece, si potrà lavorare sull'ottimizzazione dell'ambiente di disegno, includendo supporto a dispositivi grafici, livelli multipli e strumenti di editing più sofisticati.

In conclusione, il sistema sviluppato dimostra che l'unione tra disegno libero e generazione automatica di codice non solo è tecnicamente possibile, ma rappresenta anche una direzione promettente per l'evoluzione degli strumenti di design digitale. Pur con i limiti attuali legati alla precisione del modello, il progetto offre un contributo concreto alla ricerca sulla prototipazione assistita da IA, ponendo le basi per sistemi futuri sempre più intuitivi, intelligenti e creativamente collaborativi.

Bibliografia

[1] *L'evoluzion del design UI/UX. Dalla prima interfaccia utente a oggi*. BB's way. URL: https://bbsway.it/magazine/levoluzione-del-design-ui-ux-dalla-prima-interfaccia-utente-a-oggi (cit. a p. 1).

[2] *Figma vs Sketch: Un Confronto tra le Caratteristiche di Due Popolari Strumenti di Progettazione*. Kinsta. 2023. URL: https://kinsta.com/it/blog/figma-vs-sketch/#cos-sketch (visitato il giorno 13/03/2023) (cit. a p. 3).

[3] *Prototyping Tools: migliori strumenti prototipazione per UX Designer*. Talent Garden. 2023. URL: https://blog.talentgarden.com/it/blog/design/migliori-prototyping-tools-ux-designer (visitato il giorno 22/08/2023) (cit. a p. 3).

[4] *Come l'Intelligenza Artificiale sta rivoluzionando il Web Design nel 2025*. Studio 99. 2025. URL: https://www.studio99.sm/come-lintelligenza-artificiale-sta-rivoluzionando-il-web-design-nel-2025/ (visitato il giorno 20/05/2025) (cit. a p. 4).

[5] *Prototipazione siti web con AI*. New Web Solutions. URL: https://www.newebsolutions.com/prototipazione-siti-web-con-ai/ (cit. a p. 4).

[6] *I migliori LLM per la programmazione*. Flow Hunt. Giugno 2025. URL: https://www.flowhunt.io/it/blog/best-llms-for-coding-june-2025/# (visitato il giorno 06/2025) (cit. a p. 7).

[7] Chenglei Si, Zhengyuan Yang, Yanzhe Zhang, Ruibo Liu e Diyi Yang. «Design2Code: How Far Are We From Automating Front-End Engineering?» In: (2024) (cit. a p. 8).

[8] Alexander Robinson. «sketch2code: Generating a website from a paper mockup». In: *University of Bristol* (2019) (cit. a p. 8).

[9] Tony Beltramelli. «pix2code: Generating Code from a Graphical User Interface Screenshot». In: *UIzard Technologies* (2017) (cit. a p. 8).

[10] Yuwen Lu, Alan Leung, Amanda Swearngin, Jeffrey Nichols e Titus Barik. «Misty: UI Prototyping Through Interactive Conceptual Blending». In: (2024) (cit. a p. 9).