

POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Intelligenza Artificiale generativa per lo sviluppo di Applicazioni Angular



Relatore

prof. Paolo Garza
firma del relatore

.....

Candidato

Luca Basso
firma del candidato

.....

Anno Accademico 2023-2024

Ringraziamenti

A mamma, che conosce tutte le mie passioni nel dettaglio (anche se non vorrebbe). Per aver sopportato tutti gli sbuffi durante la stesura di questo lavoro, per avere una soluzione pronta a problemi per me tragici e per essere riuscita a farmi sentire coccolato persino nelle pause pranzo fuori casa.

A papà, uno dei pochi padri a cui interessi davvero sapere "Cos'hai fatto oggi a scuola?". Per le domeniche mattina fra uomini, per le lezioni di guida supplementari senza le quali non avrei ancora una patente e per la sua capacità di ascoltare in silenzio.

A Fiore, capace di portare il sorriso senza parlare. Per le innumerevoli attese dalla finestra e per avere il musetto più bello di tutti.

A nonna Maria, cuoca impareggiabile e fine commentatrice di wrestling. Per i pranzi preparati con amore e per le telefonate del dopo cena, appuntamento quotidiano da quando ne ho memoria.

A Gaia, ottima spotter e studiosa assidua che mi ha ispirato a lavorare quando non ne avevo voglia. Per la sua mentalità aperta in fatto di ristoranti, per aver disegnato degli scalini al momento giusto e per aver ascoltato i Beach Boys troppe volte. Grazie per aver sopportato le mie lamentele attraverso ogni possibile canale di comunicazione.

Ad Ambra, a Nina, a Giulia e a Valerio, fra cui si celano ottime fotografe (spesso non apprezzate a dovere) ed estimatori di manga e distillati alle erbe. Per essere stati degli amici come si deve, per i loro meme su cagnolini, Pokémon e altre cose non nominabili in questo spazio, ma soprattutto per avermi visto leccare più bicchieri di quanti avrebbero voluto.

A Sara, attenta psicologa la cui abilità nel braccio di ferro mi ha spinto verso il bodybuilding. Per la consulenza in questioni informatiche e non, per avermi regalato un pacco di pasta di Topolino e soprannomi quali "macho man" e "cappellaio matto". (Comunque sì: ho visto Star Wars).

Ad Antonio, massimo esperto di JoJo e di cultura pugliese, che non si tira mai indietro quando è ora di un cigno e dimostra per gli amici la stessa cura che ha per le interfacce grafiche. Per i pranzi in aula ospedale, per avermi insegnato cosa sia una graziellona, per i suoi contatori e per aver chiuso un occhio su righe di codice disordinate.

E infine a Matteo, primo uomo nella storia ad aver analizzato le frequenze del rumore di fondo nelle aule P e inesauribile fonte di aneddoti. Per la sua aria da papà tranquillo che ha fatto sembrare meno spaventosi certi esami, per essersi fatto convincere a provare Zelda II e per essere stato l'unico ad apprezzare Trenulețul.

Sommario

Obiettivo di questa tesi è esplorare il ruolo rivoluzionario dell'intelligenza artificiale generativa (Generative AI) nell'ambito dello sviluppo di applicazioni, in particolare di applicazioni Angular. Questo lavoro si propone di semplificare il processo di sviluppo del codice, riducendo il numero di errori umani e il tempo di scrittura, aumentando così la produttività. In particolare ci si propone di realizzare un chatbot scritto in Python che, sfruttando le API di OpenAI, sia in grado di generare applicazioni Angular a partire da prototipi visivi o testuali, realizzabili anche senza particolari competenze di programmazione.

L'indagine inizia con un'introduzione sull'intelligenza artificiale: che introdurrà concetti come machine learning, deep learning e reti neurali, in modo da arrivare alle definizioni di intelligenza artificiale generativa e di modello linguistico di grandi dimensioni, entrambe molto importanti per questo lavoro.

Segue una descrizione dell'attuale stato dell'arte in cui si valutano le prestazioni di alcuni strumenti già a disposizione dei programmatori, ovvero Tabnine e GitHub Copilot

Si passerà poi alla discussione delle principali caratteristiche del framework Angular, la cui struttura a componenti può essere un interessante problema da sottoporre ad un'intelligenza artificiale. La parte introduttiva si conclude con una panoramica sui principali strumenti utilizzati in questo lavoro, ovvero Figma, per i prototipi delle pagine web e LangChain, per includere un LLM nel chatbot.

Nella seconda parte, di carattere più pratico, vengono presentate le principali librerie e integrazioni utilizzate per poi descrivere la struttura del programma Python.

Si descriveranno poi i tentativi che hanno portato alla versione finale del chatbot e i più interessanti risultati ottenuti, nonché le varie migliorie apportate, come l'aggiunta di una sorgente in formato immagine e di un prompt più dettagliato.

La tesi si conclude con osservazioni sui risultati ottenuti e sullo stato attuale dei supporti basati sull'intelligenza artificiale messi a disposizione dei programmatori. Seguono osservazioni su destinazione d'uso del progetto e possibili sviluppi futuri.

Indice

Elenco delle figure	v
1 Intelligenza artificiale: un'introduzione	1
1.1 Definizione di intelligenza artificiale	1
1.2 Intelligenze artificiali a confronto	2
1.2.1 Weak AI e strong AI	2
1.2.2 Machine learning	2
1.2.3 Deep learning e reti neurali	4
1.2.4 I Large Language Models e generative AI	6
1.2.5 Intelligenza artificiale: una tassonomia finale	9
2 Tabnine e GitHub Copilot: una panoramica e confronto fra i due	11
2.1 Tabnine	11
2.2 GitHub Copilot	11
2.3 Valutazione delle prestazioni dei due strumenti	13
2.3.1 Spiegazioni e suggerimenti in linea	13
2.3.2 Generazione di intere funzioni	13
2.3.3 Chat	15
2.3.4 Copilot come aiuto nella stesura di testo	17
3 Introduzione ad Angular	19
3.1 Cenni storici	19
3.2 Principali caratteristiche	20
3.2.1 Componenti	20
3.2.2 Compilazione	20
3.2.3 Data binding	21
3.2.4 Dipendenze	22
3.2.5 Routing	22
3.2.6 Direttive	22
3.2.7 Altre caratteristiche	23
4 Presentazione degli strumenti ausiliari: Figma e LangChain	25
4.1 Figma	25
4.2 Langchain	28

5	Sviluppo di un chatbot in Python	33
5.1	Presentazione ambiente e librerie	33
5.2	Descrizione del codice	34
5.2.1	Inizializzazione	35
5.2.2	Main	35
5.2.3	Funzione di generazione del codice	35
6	Risultati ottenuti e osservazioni	37
6.1	HTML e CSS di singoli componenti	38
6.1.1	Tasto add	38
6.1.2	Navbar	39
6.2	HTML e CSS dell'intera pagina	40
6.3	HTML e CSS con JavaScript	42
6.4	HTML e CSS con AngularJS	42
6.5	App Angular completa	43
7	Osservazioni sui risultati precedenti e conclusioni	47
7.1	Osservazioni	47
7.2	Conclusione	48
	Bibliografia	49

Elenco delle figure

1.1	Esempio di come dall'immagine di un elefante vengano individuati i contorni principali che verranno via via messi insieme per comporre la sagoma dell'elefante.	4
1.2	Rappresentazione schematica di una rete neurale.	5
1.3	Schema che illustra come gli input di un neurone vengano processati per generare l'output.	6
1.4	Elenco completo delle applicazioni dei large language model individuati in "Challenges and Applications of Large Language Models", alcune delle quali sono riportate in questo sottoparagrafo [16] (immagine tratta dall'articolo).	9
1.5	Grafico illustrante la tassonomia di diversi tipi di intelligenze artificiali [17] (immagine tratta dall'articolo citato).	10
2.1	Il logo di tabnine.	12
2.2	Il logo di GitHub Copilot.	12
2.3	Tabnine spiega il codice.	13
2.4	A sinistra il suggerimento di Visual Studio, a destra quello di Tabnine.	14
2.5	Tabnine suggerisce un metodo simile a quello appena scritto.	14
2.6	Tabnine suggerisce un metodo simile a quello appena scritto.	15
2.7	Copilot genera una funzione dietro indicazione dell'utente.	16
2.8	La chat di Tabnine.	16
2.9	Copilot commenta il testo presentatogli (notare come per generare la risposta venga fatto riferimento ai file Python).	17
3.1	Il logo di Angular.	20
3.2	Esempio di un componente al cui interno è annidato un altro componente. Notare come tutti i file abbiano il suffisso ".component".	20
4.1	Il logo di Figma.	26
4.2	Modalità design: è possibile aggiungere componenti di vario genere.	27
4.3	Modalità prototipo: è possibile rappresentare il flusso logico fra schermate tramite frecce.	27
4.4	Modalità codice: è possibile vedere informazioni di base sul CSS del componente selezionato.	28
4.5	Il logo di LangChain.	29
4.6	Esempio di utilizzo del LCEL.	30
5.1	Diagramma delle classi illustrate in questo paragrafo	34
5.2	Diagramma di sequenza che riassume le interazioni descritte in questo paragrafo	35

6.1	Screenshot della pagina generata con Figma che si cerca di replicare	38
6.2	Versione base del prompt di sistema, presa dal sito di LangChain [48].	38
6.3	Alcuni tentativi di replicare il tasto add: notare come né il font, né il contorno né l'uso delle maiuscole siano consistenti fra un risultato e l'altro.	39
6.4	Le prime navbar generate non avevano né le giuste proporzioni né i giusti colori. Nella prima si nota addirittura un tentativo di aggiungere un'immagine che però non era stata richiesta, nella seconda è presente la scritta "Amount", presa da Figma anche se fa parte del form e non della navbar.	39
6.5	Dopo aver chiesto di aggiungere un caret (triangolino rivolto verso il basso che suggerisce all'utente la presenza di un menù a tendina) vicino alla scritta "Manage", il chatbot la aggiunge a tutti i componenti, convinto però di averlo messo soltanto vicino a quello desiderato.	40
6.6	La navbar meglio riuscita, molto fedele all'originale.	40
6.7	I primi output erano così diversi dal risultato atteso da far pensare che l'ia stesse improvvisando. Qui ad esempio "Name" viene interpretato come nome dell'utente, invece che dell'ingrediente.	41
6.8	Anche quando la pagina era graficamente simile all'originale, alcune scritte erano "inventate", invece che prese da Figma (in questo caso i nomi degli ingredienti). Notare come posizione e spaziatura nella navbar continuino ad essere problematici.	41
6.9	Un buon esempio di pagina, ottenuto però dopo numerose iterazioni.	42
6.10	Due estratti di una pagina in cui l'IA ha preferito servirsi di AngularJS: nel primo si nota il link ad AngularJS 1.8.2, nel secondo si può notare la differenza rispetto all'Angular più moderno.	43
6.11	Screenshot in cui l'IA descrive ciò che realmente può desumere dal file Figma. Anche qui si può osservare una risposta inventata e subito dopo ritrattata dalla stessa intelligenza artificiale che l'ha fornita.	44
6.12	Versione finale del prompt, che comprende tutte le direttive che è stato necessario aggiungere nel corso dei vari esperimenti eseguiti	45
6.13	Screenshot che mostra come l'intelligenza artificiale abbia individuato la funzionalità mancante dopo averle fatto genericamente notare che una parte del prompt di sistema fosse stata trascurata.	46

Capitolo 1

Intelligenza artificiale: un'introduzione

Lo scopo di questa tesi è piuttosto pratico e il suo contenuto tecnico, ma un argomento complesso e rivoluzionario come l'intelligenza artificiale merita certamente un'introduzione.

Invece di ridurre l'IA ad una scatola nera da utilizzare per facilitare il lavoro dell'informatico, ne analizzeremo in questo primo capitolo la storia e le diverse tipologie, comprendendo così quanti anni di lavoro e quante discipline diverse si celino dietro ad un fenomeno entrato a far parte del linguaggio comune solo negli ultimi anni.

1.1 Definizione di intelligenza artificiale

Nel fornire una definizione di intelligenza artificiale ci vengono in aiuto due enti autorevoli:

- l'enciclopedia Treccani definisce l'intelligenza artificiale come disciplina che *studia i fondamenti teorici, le metodologie e le tecniche che consentono di progettare sistemi hardware e sistemi di programmi software atti a fornire all'elaboratore elettronico prestazioni che, a un osservatore comune, sembrerebbero essere di pertinenza esclusiva dell'intelligenza umana*[1]
- un documento stilato dall'AI HLEG (AI High-Level Expert Group, gruppo di esperti indipendenti istituito dall'Unione Europea) classifica come IA i *sistemi che mostrano un comportamento intelligente analizzando l'ambiente che li circonda e intraprendendo azioni - con un certo grado di autonomia - per raggiungere obiettivi ben precisi*[2].

Entrambe le fonti proseguono con lunghe dissertazioni. La prima propone una serie di implicazioni etiche riguardo a privacy, affidabilità e rischio di trasmettere bias. Ne è un esempio quanto rilevato dal Federal Reserve Board di Washington, che ha notato come nel 2018 e 2019 i clienti neri avessero il doppio della possibilità di vedersi negato un prestito rispetto ai loro concittadini bianchi, oltre a ricevere tassi di interesse maggiori e punteggi di credito minori: questo pregiudizio razziale si è poi riflettuto nell'AUS, sistema automatizzato di concessione dei prestiti che ha continuato a sfavorire minoranze come quella afroamericana e ispanica.[3]

Riflessioni di questo tipo sono sicuramente stimolanti, ma esulerebbero dallo scopo di questo progetto; molto più interessante nel campo ingegneristico è invece lo spunto fornito dal testo dell'AI HLEG, che prosegue fornendo un'utile precisazione sulla parola "intelligenza". Questo

termine è in effetti piuttosto vago e secondo la commissione di esperti può al momento riferirsi soltanto ad un'intelligenza di tipo razionale: quanto intelligenti sono allora i sistemi sviluppati finora? Proveremo a fornire una risposta a questo interrogativo nel prossimo paragrafo.

1.2 Intelligenze artificiali a confronto

Per comprendere meglio gli argomenti trattati, facciamo un po' di chiarezza sulla terminologia: dopo aver introdotto la differenza tra weak e strong AI, ci concentreremo su termini come machine learning, deep learning, generative AI e large language models. Questi campi di ricerca vengono spesso confusi e i loro nomi utilizzati in modo equivalente ma, come vedremo, esiste una precisa distinzione.

1.2.1 Weak AI e strong AI

La prima distinzione da menzionare è sicuramente quella fra weak AI e strong AI.

La prima, detta anche ANI (Artificial Narrow Intelligence), è allenata per eseguire compiti specifici ed è già applicata in moltissimi ambiti, dalle app come Google Maps per calcolare il percorso migliore ai motori di ricerca per fornire risultati personalizzati per il singolo utente profilato, passando per il riconoscimento di impronte e volti, solo per citarne alcuni [4].

La seconda è anche detta AGI (Artificial General Intelligence) e rimane ancora un obiettivo da raggiungere: con questo termine ci si riferisce infatti non tanto a facoltà quali l'esecuzione di calcoli complessi, ma alla capacità di applicare questo tipo di capacità ad una gran varietà di compiti, dimostrando versatilità e adattabilità. Questo tipo di abilità è al momento attuale più sviluppato in animali come i corvidi [5] e sta quindi dando luogo a nuove definizioni di intelligenza, come quella formulata nel 2007 da Shane Legg e Marcus Hutter, attualmente al lavoro presso DeepMind, laboratorio di Google per la ricerca sull'intelligenza artificiale. Secondo i due ricercatori, *l'intelligenza misura la capacità di un agente di raggiungere obiettivi in un'ampia varietà di ambienti* [6]: una volta raggiunto questo stadio, l'IA sarebbe secondo alcuni ricercatori in grado di creare intelligenze artificiali meglio di quanto faccia l'uomo, diventando quindi paragonabile ad una nuova forma di vita e raggiungendo lo stadio di ASI (Artificial Super Intelligence) [4]. Questa fase di convivenza fra uomo e IA è conosciuta come singolarità: per quanto remota e fantascientifica possa sembrare, la sua sola presenza in diversi articoli specialistici mostra quanto alto sia l'interesse della comunità scientifica per questo campo di ricerca.

1.2.2 Machine learning

Obiettivo del machine learning è scoprire algoritmi che trovino soluzioni a problemi troppo complessi per essere risolti con il metodo tradizionale. Stabilire formalmente le specifiche e implementare un programma informatico può risultare troppo complesso e oneroso in molti scenari, primo fra tutti il riconoscimento di caratteri in un'immagine: data l'enorme varietà di font presenti nei testi scritti su supporti informatici e l'ancor più elevato numero di calligrafie riscontrabili nella scrittura manuale, definire un insieme di regole per riconoscere ogni singolo carattere sarebbe una grande sfida.

Un'alternativa a questo metodo, abbandonato negli anni Ottanta dopo lo scarso successo dei cosiddetti *expert systems*, sono gli algoritmi di machine learning. Questi sistemi non richiedono altro che un insieme di dati etichettati detto dataset (nell'esempio precedente, un insieme di immagini di lettere o parole corredate dai caratteri ASCII corrispondenti), che utilizzeranno per creare un insieme di regole detto modello. Essendo il modello generato a partire dall'esame del dataset, sarà possibile etichettare nuovi dati non presenti nell'insieme di partenza.

A partire dagli anni 2000, il machine learning ha riscosso un enorme successo grazie alla quantità di dati messi a disposizione dall'avvento di internet e dall'aumento delle prestazioni di processori e memorie. Il machine learning non offre interpretabilità, non è cioè possibile conoscere il ragionamento che ha portato all'assegnazione di una certa etichetta invece di un'altra, ma ciò non gli ha impedito di trovare applicazione in molteplici campi, dalla conversione del parlato in testo alla traduzione di testi, passando per i veicoli a guida autonoma. Senza elencare tutti gli impieghi possibili di questa tecnologia, si può però provare a suddividerli in tre tipologie principali:

classificazione ovvero l'assegnazione ad una categoria specifica presa da un insieme predefinito (l'animale nella foto è un cane o un gatto? Quest'email è spam oppure no?)

clustering suddivisione del dataset in un numero di sottogruppi detti cluster e accomunati da alcune proprietà non noto a priori

predizione previsione del valore di un dato a partire dal suo precedente andamento (Quale sarà il valore di mercato di quest'azione fra due giorni? E fra un mese?)

Un'altra distinzione utile quando si parla di machine learning riguarda proprio il termine learning. Vi sono infatti diversi modi in cui un algoritmo può apprendere le informazioni necessarie a generare il proprio modello:

Supervised learning l'algoritmo riceve un dataset etichettato, ad esempio un insieme di foto accompagnate dall'informazione sul fatto che rappresentino un cane o un gatto. Utile per problemi di classificazione (assegnazione ad una classe in un insieme definito) e regressione (previsione del valore di una variabile continua)

Unsupervised learning il dataset non è etichettato, il programma sarà unicamente in grado di raggruppare campioni simili senza fornire altre informazioni. Continuando con l'esempio di prima, l'algoritmo raggrupperà le foto in due gruppi, cani e gatti, solo in base alla loro somiglianza, ma continuerà a non sapere cosa sia un cane e cosa sia un gatto.

Semi-supervised learning solo alcuni elementi del dataset sono etichettati. Dopo una fase di clustering si usano i pochi dati etichettati conosciute per assegnare un'etichetta ad ogni gruppo. Ciò è utile perché permette di risparmiare una buona parte del tempo che richiederebbe l'aggiunta manuale di un'etichetta ad ogni punto del dataset, ma quest'area di ricerca è ancora in fase di sviluppo.

Reinforcement learning in base al proprio stato e all'ambiente esterno, l'algoritmo prende decisioni volte a massimizzare una funzione obiettivo. Questa modalità di apprendimento è indispensabile in contesti in cui i possibili stati sono troppi per essere mappati e l'ambiente esterno è in continuo mutamento, come la guida di veicoli o giochi quali gli scacchi.

1

¹Le informazioni presenti in questo sottoparagrafo sono tratte del libro "An Introduction to Machine Learning", che alle definizioni teoriche affianca indicazioni pratiche per sviluppare sistemi di machine learning [7]

1.2.3 Deep learning e reti neurali

Deep learning

Il deep learning è una sottobranca del machine learning di cruciale importanza per questo testo, in quanto comprende i large language models e l'intelligenza artificiale generativa, che sfrutteremo nei prossimi capitoli per generare del codice. Si può definire il deep learning a partire dal representation learning, una serie di metodi che permettono di far automaticamente decidere alla macchina come rappresentare i dati che le vengono passati per essere classificati. I metodi deep learning sono metodi di representation learning che sfruttano livelli di rappresentazione multipli: sfruttando la composizione di moduli non lineari, si può infatti fare in modo che ogni livello trasformi la rappresentazione dell'input che riceve, astraendola un pochino. Se ad esempio si passa un'immagine ad un algoritmo di deep learning, il primo strato riceverà un insieme di pixel e cercherà i bordi dell'immagine, il secondo strato riconoscerà particolari combinazioni di questi bordi che i successivi strati riuniranno poi in figure sempre più complesse per risalire all'oggetto di partenza, come mostrato nella figura 1.1 (immagine tratta da [8]).

Un aspetto interessante degli strati che compongono un algoritmo di deep learning è che non sono programmati dall'uomo, ma si basano sull'unsupervised learning (vedi paragrafo precedente) e imparano autonomamente a riconoscere i pattern. Ciò ha permesso al deep learning di trovare applicazioni in svariati ambiti, come il riconoscimento di strutture in dati multi-dimensionali, primi fra tutti le immagini e il linguaggio parlato. L'enorme mole di dati oggi disponibili, il richiedere poca programmazione da parte dell'uomo e non da ultimo il fatto che il nostro stesso apprendimento (così come quello degli altri animali) sia in larga parte "unsupervised" rendono il deep learning un promettente campo di ricerca [9]

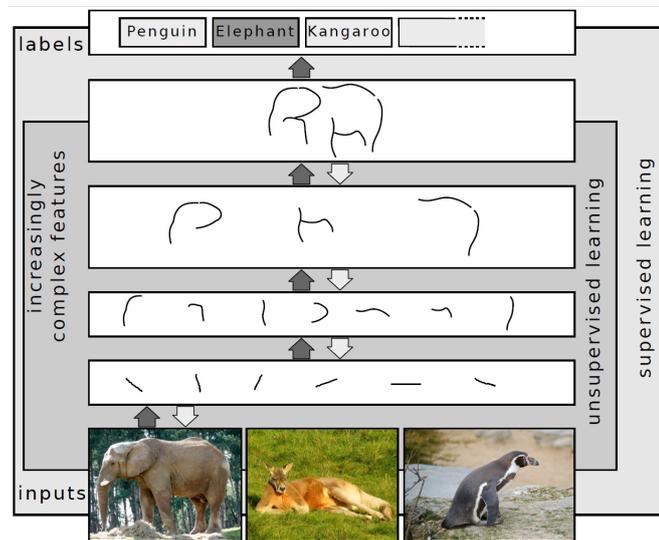


Figura 1.1. Esempio di come dall'immagine di un elefante vengano individuati i contorni principali che verranno via via messi insieme per comporre la sagoma dell'elefante.

Reti neurali

Parlando del deep learning non si possono non menzionare le reti neurali, la cui organizzazione a livelli è sfruttata per astrarre sempre di più gli input come discusso sopra. Si può definire una rete neurale come un modello computazionale ispirato alla struttura delle reti di neuroni presenti nel cervello umano [10]: questi modelli si basano infatti sulla fitta interconnessione di un gran numero di nodi di calcolo, rappresentabile come un grafo orientato pesato in cui tutti i nodi della rete ricevono dati in input, li processano e restituiscono un segnale di output diretto ad altri nodi o verso l'esterno. Ciò che differenzia i nodi uno dall'altro è lo strato a cui appartengono; a seconda della loro posizione possono infatti appartenere a tre tipi di strato: input, output o strato nascosto. Fra questi strati, il più interessante è sicuramente quello nascosto, perché è qui che avvengono le trasformazioni degli input e il processo di training: modificando i pesi degli archi che interconnettono i vari strati nascosti si può determinare quanto peso abbia l'output di ogni neurone sul risultato finale; la determinazione dei pesi avviene tramite un processo di training in cui vengono forniti sia gli input che gli output attesi.

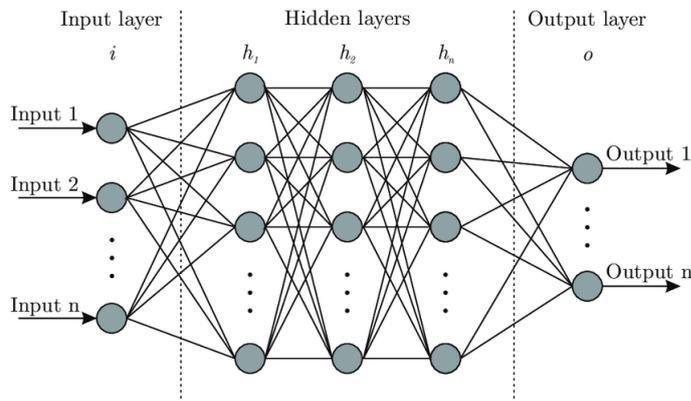


Figura 1.2. Rappresentazione schematica di una rete neurale.

In sintesi, ogni neurone riceve una serie di valori in input, a cui vengono applicati dei pesi e a cui può essere aggiunto un bias esterno: la somma di questi numeri verrà poi passata ad una funzione detta di attivazione, che determinerà il valore in uscita dal nodo, come mostrato nella figura 1.3 [12].

A seconda che il grafo sia aciclico o meno si può parlare di reti feedforward o ricorrenti: le prime, più semplici furono fra le prime ad essere sviluppate negli anni '60, ma le loro scarse potenzialità fecero desistere i ricercatori, mentre le seconde trovano ampio utilizzo in campi in cui sia richiesta la predizione di un risultato. Basandosi su dati sequenziali o su serie temporali, le reti ricorrenti si prestano infatti a problemi di tipo temporale o ordinale come la traduzione di testi, il riconoscimento vocale e persino la predizione del cambiamento climatico, che coinvolge dati multi-dimensionali che un'analisi di tipo tradizionale non sarebbe in grado di gestire al meglio [11].

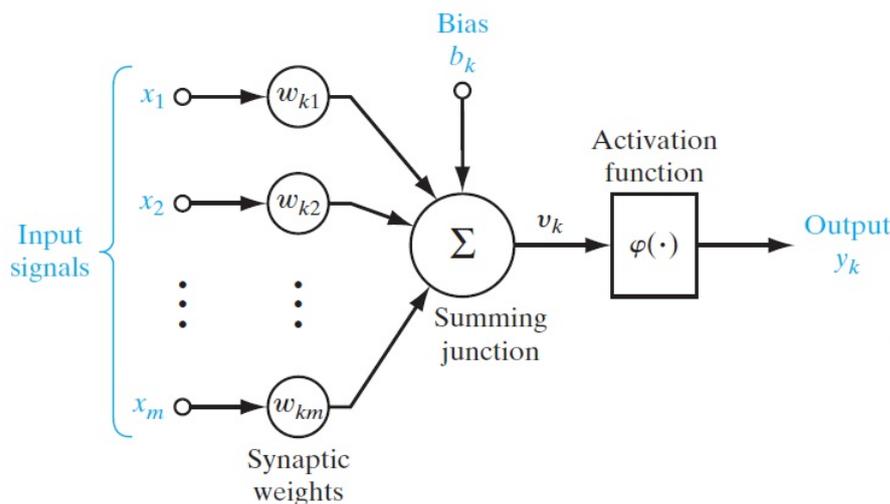


Figura 1.3. Schema che illustra come gli input di un neurone vengano processati per generare l'output.

1.2.4 I Large Language Models e generative AI

La serie di definizioni elencate finora ci permette infine di introdurre i concetti di large language model e di intelligenza artificiale generativa, importantissimi per questo lavoro in quanto l'intelligenza usata per generare il codice si basa proprio su un large language model.

Intelligenza artificiale generativa

Per definire i modelli generativi, è utile distinguerli da quelli discriminativi: gli esempi proposti finora avevano tutti in comune l'obiettivo di distinguere se ad un'entry del dataset fosse possibile assegnare una certa etichetta, calcolando la proprietà che l'oggetto x appartenesse alla categoria y . Lo scopo dei modelli generativi è invece calcolare la probabilità che l'oggetto x esista [13]. Prendendo come esempio la generazione di immagini, se un modello discriminativo impara a distinguere quali immagini rappresentino un dato animale (la figura x ha l'80% di probabilità di raffigurare un cavallo), un modello generativo genera invece come output una sequenza di pixel con un'elevata probabilità di rappresentare l'animale in questione.

Un'altra peculiarità che differenzia questi modelli da quelli discriminativi è il generare output diversi anche ricevendo più volte lo stesso input: molti modelli generativi sfruttano addirittura componenti randomiche in fase di training o per introdurre variabilità durante la generazione dell'output.

Questo tipo di intelligenza artificiale sta trovando sempre più impieghi: dalla generazione di testo a quella di immagini o video fino alla creazione di molecole e ovviamente alla generazione di codice, come vedremo nei capitoli successivi.

Large Language Models (LLM)

Uno studio dell'università di Pechino [14] definisce i large language models come modelli linguistici generativi di grandi dimensioni. Ciò che differenzia questi modelli dai precedenti è proprio

la mole di dati forniti in fase di training: superato un certo numero di parametri (componenti del modello quali pesi e bias della rete neurale modificabili con l'addestramento), le reti mostrano nuovi comportamenti e abilità come l'in-context learning, ovvero la capacità di eseguire un compito semplicemente ricevendo istruzioni in linguaggio naturale (si fornisce un esempio o si definisce il problema a parole, senza bisogno di training aggiuntivo). Esempi di large language models sono i sempre più diffusi gpt-3 e 4: le capacità di questi modelli hanno permesso loro di passare da semplici assistenti per compiti specifici ad essere in grado di risolvere problemi di tipo più generico della vita di tutti i giorni, come la traduzione di tesi, la scrittura di brevi testi o pezzi di codice.

Le abilità sviluppate da questi sistemi li hanno resi popolari anche fra i non appartenenti al settore informatico, generando al contempo nuove opportunità e apprensione in molti settori:

educazione se da una parte si teme che studenti e docenti abusino dello strumento senza spirito critico, dall'altra non si possono non notare le potenzialità nella pianificazione di lezioni o nell'assistenza durante lo studio da casa [15].

chatbot sono stati sviluppati diversi agenti conversazionali in grado di fornire risposte "*educate, utili, sintetiche, amichevoli e attente alla sicurezza*" e basate su metriche quali "*interessante, coerente, naturale e consistente*". Questi sistemi però sono ancora lontani dall'essere perfetti, in quanto tendono a "dimenticare" rapidamente quanto detto in precedenza, rischiando così di ripetersi e soffrono di latenze piuttosto elevate nel fornire una risposta [16].

biologia computazionale in questo campo di studi, che si serve di strumenti matematici, informatici e di analisi dati per comprendere i sistemi biologici, gli LLM trovano applicazioni nell'embedding delle proteine (rappresentazione della loro struttura in un formato comprensibile ad una macchina) e nell'analisi del genoma umano. Come nell'esempio precedente, la finestra di contesto limitata pone limiti alla lunghezza delle sequenze di genoma che si possono studiare o alla dimensione del dataset di proteine analizzabili [16].

programmazione il training dei modelli specializzato esclusivamente per la generazione di codice, ma anche i modelli più generici come gpt, rendono gli LLM un valido aiuto per gli informatici. Nonostante la ridotta finestra contestuale permetta di lavorare solo su piccole porzioni di codice, questi strumenti stanno affiancando i programmatori sempre di più sia nella creazione che nella revisione e documentazione del codice, come approfondiremo con alcuni esempi nel prossimo capitolo.

legge nonostante la velocità con cui il materiale di training diventa obsoleto a causa della continua emissione di leggi nuove o di aggiornamenti di quelle già esistenti, gli LLM possono facilitare il lavoro dei legali nella ricerca di casi passati simili a quello in esame, nella previsione dell'esito di una controversia e nella spiegazione di concetti legali. In questo campo la diffusione dei modelli linguistici di grandi dimensioni è ancora molto limitata a causa dei risultati finora mediocri [16].

medicina la soluzione di quesiti medici e la formulazione di diagnosi è al momento un ambito ancora troppo delicato, dove il rischio di errori e la presenza di bias rendono le prestazioni dei modelli di grandi dimensioni inferiori a quelle di modelli più piccoli addestrati in maniera più specifica solo su documentazione clinica [16].

scienze sociali e psicologia gli LLM sono stati applicati con tre scopi principali: modellare il comportamento umano, analizzare il comportamento degli LLM stessi e simulare

relazioni sociali. Nel primo caso si è osservato che i modelli costituiscono un sostituto che fornisce all'incirca le stesse risposte dei partecipanti umani ma è più economico e permette di condurre esperimenti in tempi più brevi; nel secondo si sono sottoposti i modelli a test volti a misurare quanto sviluppati siano alcuni tratti della personalità, ottenendo risultati simili a quelli delle persone in quasi tutti gli ambiti ad eccezione della creatività; nel terzo caso si è riusciti con successo a far interagire 25 agenti artificiali che hanno conversato per replicare uno studio sul comportamento umano. Ciò a cui bisogna prestare attenzione sono i possibili preconcetti o opinioni personali nascosti nel dataset fornito in fase di addestramento [16].

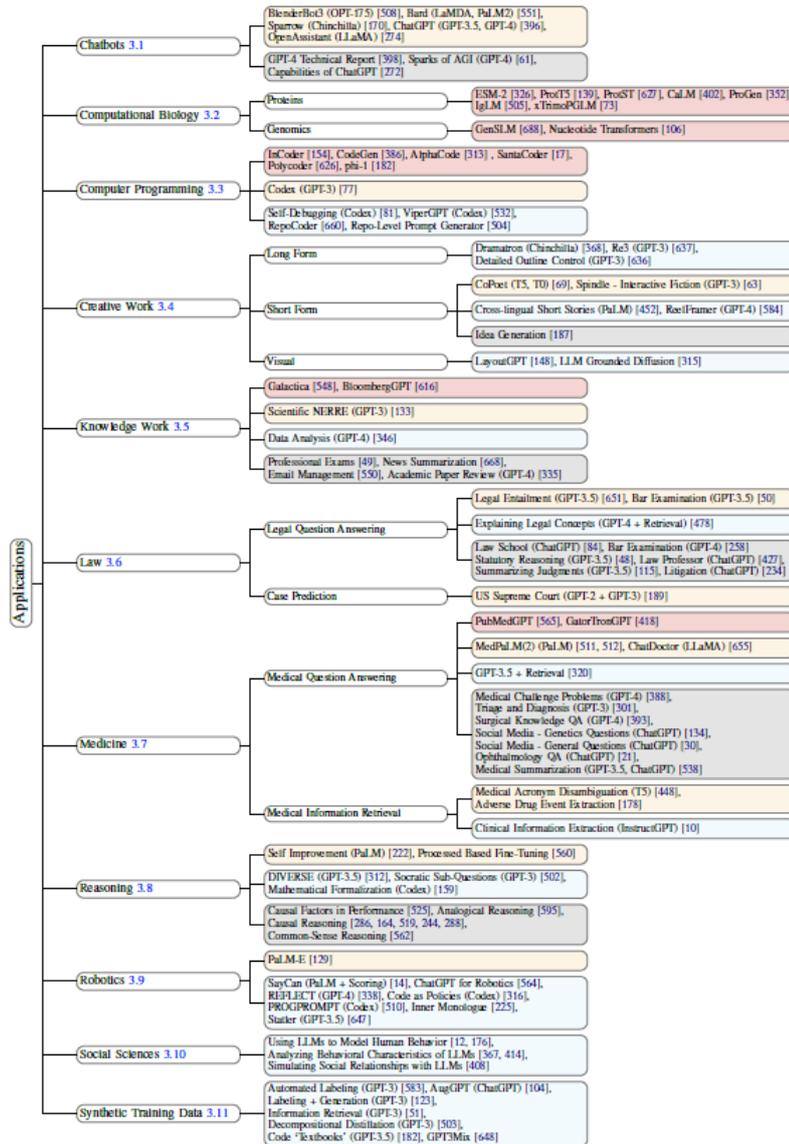


Figura 1.4. Elenco completo delle applicazioni dei large language model individuati in "Challenges and Applications of Large Language Models", alcune delle quali sono riportate in questo sottoparagrafo [16] (immagine tratta dall'articolo).

1.2.5 Intelligenza artificiale: una tassonomia finale

Una buona sintesi di quanto detto finora è fornita in "Large language models: a primer and gastroenterology applications" [17]. Gli autori di questo articolo si rivolgono ad un pubblico di medici per illustrare le potenzialità dell'applicazione dell'intelligenza artificiale nel loro settore di studio e forniscono ai propri colleghi un'introduzione ai large language models, accompagnandola

con un grafico di Venn che ben riassume le relazioni fra le categorie che abbiamo elencato nei precedenti paragrafi:

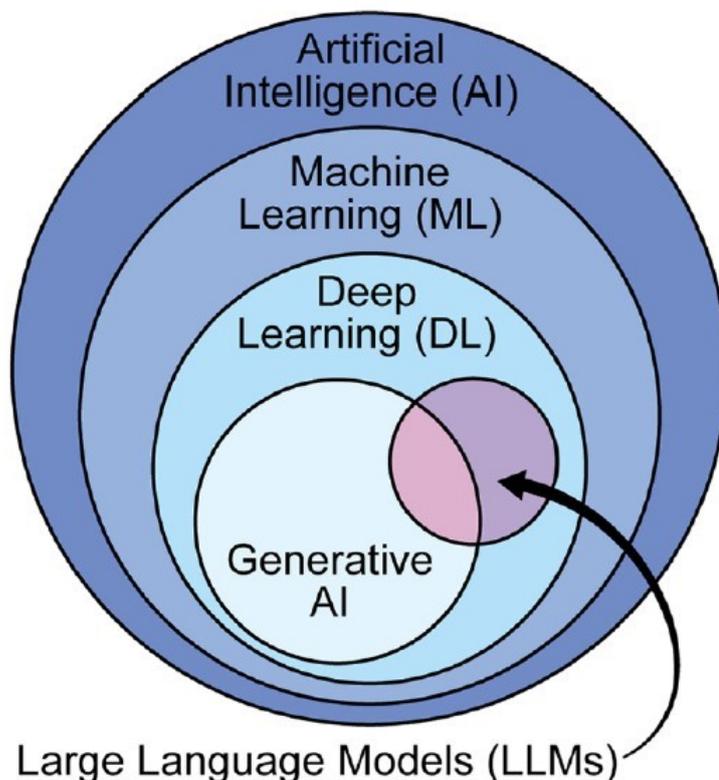


Figura 1.5. Grafico illustrante la tassonomia di diversi tipi di intelligenze artificiali [17] (immagine tratta dall'articolo citato).

Come mostrato nell'immagine 1.5, generative AI e large language models risultano essere entrambi sottoinsiemi del deep learning, ma senza coincidere uno con l'altro: esistono large language models il cui scopo non è generativo (sentiment analysis, riconoscimento di testi simili, ecc), così come ci sono esempi di generative AI che invece degli LLM si basano su altri tipi di rete. Si può ad esempio basare un'intelligenza artificiale di tipo generativo sulle reti GANs (Generative Adversarial Networks), che sfruttano due reti neurali addestrate congiuntamente: il compito di queste reti, dette generatore e discriminatore è rispettivamente quello di generare dati e quello di confrontarli con i dati di training.

La parte di questo diagramma su cui si focalizzerà questo lavoro è però l'intersezione fra i due insiemi più interni: il nostro studio si baserà infatti sull'uso dei large language models per generare il codice del prototipo di una pagina web.

Capitolo 2

Tabnine e GitHub Copilot: una panoramica e confronto fra i due

Prima di sperimentare nuove soluzioni, cercheremo in questo capitolo di capire quale sia il livello attuale dell'aiuto che l'intelligenza artificiale offre attualmente ai programmatori. Ci concentreremo in particolare su Tabnine e su GitHub Copilot, fornendo una panoramica di ciascuno dei due per poi confrontare le loro prestazioni.

2.1 Tabnine

L'inizio della storia di Tabnine risale al 2015, quando Dror Weiss ed il professor Eran Yahav dell'università israeliana Technion (istituto israeliano di tecnologia) hanno creato Codota con l'intento di sfruttare l'IA per imitare il modo in cui il codice viene generato dagli esseri umani. Basato su una finestra aggiuntiva da affiancare a quella di scrittura del codice nei più popolari IDE, Codota tentava di prevedere il codice che l'utente avrebbe generato basandosi sull'analisi di milioni di repository di codice e sulla ricerca svolta alla Technion.

Nel 2019 la piattaforma ha acquistato Tabnine, fondata un anno prima dal canadese Jacob Jackson, che all'epoca dell'acquisto aveva solo 22 anni. Se Codota si basava su un'analisi di tipo semantico per costruire un modello che rappresentasse il significato del codice, l'approccio di Tabnine era più di tipo testuale: il rilevamento della startup canadese mirava all'unione di queste due strategie e ha portato, secondo quanto affermato dallo stesso professor Yahav, a risultati da subito soddisfacenti e non raggiungibili da nessuno dei due approcci singolarmente [18].

Dopo l'acquisto del concorrente, Codota ha visto aumentare i propri utenti del 1000% in un anno ed è utilizzato da più di un milione di persone al mese. L'azienda vanta importanti finanziatori: insieme al partner di vecchia data Khosla Ventures, si possono infatti trovare aziende come Qualcomm e Samsung [19].

2.2 GitHub Copilot

Pur essendo stato lanciato sei anni dopo Tabnine, GitHub Copilot non ha però tardato ad affermarsi, grazie alla sua abilità di estrapolare il contesto dal codice che l'utente sta digitando. Rilasciato in anteprima il 29 giugno 2021, Copilot è stato poi reso disponibile come plugin per i prodotti JetBrains il 29 ottobre dello stesso anno. Dopo il debutto su Visual Studio nel marzo



Figura 2.1. Il logo di tabnine.

2022, quest'estensione è diventata a pagamento tre mesi più tardi. La collaborazione con OpenAI continua tuttora: a novembre 2023 la chat di Copilot è stata migliorata con l'introduzione di GPT-4.

Questo strumento è stato sviluppato da GitHub in collaborazione con OpenAI, che ha realizzato il sistema di intelligenza artificiale OpenAI Codex apposta per l'occasione, in modo da far basare l'assistente virtuale su un dataset che contenesse molto più codice rispetto a quello di GPT-3 (modello di riferimento di OpenAI dell'epoca). Per valutare le prestazioni di Codex, OpenAI ha raggruppato una serie di prompt in una raccolta chiamata HumanEval e dotata di test a cui sottoporre il codice generato a partire dai prompt: GPT-3 superava lo 0% di questi test e Codex ne superava il 28,7% al primo tentativo, mentre se gli veniva concessa la possibilità di generare 100 risultati ad ogni problema, il 77,5% delle volte almeno una delle soluzioni superava i test. Il più grande modello gratuito di Tabnine si è fermato invece a punteggi di 2,58% e 7,59% con 1 e 100 tentativi a disposizione [20].

Codex è stato addestrato ponendo particolare attenzione a JavaScript, TypeScript, Ruby, Go e Python [21]. La collaborazione con GitHub ha sicuramente contribuito al risultato finale, dato che per quest'ultimo linguaggio sono stati forniti 159 GB di codice provenienti da 54 milioni di repository pubbliche di GitHub: questa fonte di codice è però stata causa di controversie legali, la più rilevante delle quali è l'azione collettiva guidata da Matthew Butterick, avvocato, tipografo e programmatore statunitense che nel novembre 2022 ha avviato una causa contro GitHub, Microsoft (proprietaria di GitHub) e OpenAI. Le tre aziende sono state accusate di non rispettare le licenze open-source in cui viene stabilito il diritto degli utenti a veder riconosciuto il proprio nome quando si riporta codice da loro prodotto né i termini di servizio di GitHub e le sue politiche sulla privacy [22].



Figura 2.2. Il logo di GitHub Copilot.

Nonostante le controversie legali, i dubbi sulla sicurezza del codice generato e le preoccupazioni sulla sua crescente diffusione fra gli studenti, il successo di Copilot è innegabile: a metà 2024 Microsoft ha affermato di aver ricavato 33 milioni di dollari da questo strumento contro i 10 investiti per realizzarlo e di avere oltre 1,3 milioni di persone abbonate al servizio. L'azienda ha inoltre dichiarato che il ricavo di GitHub è aumentato del 40% ogni anno e che Copilot Business ha aumentato la produttività di compagnie del calibro di Dell e Goldman Sachs [23].

2.3 Valutazione delle prestazioni dei due strumenti

Passiamo ora ad un esame di GitHub Copilot e di Tabnine in azione, osservando le principali funzionalità fornite dai due strumenti.

2.3.1 Spiegazioni e suggerimenti in linea

Sia Tabnine che Copilot offrono la possibilità di generare spiegazioni di codice già esistente: nell'esempio della figura 2.3 Tabnine spiega addirittura il codice commentato. Per quanto riguarda il suggerimento dei tipi in fase di dichiarazione, sia Tabnine che Copilot hanno buona capacità di indovinare il tipo a partire dal nome del dato, ma non sono in grado di suggerire l'import della classe in caso quel tipo non sia ancora stato utilizzato. In quanto a import, i suggerimenti migliori arrivano da Visual Studio, perché vede l'intero ambiente di lavoro. Nella figura 2.4 si possono vedere sulla sinistra la classe suggerita da Tabnine e sulla destra l'import della classe proposto da Visual Studio dopo aver iniziato a digitare il nome della classe: nel secondo caso, la riga con l'import viene aggiunta in automatico, mentre nel primo no.

Anche chiedendo a Copilot di risolvere un errore di classe creata dall'utente e non importata, si riceve come suggerimento un percorso di import non corretto, dimostrando così come Copilot non abbia davvero chiara la struttura del progetto.

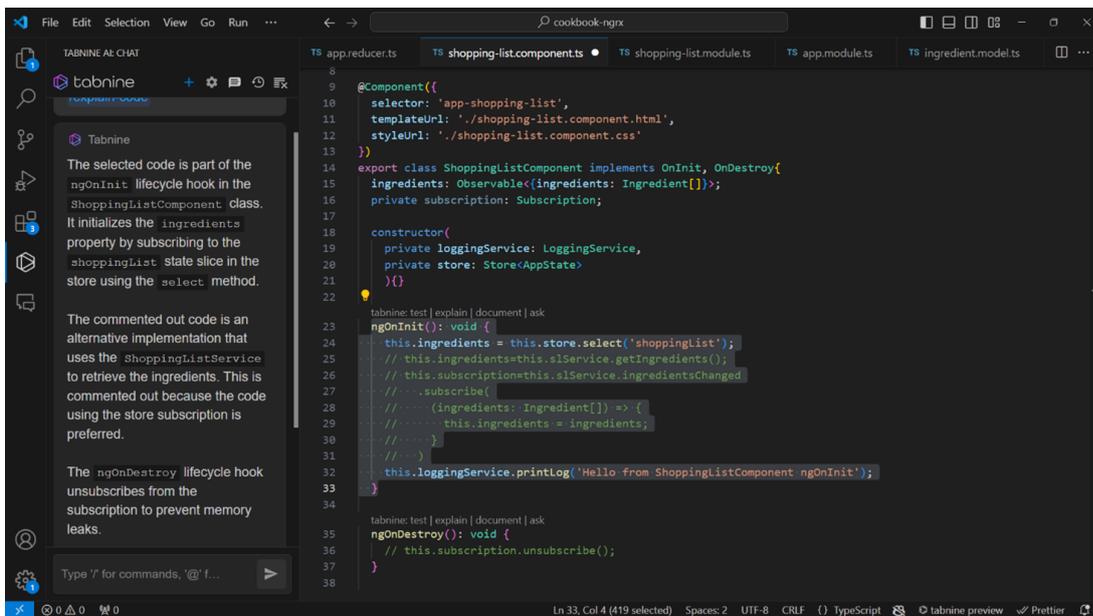


Figura 2.3. Tabnine spiega il codice.

2.3.2 Generazione di intere funzioni

Per quanto riguarda la comprensione del codice prodotto dall'utente, sia Tabnine che Copilot dimostrano buone capacità, suggerendo metodi complementari a quelli già scritti. Ad esempio nella figura 2.5 è mostrato un metodo per inviare ai componenti che siano interessati informazioni sull'utente appena autenticatosi con successo: Tabnine comprende lo scopo del metodo e decide di

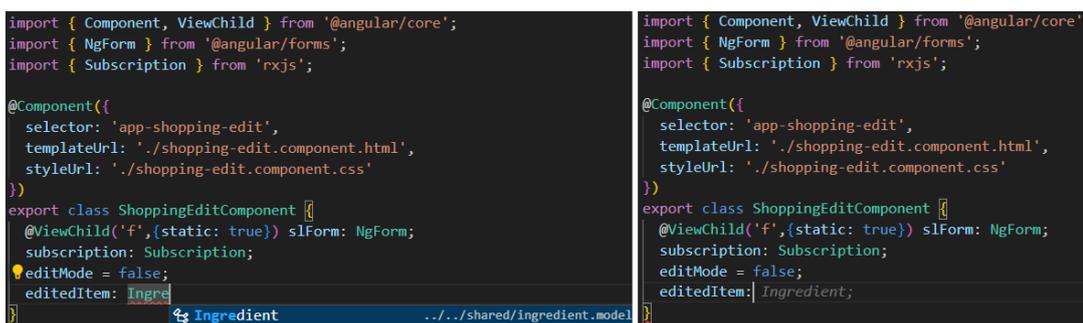


Figura 2.4. A sinistra il suggerimento di Visual Studio, a destra quello di Tabnine.

proporre l'aggiunta di un metodo da utilizzare come reazione ad un evento di autenticazione non andata a buon fine, adattando anche il payload invece di copiare ciecamente quello precedente.

```

export const authenticateSuccess = createAction(
  '[Auth] AuthenticateSuccess',
  props<{
    payload: {
      email: string,
      userId: string,
      token: string,
      tokenExpirationDate: Date,
      redirect: boolean
    }>()
  });

export const authenticateFail = createAction(
  '[Auth] AuthenticateFail',
  props<{ payload: {error: string} }>()
);
  
```

Figura 2.5. Tabnine suggerisce un metodo simile a quello appena scritto.

Tutti e due gli agenti virtuali si sono anche dimostrati capaci di generare porzioni di codice piuttosto grandi: nella figura 2.6 si può vedere una funzione di discreta complessità che ha richiesto pochissime correzioni generata quasi interamente da Tabnine dopo averne scritta una simile, mentre nella figura 2.7 Copilot ha generato grazie alla chat in linea una funzione generica ma comunque corretta e corredata di commenti. Nella seconda immagine si può notare ancora una volta come questi sistemi di intelligenza artificiale non siano in grado di osservare il programma nel suo complesso: pur facendo riferimento alle azioni (particolari eventi a cui Angular fornisce una maniera strutturata di reagire) definite in un file specifico, Copilot non è in grado di recuperare informazioni su quelle azioni e inserisce nomi di azioni generici come `action1` e `action2`.

```
authLogin = createEffect(() => {
  this.actions$.pipe(
    ofType(loginStart),
    switchMap((loginAction) => {
      return this.http
        .post<AuthResponseData>(
          'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=' +
            environment.firebaseAPIKey,
          {
            email: loginAction.payload.email,
            password: <PASSWORD>,
            returnSecureToken: true,
          }
        )
        .pipe(
          tap((resData) => {
            this.authService.setLogoutTimer(+resData.expiresIn * 1000);
          }),
          map((resData) => {
            return handleAuthentication(
              +resData.expiresIn,
              resData.email,
              resData.localId,
              resData.idToken
            );
          })
        )
        .catchError((errorRes) => {
          return handleError(errorRes);
        })
    })
  );
});
```

Figura 2.6. Tabnine suggerisce un metodo simile a quello appena scritto.

2.3.3 Chat

Entrambi gli strumenti offrono anche un servizio di chat in cui è possibile sia parlare del codice a cui si sta lavorando che porre domande generiche di programmazione. Sia Copilot che Tabnine a volte dimostrano sotto questo aspetto poca capacità di intuizione: quando gli viene posta una domanda su come aggiungere un package ad un progetto Angular, entrambi forniscono come risposta il generico `npm install`, al posto di parlare del comando `ng add`, tipico di Angular CLI, l'interfaccia da linea di comando che permette di lanciare, inizializzare e gestire i componenti di un'applicazione Angular. L'esempio con Tabnine è visibile nell'immagine 2.8. Copilot ha reagito bene al suggerimento di generare gli override mancanti dei metodi delle interfacce implementate.

```
use createReducer() to create reducers for the actions listed in recipe.actions.ts
Accept Discard ↺

1 import { createReducer, on } from '@ngrx/store';
2 import { Action } from '@ngrx/store';
3 import * as RecipeActions from './recipe.actions';
4
5 // Define the initial state for your reducer
6 const initialState = {
7   // Define your initial state properties here
8 };
9
10 // Create the reducer using createReducer()
11 export const recipeReducer = createReducer(
12   initialState,
13   // Define your reducer logic using the `on` function
14   on(RecipeActions.action1, (state, action) => {
15     // Handle the action1 logic and return the updated state
16     return {
17       ...state,
18       // Update the state properties based on action1
19     };
20   }),
21   on(RecipeActions.action2, (state, action) => {
22     // Handle the action2 logic and return the updated state
23     return {
24       ...state,
25       // Update the state properties based on action2
26     };
27   }),
28   // Add more `on` functions for other actions if needed
29 );
```

Figura 2.7. Copilot genera una funzione dietro indicazione dell'utente.

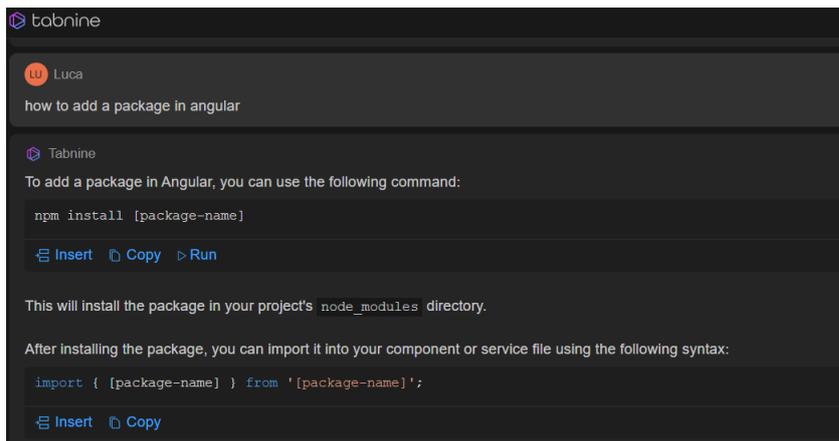


Figura 2.8. La chat di Tabnine.

2.3.4 Copilot come aiuto nella stesura di testo

L'utilizzo forse più sorprendente della funzionalità di chat (in questo caso quella di Copilot) è stato sottoporle parte di una versione preliminare del capitolo 5 in cui si descriveva la struttura di un programma scritto diversi mesi prima di cui non si ricordava tutto. Chiedendo all'assistente virtuale di verificare se le informazioni fossero corrette, si è ottenuta una risposta completa, che riprendeva addirittura alcune frasi scritte dall'utente (vedi figura 2.9).

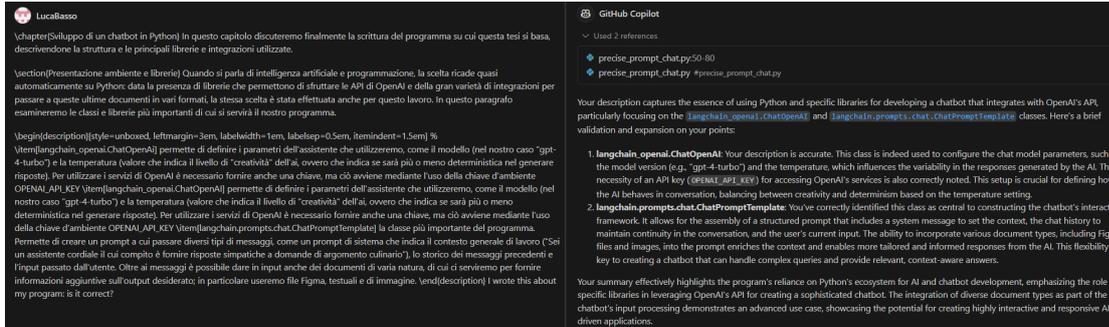


Figura 2.9. Copilot commenta il testo presentatogli (notare come per generare la risposta venga fatto riferimento ai file Python).

Capitolo 3

Introduzione ad Angular

Presentiamo ora Angular, la piattaforma di sviluppo utilizzata per il codice delle pagine web che faremo generare al nostro chatbot.

3.1 Cenni storici

Angular è nato nel 2009 come progetto di Miško Hevery come progetto per facilitare lo sviluppo di alcune applicazioni web all'interno della compagnia per cui lavorava, diventando open source l'anno successivo: da questo momento la sua diffusione è stata rapida grazie all'appoggio di Google e al supporto da parte della comunità degli sviluppatori. Nel 2016 è uscita la versione 2 di Angular, che per adeguarsi da progetto di piccole dimensioni interno ad una compagnia a framework utilizzato su larga scala ha richiesto talmente tante modifiche da essere considerato un linguaggio a parte. Questo aggiornamento ha diviso gli sviluppatori in due, perché la nuova versione non risultava retrocompatibile con la prima a causa di una serie di differenze:

- il nuovo Angular si basa su TypeScript, mentre quello vecchio su JavaScript (per questo è stato ribattezzato AngularJS, in contrapposizione alla nuova versione, che ha mantenuto il nome Angular).
- l'architettura di Angular si basa su un albero di componenti, i controller e gli scope di AngularJS non esistono più.
- numerose direttive cambiano nome, come `ng-repeat` che diventa `ngFor` ed `ng-click` che diventa `click`
- Angular sfrutta Zone.js, che rileva automaticamente cambiamenti nel modello dell'applicazione, rimpiazzando il ciclo di digest di AngularJS, che prevedeva di controllare ciclicamente ogni componente per vedere se il suo stato fosse cambiato.
- Angular introduce i form reattivi, che garantiscono maggior controllo e flessibilità rispetto a quelli costruiti con `ngModel` di AngularJS (che sono però ancora utilizzabili).
- Il ciclo di vita dei componenti è monitorato da degli hook che offrono un controllo più fine rispetto agli `$scope` di AngularJS (`ngOnInit`, `ngOnDestroy`, ...).
- il sistema di gestione dei moduli, blocchi in cui vengono raggruppate parti dell'applicazione con funzionalità correlate, è cambiato e si basa sull'uso della direttiva `@NgModules`.

Dopo un periodo di iniziale insuccesso dovuto alla difficoltà di adattamento incontrate dagli sviluppatori che si trovavano a dover riscrivere intere applicazioni da capo, Angular ha ricominciato ad essere riconosciuto grazie alla sua capacità di fornire soluzioni per quasi ogni aspetto di un'applicazione, senza doversi affidare a troppe librerie esterne come avviene con React [24].



Figura 3.1. Il logo di Angular.

3.2 Principali caratteristiche

In questo paragrafo ci si riferirà esclusivamente all'Angular più moderno, mai ad AngularJS.

3.2.1 Componenti

La struttura di un progetto Angular è basata sull'uso di componenti, blocchi elementari componibili a piacimento: ogni blocco è costituito da una cartella contenente un file CSS, un file HTML con la struttura del componente e un file TypeScript responsabile della sua logica. È possibile richiamare un componente Angular all'interno del codice HTML semplicemente includendo il nome definito nel suo file .ts tra i simboli `<` e `>`, come un qualunque tag HTML (Angular deve il proprio nome proprio all'uso di queste parentesi angolari). Qualunque file appartenga ad un componente è identificato dal suffisso `.component` prima dell'estensione (vedi figura 3.2). Per componenti molto semplici è possibile includere il codice HTML direttamente nel file .ts grazie ad un attributo incluso nell'annotazione `@Component`, che permette ad una classe TypeScript di essere riconosciuta come componente Angular. Lo stesso vale per la definizione di stili CSS [25].

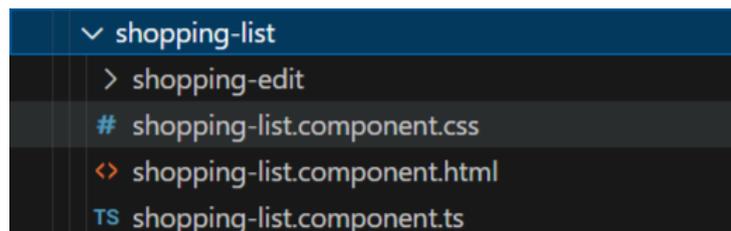


Figura 3.2. Esempio di un componente al cui interno è annidato un altro componente. Notare come tutti i file abbiano il suffisso ".component".

3.2.2 Compilazione

Basandosi su TypeScript, un progetto Angular necessita di essere ricompilato prima di essere aperto da un utente, perché questo linguaggio non è comprensibile dai browser. Per questo motivo

Angular si serve dell'Ahead-Of-Time compilation (AOT), sistema che compila un'applicazione in tre fasi:

- il compilatore TypeScript e l'AOT collector generano una rappresentazione del codice, metadati inclusi
- i metadati raccolti nel passo precedente vengono ora interpretati e validati
- il compilatore di TypeScript viene utilizzato per validare le espressioni usate nei binding (opzionale)

La compilazione AOT è utilizzata di default a partire dalla versione 9 di Angular e permette di compilare un'applicazione prima che venga lanciata nel browser, riducendo i tempi di caricamento iniziali per l'utente finale e le dimensioni della pagina (che non richiede più un compilatore): ciò rende questo tipo di compilazione ottimale per la produzione. In fase di sviluppo può però essere utile la strategia JIT (Just-In-Time), che pur richiedendo pacchetti più grandi per includere il compilatore e tempi di attesa più lunghi perché la compilazione viene eseguita a runtime dal browser, è più comodo in fase di debug in quanto permette di propagare le modifiche del codice in tempo reale, senza ricompilare l'intera applicazione e ha tempi di compilazione minori [26].

3.2.3 Data binding

Per assicurare la consistenza fra dati nella vista utente e dati nel modello sottostante, Angular ha introdotto dalla versione 2 il binding bidirezionale. Questa strategia permette, quando lo si desidera, di combinare i già esistenti property ed event binding: il primo permette di assegnare ad un elemento del DOM un valore che viene aggiornato dinamicamente legandolo ad una proprietà della logica del modello [27], il secondo di reagire ad eventi dell'utente legati a mouse, tastiera e simili [28].

Un esempio di property binding è la linea di codice

```
<img alt="item" [src]="itemImageUrl" > ,
```

che aggiorna il valore di `src` ogni volta che la proprietà `itemImageUrl` viene modificata nel modello del componente, mentre un esempio di event binding è

```
<button (click)="onSave()">Save</button> ,
```

che chiama il metodo `onSave()` ogni volta che il tasto viene premuto. Notare l'uso dei doppi apici, che permettono di interpretare come variabile ciò che racchiudono; senza di esse, all'attributo `src` del primo esempio verrebbe assegnata la stringa `"itemImageUrl"` come valore fisso.

Questi due tipi di binding possono essere combinati per propagare modifiche in entrambe le direzioni, come nell'esempio proposto dalla guida di Angular

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
```

per reagire ad un evento di un componente annidato e modificare la dimensione del testo: questa notazione consente modifiche in entrambe le direzioni fra componente padre e figlio. L'uso di entrambe le parentesi tonde e quadre rappresenta proprio la combinazione dei due tipi di binding ed è nota come Two-way binding [29]: la linea appena proposta può essere infatti riscritta come

```
<app-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></app-sizer>
```

3.2.4 Dipendenze

Angular semplifica notevolmente la risoluzione delle dipendenze, tramite un meccanismo detto `injector` che tiene memoria di tutte le classi marcate come `@Injectable`: una volta registrato un servizio, è sufficiente che un componente aggiunga nel proprio costruttore un parametro appartenente a quella classe perché l'injector li colleghi. È possibile dichiarare i servizi in qualunque punto dell'albero dei componenti, in modo che ogni componente veda solo i servizi dichiarati in `directory` che siano sue antenate; ciò avviene perché l'injector di Angular funziona in modo gerarchico, cercando una classe che soddisfi una richiesta di servizio a partire dalla cartella dove il componente è dichiarato per poi muoversi verso l'alto fino a quando non trova una classe adatta o non arriva alla radice. Tenendo conto di questo meccanismo è possibile sistemare i propri servizi in punti strategici della propria applicazione, in modo da farli condividere a più componenti o renderli esclusivi solo ad alcuni [30].

3.2.5 Routing

Persino il routing in Angular è realizzato tenendo conto della struttura a componenti: esiste infatti un modulo di routing a cui è possibile passare coppie di oggetti che ad ogni path facciano corrispondere un componente: il componente della route arrivata verrà visualizzato nel componente padre in corrispondenza del tag `<router-outlet>`, che è una direttiva speciale di Angular. Oltre ad una dichiarazione intuitiva delle route, Angular fornisce anche la possibilità di attivare alcuni percorsi solo a patto che determinate condizioni siano rispettate (utente che ha eseguito l'accesso, utente con permessi specifici, ...), di servirsi del lazy loading (caricare un modulo solo quando è necessario per ridurre il tempo di caricamento iniziale dell'applicazione) e di annidare le route. Esistono persino eventi associati all'uso dei router che permettono di controllare anche questo aspetto della vita dell'applicazione [31].

3.2.6 Direttive

Angular mette a disposizione dei programmatori diverse direttive, che sono raggruppabili in tre principali categorie [32]:

Componenti usate nel template, codice HTML di un componente, indicano che in quel punto va inserito un componente. Sono rappresentate dal nome del componente fra parentesi angolari e sono la direttiva più comune

Di attributo Modificano le proprietà di componenti o di altre direttive agendo sull'HTML. Ne sono esempi `ngClass` ed `ngStyle`, che aggiungono o rimuovono rispettivamente classi CSS ed HTML, ed `ngModel`, che viene utilizzato con il binding bidirezionale per aggiornare in tempo reale campi come gli input testuali di un form mentre l'utente li modifica

Strutturali modificano la struttura del DOM, aggiungendo, rimuovendo o moltiplicando gli elementi. `ngIf` mostra blocchi di codice condizionalmente, `ngFor` genera un oggetto per ogni elemento di una lista ed `ngSwitch` genera elementi diversi a seconda del valore di un'espressione.

Esiste anche la possibilità di creare direttive personalizzate: è sufficiente marcare con `@Directive` una classe che contenga dei metodi che permettano di reagire a determinati eventi perché quella classe diventi una direttiva applicabile ad un qualunque elemento del DOM [33].

3.2.7 Altre caratteristiche

Moltissime operazioni in Angular possono essere eseguite tramite Angular CLI, la sua interfaccia da linea di comando installabile con npm: è possibile ad esempio creare una nuova applicazione già inizializzata con `ng new` o compilarla con `ng build`. Anche la generazione di un componente può essere effettuata da terminale con il comando `ng add`, che può essere corredato da flag che indichino se generare o meno un file di test. Questi comandi permettono di risparmiare molto tempo, sostituendo la creazione tramite interfaccia grafica di cartelle e file e il loro riempimento con codice ripetitivo (dichiarazioni di classi sempre uguali eccezion fatta per il nome) con un semplice comando. Angular CLI permette anche di gestire le versioni delle librerie in modo simile ad npm [34].

Angular offre anche supporto per lo sviluppo di applicazioni multilingue, strumenti e librerie di testing come Karma e Jasmine, librerie per la programmazione asincrona come RxJS e form reattivi in grado di eseguire validazioni dei campi.

Quelle elencate in queste pagine sono solo alcune delle funzionalità che rendono Angular un framework versatile e completo, utilizzato da compagnie come IBM, Deutsche Bank, PayPal, Forbes e Samsung [35].

Capitolo 4

Presentazione degli strumenti ausiliari: Figma e LangChain

Se il linguaggio Angular presentato nel capitolo precedente era ciò che questo lavoro mira a generare, quanto introdotto in questo capitolo è ciò su cui esso si basa. In queste pagine si parlerà infatti di Figma, lo strumento che verrà utilizzato dall'utente per il design delle pagine web, e di LangChain, il framework principale per sviluppare applicazioni basate sui modelli linguistici di grandi dimensioni.

4.1 Figma

Nata nel 2011 come idea di Dylan Field ed Evan Wallace alla Brown University, Figma ha avuto una crescita rapidissima, facendo ottenere a Field 100.000 dollari per lo sviluppo del suo progetto appena un anno più tardi [36]. Nel 2015 la startup aveva già raggiunto 18 milioni di dollari di finanziamenti ed era pronta a presentare il suo prodotto in anteprima, per poi rilasciarlo pubblicamente l'anno successivo [37].

Ciò che ha permesso uno sviluppo così fulmineo è stato l'approccio innovativo di Figma, che lo ha subito reso competitivo anche rispetto a prodotti di compagnie già affermate come Adobe. Se gli altri programmi utilizzati dai designer di pagine web erano per lo più di tipo desktop, Figma è invece utilizzabile direttamente dal browser: ciò permette una collaborazione in tempo reale fra più membri dello stesso gruppo di lavoro. Questo approccio di aggiunta di componenti e di commenti in tempo reale ha segnato una svolta rispetto al metodo di lavoro precedente, un processo piuttosto lungo che comprendeva più applicazioni e richiedeva tempi relativamente lunghi. La prassi era infatti quella di creare localmente il prototipo della pagina (quasi sempre con Photoshop) per poi condividerlo su strumenti come Dropbox e attendere che i colleghi ripetessero la stessa operazione prima di poter vedere i loro commenti: la forza di Figma sta proprio nel permettere di fare tutto ciò in tempo reale servendosi di una singola applicazione web.

Sviluppare Figma come applicazione web permette di utilizzarlo indipendentemente dal proprio sistema operativo (a differenza di Sketch e dei sistemi Adobe, principali concorrenti [38]) rientra nell'ottica del suo creatore, il cui intento era offrire al mondo dei designer uno strumento analogo a Google Docs, piattaforma le cui potenzialità lo avevano sempre affascinato [37].

Altra potenzialità dei design basati su Figma è il servirsi di immagini vettoriali, che si differenziano da quelle tradizionali (dette raster) in quanto invece di basarsi su una griglia di pixel predefinita sfrutta calcoli matematici per generare le figure. Questo tipo di grafica si presta bene

alla grafica computerizzata perché permette di scalare le immagini e i caratteri indefinitamente: se aumentando le dimensioni di un'immagine raster si va inevitabilmente incontro a sgranatura, ciò non avviene se la stessa immagine è salvata vettorialmente, perché in questo caso le curve vengono ricalcolate e adattate alla nuova dimensione. Oltre alla ridimensionabilità, anche la capacità di modifica beneficia dell'approccio vettoriale: è infatti possibile manipolare una linea a cui ne siano sovrapposte altre, cosa che in un'immagine raster richiederebbe sforzi notevoli. Figma sfrutta queste immagini per permettere all'utente di realizzare loghi scalabili in qualunque dimensione e di svolgere operazioni booleane tra forme, creando ad esempio loghi complessi dall'unione di forme più semplici o ritagliando forme per sottrazione. Per adattare la pagina al formato dello schermo è stata inoltre sviluppata la funzione di auto layout, che ridimensiona gli oggetti in modo intelligente basandosi, sempre tramite calcoli matematici, sulla dimensione della pagina o del container in cui un oggetto si trova.



Figura 4.1. Il logo di Figma.

Figma offre varie modalità di lavoro:

Modalità design permette di realizzare manualmente il layout delle pagine, senza necessità di scrivere codice (figura 4.2). Si basa su meccanismi come trascinamento, selezione e raggruppamento di oggetti e permette di combinare componenti di base come campi di testo, rettangoli, immagini e linee. È responsabile dell'aspetto esteriore della pagina.

Modalità prototipo consente di collegare con delle frecce le varie schermate create in modalità design per rappresentare il flusso logico che le collega (figura 4.3). Una freccia che collega un tasto di una finestra alla finestra successiva ad esempio indica che la pressione di quel pulsante porterà alla visualizzazione dell'altra schermata. Molto utile la possibilità di mostrare questo flusso sotto forma di presentazione in cui è possibile cliccare i pulsanti e mostrare il progetto a colleghi o clienti.

Modalità codice modalità molto utile che permette di colmare in parte il divario fra designer e programmatori, in quanto mostra informazioni di base sullo stile di un componente (figura 4.4). Si possono selezionare vari formati, a seconda che si intenda lavorare in ambiente web (CSS), Android o iOS.

Un altro aspetto che contribuisce alla versatilità di Figma è la possibilità di creare componenti, ovvero elementi riutilizzabili che permettono di mantenere la consistenza stilistica attraverso schermate e progetti diversi. Qualunque utente può creare un componente a partire da qualsiasi oggetto nel proprio progetto, da semplici pulsanti a strutture più complesse, per poi riusarlo in altri progetti. È anche possibile creare librerie di componenti e riutilizzare le librerie di altri utenti [39].

Una maniera comoda di organizzare i componenti è quella di realizzare delle varianti, versioni

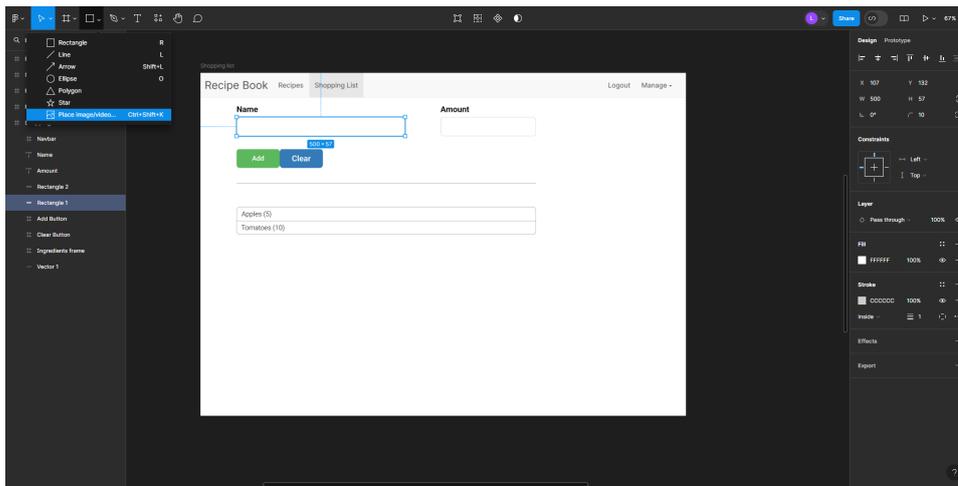


Figura 4.2. Modalità design: è possibile aggiungere componenti di vario genere.

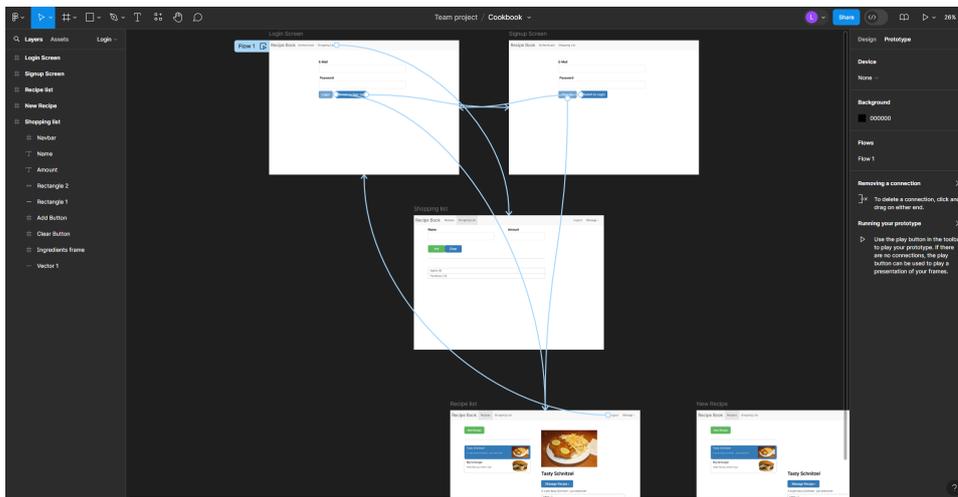


Figura 4.3. Modalità prototipo: è possibile rappresentare il flusso logico fra schermate tramite frecce.

diverse dello stesso componente che pur distinguendosi per alcune proprietà mantengono una struttura di base comune (come ad esempio le proprietà `selected`, `primary`, ecc dei pulsanti delle pagine web). Figma permette di realizzare le diverse versioni del componente in questione in una stessa pagina per poi evidenziarle con il cursore e raggrupparle in un unico componente con più varianti. Una volta raggruppate le varianti si ottiene un unico componente con proprietà aggiuntive che sono un'esclusiva degli oggetti con varianti: modificando i valori di questi attributi si ottengono le diverse varianti [40].

Un'altra possibilità di personalizzazione di semplice utilizzo è costituita dagli stili. Applicare uno stile permette di modificare molte proprietà relative al testo (dimensione, colore, font e

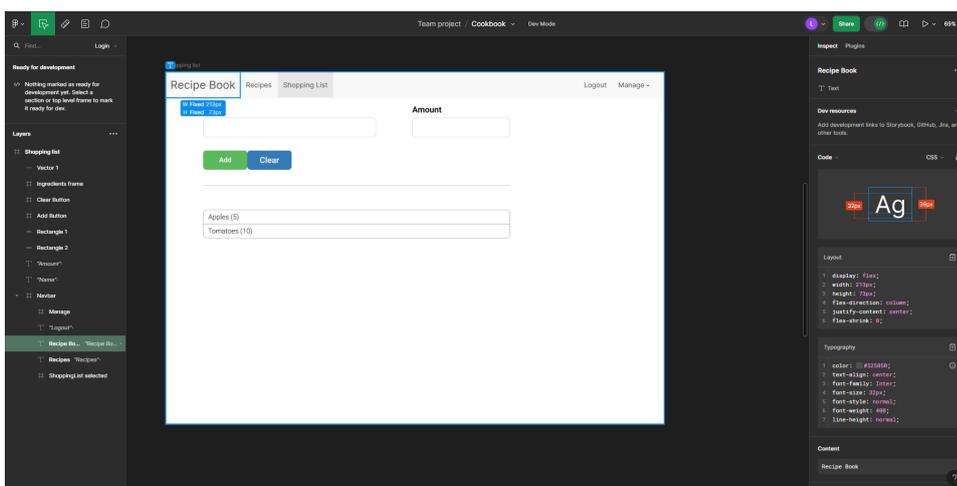


Figura 4.4. Modalità codice: è possibile vedere informazioni di base sul CSS del componente selezionato.

interlinea), ai colori (colore di riempimento e suo gradiente, bordo), agli effetti (ombre esterne o interne agli oggetti e sgranature per suggerire elementi in secondo piano) e alle griglie secondo cui allineare i componenti. In modo simile alle varianti e ai componenti, è possibile definire uno stile a partire da quello degli oggetti già esistenti o crearne uno nuovo da zero e si possono importare o condividere stili per mezzo delle librerie [41].

4.2 Langchain

LangChain è stato lanciato nell'ottobre 2022 da Harrison Chase, che all'epoca lavorava per Robust Intelligence (azienda che si occupa di sicurezza in ambito IA) e ha avuto un'ascesa rapidissima. La grande quantità di tutorial su YouTube lo ha reso popolare e di semplice utilizzo, mentre il fatto di essere open source ha permesso di ricevere contributi da centinaia di utenti GitHub. Il successo riscosso da questo framework è dovuto alla sua capacità di creare applicazioni basate su modelli linguistici di grandi dimensioni, comportandosi come una generica interfaccia che permette di utilizzare quasi tutti gli LLM più noti combinandoli con fonti esterne e persino fra di loro, realizzando ad esempio applicazioni che sfruttino il modello linguistico più adatto ad una particolare funzionalità.

L'unità fondamentale su cui Langchain si basa sono le catene, astrazioni che riassumono passaggi e concetti comuni a tutte le applicazioni basate su LLM; collegando diverse catene si può realizzare un'applicazione che elimina alcuni gradi di libertà che ad uno specialista del settore potrebbero servire, permettendo però ai neofiti di sperimentare e avvicinarsi al mondo dell'intelligenza artificiale e agli esperti di realizzare prototipi in breve tempo.

Altro componente interessante sono i document loader, che permettono di integrare i modelli con altre fonti di dati. Come vedremo nel capitolo 6.4, spesso gli LLM hanno dati di training che si fermano solo fino ad un certo punto nel tempo, oppure non includono alcuni testi specializzati richiesti dall'applicazione: l'aggiunta di un documento permette di ovviare a queste mancanze, integrando documenti in svariati formati (al momento della scrittura di questa tesi 170) che includono immagini, testo, contenuti web, database e persino servizi di archiviazione



Figura 4.5. Il logo di LangChain.

online come Dropbox e Google Drive. Uno degli usi principali di questi componenti è la realizzazione di chatbot che rispondano a domande basandosi sulla tecnica della Retrieval Augmented Generation (RAG): questa tecnica consiste nel suddividere una sorgente di dati (proveniente da un document loader) in documenti più piccoli, per poi generare degli indici. Una volta ricevuta la richiesta dell'utente si effettua una ricerca numerica basata sugli indici per trovare i documenti più semanticamente simili all'input fornito in modo da usarli per la generazione della risposta [42]. Di quest'ultimo compito si occupano speciali interfacce dette retriever, che ricevono in input una query in forma di stringa e restituiscono la lista dei documenti più pertinenti; come per i documenti, esistono varie classi in grado di svolgere la funzione di retriever, in modo da adattare la propria applicazione al tipo di documenti, al modo in cui le informazioni all'interno di essi sono distribuite e all'eventuale presenza di metadati [43].

Un'interessante applicazione di questi strumenti è la realizzazione dei chatbot che sempre più aziende mettono a disposizione sui propri siti per rispondere alle domande dei propri clienti: è infatti possibile passare ad un modello linguistico dei documenti specifici (manuali di istruzioni, regolamenti, cataloghi di prodotti, ...) in modo da renderlo in grado di rispondere a domande di base e alleggerire il carico di lavoro dell'assistenza clienti svolta dagli umani.

Per combinare tutti gli strumenti che mette a disposizione, LangChain ha sviluppato un proprio metodo di concatenazione noto come LCEL (LangChain Expression Language), che offre potenzialità come riduzione del time-to-first-token (tempo impiegato per produrre il primo frammento di output), supporto per API sincrone e asincrone (che permette di riutilizzare lo stesso codice in fase di produzione e di test), parallelizzazione automatica di compiti che possono essere svolti in parallelo, inclusione di fallback nella catena (meccanismo simile a quello dei blocchi try-catch di numerosi linguaggi di programmazione che consiste nel passare una funzione di callback da chiamare in caso di errore lungo la catena), accesso ai risultati intermedi per scopi di debug o di informazione dell'utente in tempo reale, validazione di input e output mediante schemi tipo JSON e mantenimento di un log dei passaggi intermedi. Tutte queste funzionalità possono essere concatenate a piacimento per mezzo di un operatore pipe `|` analogo a quello dei sistemi Unix, che manda l'output di un elemento all'input di quello successivo.

Nella figura 4.6 è mostrato un esempio di catena che collega in serie il prompt con i messaggi di sistema e utente (vedi paragrafo 5.1), un LLM e un parser, oggetto che può manipolare in vari modi l'output del modello linguistico (in questo esempio convertendolo in stringa). Collegare questi elementi tramite pipe equivale a passare il prompt al modello di intelligenza artificiale, che produrrà un output sulla base dei messaggi per poi farlo formattare al parser: ciò che verrà ritornato dalla funzione sarà quindi una stringa contenente la risposta generata dal LLM. Esistono diversi tipi di parser, fra i quali liste separate da virgola, JSON, XML, YAML e persino wrapper che in caso di errore del parser più interno chiedono ad un LLM di risolvere il problema [45].

Un'altra funzionalità che in seguito ci tornerà molto utile è quella di mantenere una memoria dei messaggi precedentemente scambiati. Questa capacità è indispensabile per la realizzazione di chatbot e LangChain mette a disposizione numerosi metodi per farlo:

```
def call_string_output_parser():
    prompt = ChatPromptTemplate.from_messages([
        ("system", "Tell me a joke about the following subject"), ("human", "{input}")
    ])

    parser = StrOutputParser()
    chain = prompt | model | parser

    return chain.invoke({"input": "dog"})
```

Figura 4.6. Esempio di utilizzo del LCEL.

salvataggio di tutti i messaggi precedenti la strategia più semplice, che useremo per la realizzazione del nostro chatbot. Consiste nel salvare tutti i precedenti messaggi dell'utente e dell'IA in un'istanza della classe `ChatMessageHistory`, che tramite i metodi `add_user_message()` e `add_ai_message()` permette di aggiornare la cronologia. Un metodo più semplice e altrettanto valido consiste nel salvare i messaggi in un vettore. Esistono anche classi come `RunnableWithMessageHistory` che accodano i messaggi alla cronologia in automatico.

salvataggio di alcuni messaggi è possibile aggiungere la funzione `trim_messages()` alla catena che costituisce l'applicazione. Questa funzione permette di aggiungere vari parametri per specificare se si voglia mantenere il prompt di sistema, se si vogliono salvare i primi o gli ultimi messaggi e quanti messaggi lasciare. Molto utile in casi in cui si voglia realizzare un'applicazione che non gravi troppo sulla memoria del dispositivo, quando si hanno limiti sulla finestra di contesto o quando non si vuole confondere l'intelligenza artificiale perché magari si cambia discorso frequentemente.

sintesi dei messaggi precedenti strategia interessante per risparmiare memoria, consiste nel chiamare una seconda istanza del modello linguistico per far generare un riassunto dei messaggi precedenti, che poi verrà passato al modello responsabile della conversazione con l'utente.

Per espandere ancora di più le proprie potenzialità, LangChain mette a disposizione degli sviluppatori LangSmith, strumento che permette di monitorare le prestazioni delle applicazioni basate su modelli linguistici di grandi dimensioni. Pur offrendo la possibilità di sviluppare prototipi di base in breve tempo, ciò che più rende interessante LangSmith sono le sue capacità di debug e di monitoraggio:

debugging gli strumenti di debug di LangSmith permettono, oltre alla classica esecuzione passo passo, di testare i propri programmi in vari scenari per comprendere il suo comportamento nei diversi ambienti e di tenere traccia di ogni evento grazie a file di log dettagliati

monitoraggio è possibile visualizzare graficamente il flusso dei dati fra un componente e l'altro dell'applicazione, tenendo traccia del numero di esecuzioni e della durata di ognuno e aiutando a comprendere meglio i vari flussi di dati. Ciò può essere utile anche a scopo di debug per vedere quali siano i punti in cui l'applicazione genera errori. Si può inoltre tenere traccia di tempi di risposta, frequenza degli errori e utilizzo delle risorse, individuando così eventuali punti che rallentino particolarmente l'applicazione e ricevendo

notifiche quando si supera un certo livello di prestazione negativa (risposte eccessivamente lente o errori troppo frequenti). Langsmith permette inoltre di raccogliere feedback dall'utente in merito a vari criteri di valutazione e assegna un punteggio alle risposte in base a criteri di valutazione predefiniti (correttezza, misoginia, utilità, coerenza, concisione, ecc) o creati dallo sviluppatore tramite costruzione di catene di valutazione.

Tutte le metriche appena descritte possono essere monitorate grazie a grafici che ne monitorano l'andamento nel tempo

In sintesi, LangSmith permette di sfruttare al meglio un modello linguistico, se affiancato a LangChain: se il primo da solo permette di creare solo catene poco personalizzabili, il lavoro congiunto dei due permette di unire la complessità delle catene di LangChain alla possibilità di perfezionare il programma offerte dagli strumenti di LangSmith, combinando il meglio dei due strumenti.

Dopo aver introdotto le caratteristiche di LangChain, passeremo nel prossimo capitolo a vedere una sua applicazione pratica con la realizzazione di un chatbot.

Capitolo 5

Sviluppo di un chatbot in Python

In questo capitolo discuteremo finalmente la scrittura del programma su cui questa tesi si basa, descrivendone la struttura e le principali librerie e integrazioni utilizzate.

5.1 Presentazione ambiente e librerie

Quando si parla di intelligenza artificiale e programmazione, la scelta ricade quasi automaticamente su Python: data la presenza di librerie che permettono di sfruttare le API di OpenAI e della gran varietà di integrazioni per passare a queste ultime documenti in vari formati, la stessa scelta è stata effettuata anche per questo lavoro. In questo paragrafo esamineremo le classi e librerie più importanti di cui si servirà il nostro programma.

ChatOpenAI Definito in `langchain_openai`. Permette di definire i parametri dell'assistente che utilizzeremo, come il modello (nel nostro caso `gpt-4-turbo`) e la temperatura (livello di creatività dell'IA, che determina se sarà più o meno deterministica nel generare risposte). Per utilizzare i servizi di OpenAI è necessario fornire anche una chiave, ma ciò avviene mediante l'uso della chiave d'ambiente `OPENAI_API_KEY`.

ChatPromptTemplate Definito in `langchain.prompts.chat`. La classe più importante del programma. Permette di creare un prompt a cui passare diversi tipi di messaggi, come un prompt di sistema che indica il contesto generale di lavoro ("Sei un assistente cordiale il cui compito è fornire risposte simpatiche a domande di argomento culinario"), lo storico dei messaggi precedenti e l'input passato dall'utente. Oltre ai messaggi è possibile dare in input anche dei documenti di varia natura, di cui ci serviremo per fornire informazioni aggiuntive sull'output desiderato; in particolare useremo file Figma, testuali e di immagine.

UnstructuredImageLoader Definito in `langchain_community.document_loaders`. Permette di passare al `ChatPromptTemplate` informazioni su un file immagine, che useremo per fornire all'intelligenza artificiale uno screenshot della pagina per cui vorremmo scrivere il codice.

TextLoader Definito in `langchain_community.document_loaders`. Funzionamento pressoché analogo a quello della classe precedente, ma improntato sui file testuali. Entrambe le classi richiedono di invocare il metodo `load()`, che ritorna una lista di oggetti di tipo `Document` che verranno passati al prompt tramite il parametro `context`.

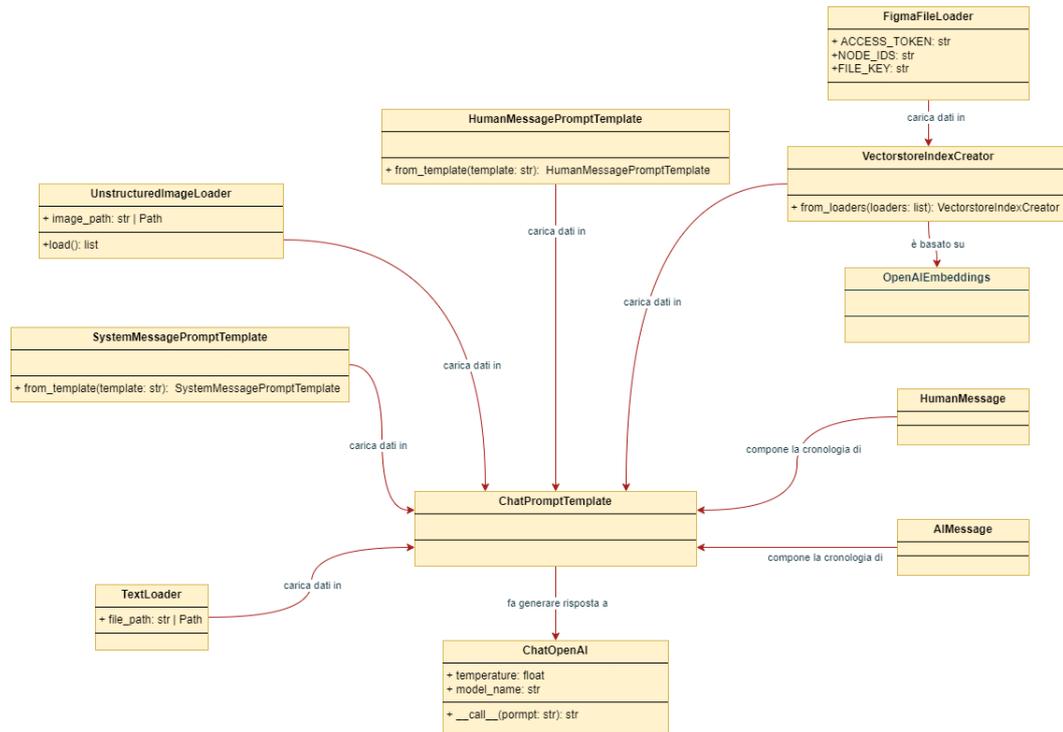


Figura 5.1. Diagramma delle classi illustrate in questo paragrafo

FigmaFileLoader Definito in `langchain_community.document_loaders.figma`. Lo scopo è simile a quello dei loader precedenti, ma permette di passare informazioni sul prototipo Figma della pagina web. Deve ricevere un token di accesso, l'id del progetto Figma e l'id della specifica pagina all'interno del progetto; per praticità passeremo questi tre valori tramite le variabili d'ambiente `ACCESS_TOKEN`, `FILE_KEY` e `NODE_IDS`. A differenza degli altri due loader richiede la creazione di un retriever, classe il cui metodo `invoke()` riceve in input il messaggio dell'utente, in modo da individuare i documenti più utili al soddisfacimento della richiesta.

HumanMessage ed AIMessage Definiti in `langchain_core.messages`. Nel vettore della cronologia dei messaggi rappresentano rispettivamente un messaggio dell'interlocutore e una risposta dell'agente virtuale.

HumanMessagePromptTemplate e SystemMessagePromptTemplate Definiti in `langchain_core.prompts.chat`. Nella creazione del prompt della chat rappresentano rispettivamente il prompt di sistema introdotto poco fa e l'input ricevuto dall'utente.

5.2 Descrizione del codice

Passiamo ora a descrivere la struttura del codice in maniera più approfondita.

5.2.1 Inizializzazione

L'inizializzazione richiesta è minima e si limita ad operazioni quali import di classi provenienti da moduli esterni, definizione dei colori per distinguere da terminale i messaggi dell'utente da quelli del chatbot e caricamento delle variabili d'ambiente. L'unica operazione degna di essere menzionata è la costruzione di un indice a partire dal documento Figma, che porterà alla creazione del retriever descritto nel paragrafo precedente.

5.2.2 Main

Il main del programma è molto semplice, essendo costituito da un ciclo while che gestisce ogni messaggio dell'utente, facendo generare una risposta dalla funzione `generate_code()` ad ogni iterazione. Per interrompere il programma è sufficiente digitare "e" o "E" al posto di porre una domanda. Qui avviene anche la gestione della memoria dei messaggi precedenti e la visualizzazione su terminale della risposta prodotta.

5.2.3 Funzione di generazione del codice

La sezione più consistente del programma, nonché quella che svolge la parte principale del lavoro, è la funzione `generate_code()`, che riceve l'input dell'utente e la lista con lo storico dei messaggi precedenti.

Questa funzione definisce innanzitutto il prompt di sistema (fisso) e quello umano (ricavato ogni volta dall'input da tastiera), per poi procedere alla definizione di parametri come temperatura e modello. Seguono il caricamento dei file accessori (testo e immagine) e il recupero dei documenti Figma più importanti, che avviene tramite ricerca basata sull'input dell'utente nell'indice generato in fase di inizializzazione. I tre tipi di documento vengono caricati in un unico vettore che costituirà il contesto della chat.

Dopo queste preparazioni si passano al `ChatPromptTemplate` i due prompt e la cronologia, per poi invocare il modello e generare finalmente una risposta basata su questi ultimi e sul contesto. Il risultato così generato viene passato al main, che lo mostra all'utente e ne tiene memoria per le prossime iterazioni. Il mantenimento della cronologia nel main è essenziale perché permette di far correggere all'agente virtuale eventuali incomprensioni o errori di programmazione da esso commessi.

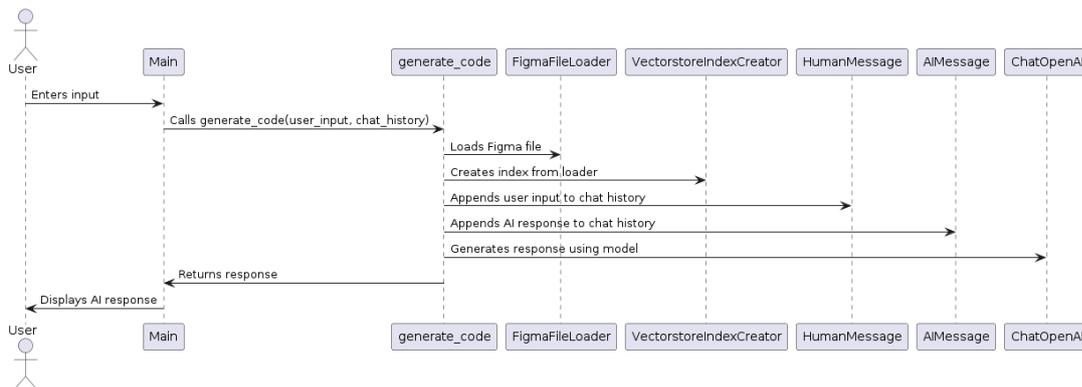


Figura 5.2. Diagramma di sequenza che riassume le interazioni descritte in questo paragrafo

Capitolo 6

Risultati ottenuti e osservazioni

La stesura del programma descritto nel capitolo precedente ha richiesto diversi tentativi e la struttura appena presentata non è che l'ultimo di questi tentativi. In questa sezione esploreremo, oltre ai risultati del codice finale che verranno analizzati nelle conclusioni, quelli dei fallimenti che l'hanno preceduto, con la speranza di fornire una guida a chi volesse tentare di portare avanti questo lavoro.

In tutti i tentativi si è richiesto di generare la stessa pagina, in modo da poter confrontare i risultati ottenuti. La pagina rappresenta un immaginario sito in cui sia possibile tenere traccia di una lista della spesa (figura 6.1) ed è costituita da un form in cui inserire nome e quantità del prodotto e da una lista di ingredienti da acquistare. Nel form sono contenuti due pulsanti aventi rispettivamente la funzione di aggiungere un nuovo prodotto e di eliminare tutti i prodotti o soltanto quello corrente (lo stato di questo pulsante deve cambiare a seconda che un elemento nella lista sia selezionato o meno). Nella parte superiore della pagina è presente una navbar il cui scopo è solo estetico, ma per cui si è rivelato sorprendentemente difficile ottenere risultati soddisfacenti. Nel prompt iniziale viene richiesto di soddisfare tutti questi requisiti, oltre a quello di creare una pagina visualmente simile a quella proposta nel prototipo Figma e nell'immagine.

I test si sono svolti per gradi, partendo dalla generazione di componenti semplici per poi passare a quella di pagine HTML statiche e arrivare infine a pagine dotate di script e ad applicazioni Angular vere e proprie: la struttura dei paragrafi seguirà questo percorso.

Recipe Book Recipes Shopping List Logout Manage -

Name Amount

Add Clear

Apples (5)
Tomatoes (10)

Figura 6.1. Screenshot della pagina generata con Figma che si cerca di replicare

6.1 HTML e CSS di singoli componenti

6.1.1 Tasto add

Le prime versioni del codice non presentavano struttura ciclica: si inseriva il prompt utente direttamente nel file Python e si chiedeva di generare il codice HTML di un componente. Il primissimo tentativo è stato quello di generare il pulsante add, con prompt di sistema mostrato nella figura 6.2 e "Code the add button" come messaggio user.

```
"""You are expert coder Jon Carmack. Use the provided design context to create idiomatic HTML/CSS code as possible based on the user request. Everything must be in one file and your response must be directly renderable by the browser. Figma file nodes and metadata: {context}"""
```

Figura 6.2. Versione base del prompt di sistema, presa dal sito di LangChain [48].

Anche con istruzioni così semplici e con un componente piccolo da creare, è apparsa subito evidente una caratteristica tipica dell'intelligenza artificiale: il non determinismo dell'output (vedi figura 6.3). Anche con istruzioni precise e un compito che offre poco margine di libertà non è scontato ottenere due volte lo stesso risultato, proprio come accadrebbe chiedendo ad un essere umano di disegnare pi volte lo stesso oggetto.



Figura 6.3. Alcuni tentativi di replicare il tasto add: notare come né il font, né il contorno né l'uso delle maiuscole siano consistenti fra un risultato e l'altro.

6.1.2 Navbar

Il passo successivo è stato tentare di far ricreare all'IA un componente più complesso, ovvero la navbar, ma i risultati erano inizialmente molto deludenti (figura 6.4). Per quanto si provasse a generare messaggi utente precisi, non si otteneva molto con un solo tentativo: un'idea per migliorare il risultato è stata quella di far correggere al modello i propri stessi errori, rendendolo in grado di ricordare quanto generato in precedenza. Il programma si è trasformato così in una sorta di chatbot in cui però solo l'utente poteva scrivere dei messaggi: il codice generato veniva direttamente stampato su un file HTML che l'utente poteva poi aprire, solo occasionalmente l'intelligenza artificiale raramente aggiungeva commenti o spiegazioni sul proprio lavoro (figura 6.5).



Figura 6.4. Le prime navbar generate non avevano né le giuste proporzioni né i giusti colori. Nella prima si nota addirittura un tentativo di aggiungere un'immagine che però non era stata richiesta, nella seconda è presente la scritta "Amount", presa da Figma anche se fa parte del form e non della navbar.

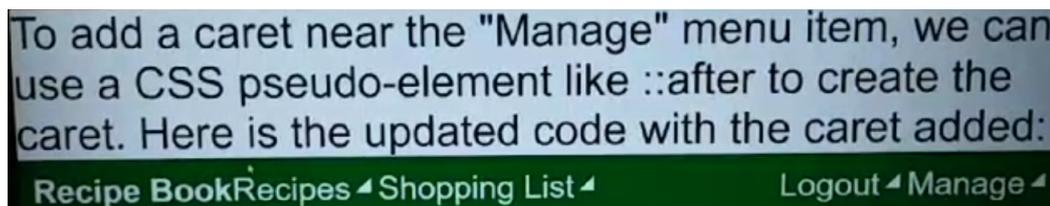


Figura 6.5. Dopo aver chiesto di aggiungere un caret (triangolino rivolto verso il basso che suggerisce all'utente la presenza di un menù a tendina) vicino alla scritta "Manage", il chatbot la aggiunge a tutti i componenti, convinto però di averlo messo soltanto vicino a quello desiderato.

La semplice aggiunta di un ciclo while nel main, unita all'implementazione di una chat con memoria si è rivelata una buona strategia, che ha permesso di ottenere risultati decisamente più soddisfacenti (figura 6.6). Anche in questo caso il primo messaggio dell'utente era molto generico ("Code the whole navbar") e il prompt di sistema era identico a quello dell'esempio dei tasti; con i messaggi successivi si andavano a richiedere modifiche nella posizione, nella spaziatura o nell'ordine dei componenti. L'errore più frequente era il non capire quali componenti posizionare a destra e quali a sinistra. Questa difficoltà rimarrà una costante anche nelle versioni successive del programma.



Figura 6.6. La navbar meglio riuscita, molto fedele all'originale.

6.2 HTML e CSS dell'intera pagina

La trasformazione del programma in chat ha finalmente permesso di generare il codice per l'intera pagina: le prime iterazioni continuavano a generare pagine molto strane (figura 6.7), probabilmente basate su ciò che l'IA intuiva dai componenti di testo del file Figma. Concentrandosi su un componente per volta si ottenevano buoni risultati, ma erano necessarie sempre molte iterazioni. Non capitava mai che il solo messaggio utente "Code the whole page" fosse sufficiente ad ottenere una fedele replica dell'originale: si dovevano prima aggiungere specifiche sulla navbar e poi in seguito chiedere di generare il form e la lista. In media erano richiesti tre messaggi per macro-componente (navbar, form e lista), più altri cinque messaggi circa per aggiustamenti stilistici vari (prevalentemente allineamenti e spaziatura); nei casi peggiori invece è stato addirittura raggiunto il numero massimo di token¹ consentito da gpt-3.5.

Da questo punto in poi ha iniziato a manifestarsi il limite costituito dalla finestra di contesto del modello utilizzato: questa grandezza, che indica il numero massimo di token di cui sia possibile mantenere memoria, pone un limite a quanto dei precedenti errori o istruzioni ricevute possa essere ricordato dal modello, perciò può verificarsi che imprecisioni già commesse vengano ripetute o che certe correzioni vengano dimenticate e sia necessario ripeterle. Nelle API di

¹Un token rappresenta un pezzo di parola: nella lingua inglese è assimilabile a 4 caratteri, perciò una frase o due corrispondono a circa 30 token. Molta influenza hanno anche la lingua, la posizione della parola all'interno della frase e persino l'aver l'iniziale maiuscola o minuscola. Ad ogni token corrisponde un numero: più il token ha probabilità di apparire, più il numero sarà basso [46].

Recipe Book

Name Enter your name
Ingredients Enter ingredients
Instructions Enter instructions

Figura 6.7. I primi output erano così diversi dal risultato atteso da far pensare che l'ia stesse improvvisando. Qui ad esempio "Name" viene interpretato come nome dell'utente, invece che dell'ingrediente.

OpenAI usate nei primi test, il limite era di 16.000 token circa, ma passando da gpt-3.5-turbo a gpt-4-turbo, si è arrivati a ben 128.000 token.

I apologize for the confusion. I will adjust the button sizes to be narrower and remove the margin between list items while restoring the grey borders. Here's the updated HTML and CSS:

Recipe Book	Recipes	Shopping List	Manage	Logout
-------------	---------	---------------	--------	--------

Name: Amount:

- Milk (2)
- Bread (1)
- Butter (1)
- Eggs (12)
- Cheese (1)

Figura 6.8. Anche quando la pagina era graficamente simile all'originale, alcune scritte erano "inventate", invece che prese da Figma (in questo caso i nomi degli ingredienti). Notare come posizione e spaziatura nella navbar continuano ad essere problematici.

6.3 HTML e CSS con JavaScript

Con le specifiche descritte finora si riusciva a volte ad ottenere una pagina soddisfacente come quella della figura 6.9, ma nella maggior parte dei tentativi erano invece richieste molte interazioni perché ciò accadesse e il programma si interrompeva a causa del raggiungimento del numero massimo di token consentiti dal modello.

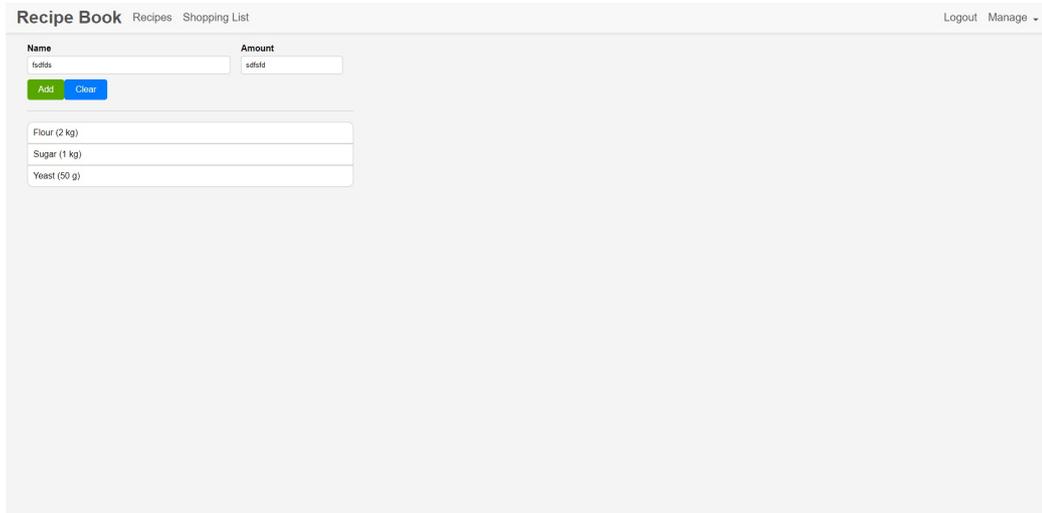


Figura 6.9. Un buon esempio di pagina, ottenuto però dopo numerose iterazioni.

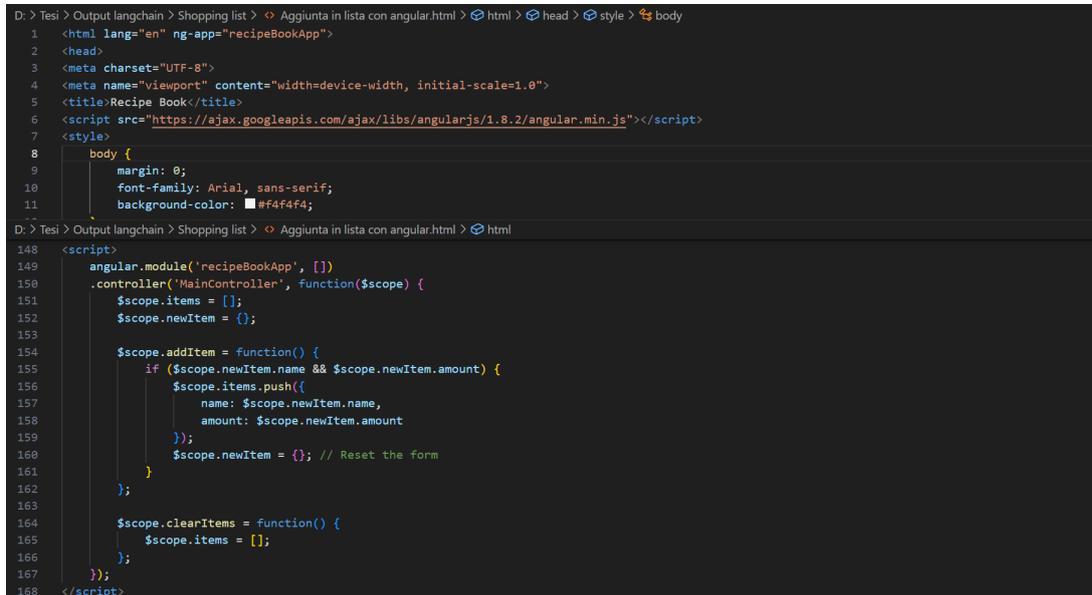
In alcuni casi però si otteneva un buon risultato dopo poche interazioni: avanzavano così abbastanza token da permettere di far inserire del codice JavaScript direttamente nel file HTML generato. Essendo abbastanza elementari (aggiunta e rimozione da una lista), le richieste venivano soddisfatte molto più in fretta di quelle di carattere stilistico; spesso bastava una sola iterazione per avere il comportamento richiesto. Si è scelto di iniziare con JavaScript per la sua semplicità, in modo da non sprecare token con compiti troppo gravosi per il modello.

6.4 HTML e CSS con AngularJS

Chiedendo di generare il codice Angular obiettivo del nostro lavoro al posto del più semplice JavaScript sembrava comunque funzionare tutto bene. Un esame più attento del codice generato mostrava però come il chatbot non inserisse script in Angular all'interno del codice, ma bensì codice scritto in AngularJS. Questo linguaggio, pur essendo ancora largamente sfruttato da aziende del calibro di Amazon, Microsoft e persino nei portali online della Deutsche Bank [47], è obsoleto e non è l'Angular che ci si propone di utilizzare in questa tesi. Il 31 dicembre 2021 Google ha infatti dichiarato deprecato AngularJS, ma i dati di training di gpt-3.5 si fermavano al settembre di quell'anno. Al di là della deprecazione del linguaggio, i risultati finora ottenuti rimanevano comunque dei semplici file HTML con del CSS e uno script inserito per implementare funzionalità minime, qualcosa di molto lontano dalla complessità di un'applicazione Angular completa.

Per superare questi limiti si è aggiornato il modello utilizzato a gpt-4.0. La sua introduzione ha

portato diversi benefici che hanno permesso di arrivare alla stesura definitiva del programma, primi fra tutti la più ampia finestra di contesto e il maggior numero di token generabili.



```

D: > Tesi > Output langchain > Shopping list > Aggiunta in lista con angular.html > html > head > style > body
1 <html lang="en" ng-app="recipeBookApp">
2 <head>
3 <meta charset="UTF-8">
4 <meta name="viewport" content="width=device-width, initial-scale=1.0">
5 <title>Recipe Book</title>
6 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
7 </style>
8
9   body {
10     margin: 0;
11     font-family: Arial, sans-serif;
12     background-color: #f4f4f4;
13   }
14
D: > Tesi > Output langchain > Shopping list > Aggiunta in lista con angular.html > html
148 <script>
149   angular.module('recipeBookApp', [])
150     .controller('MainController', function($scope) {
151       $scope.items = [];
152       $scope.newItem = {};
153
154       $scope.addItem = function() {
155         if ($scope.newItem.name && $scope.newItem.amount) {
156           $scope.items.push({
157             name: $scope.newItem.name,
158             amount: $scope.newItem.amount
159           });
160           $scope.newItem = {}; // Reset the form
161         }
162       };
163
164       $scope.clearItems = function() {
165         $scope.items = [];
166       };
167     });
168 </script>

```

Figura 6.10. Due estratti di una pagina in cui l'IA ha preferito servirsi di AngularJS: nel primo si nota il link ad AngularJS 1.8.2, nel secondo si può notare la differenza rispetto all'Angular più moderno.

6.5 App Angular completa

Il cambio di modello dell'agente virtuale ha effettivamente risolto le difficoltà sopra citate, ma quella più grande permaneva. L'intelligenza artificiale continuava infatti a generare la pagina in modo abbastanza casuale, necessitando di indicazioni precise da parte dell'utente nonostante il file Figma venisse passato correttamente. La soluzione al problema è stata suggerita dallo stesso gpt-4.0: dopo avergli chiesto quanto effettivamente capisse del file Figma, la risposta è stata piuttosto sorprendente, come mostrato nella figura 6.11.

Di tutto il design creato con Figma, le uniche cose che sembrassero venire afferrate correttamente erano le parti testuali, e in effetti anche nei fallimenti precedenti era molto raro che le scritte fossero errate. Dall'esame di questa chat appare chiaro come mai la maggior parte delle volte la struttura della pagina sembrasse "inventata": non avendo altre informazioni al di fuori delle scritte, l'unica cosa che il nostro modello potesse fare era provare a dedurre la struttura della pagina da quanto contenuto nei campi testuali.

Per ovviare a questo problema è stata d'aiuto l'ampia disponibilità di integrazioni messe a disposizione da LangChain, delle quali quella per Figma era solo una delle tante: in particolare si è deciso di sfruttare la classe `UnstructuredImageLoader`, di cui discusso al paragrafo 5.1. L'introduzione di questa libreria per accompagnare al file Figma uno screenshot della pagina desiderata, ha migliorato di molto le prestazioni dell'assistente virtuale.

```

YOU: Just answer a question: can you see the figma document I passed you? It doesn't seem so, to me
AI: I'm sorry for any confusion, but as an AI, I don't have the ability to directly view or interpret Figma documents. I can only provide code based on the information and descriptions provided to metadata
You: Explain yourself: tell me everything you know about the figma document
AI: As an AI, I don't have the ability to directly view or interpret Figma documents. However, based on the metadata and nodes provided in the text, here's what I can infer:
1. The document is named "Cookbook" and was last modified on April 16, 2024.
2. The document contains a frame named "Shopping list" with an id of 10:16. This frame has a child vector with an id of 10:30.
3. There are text elements with ids 10:58 and 10:42, named "Name" and "Apples (5)" respectively. Both text elements use the Roboto font, with the "Name" text being semi-bold and 24px, and the "Apples (5)" text being light and 20px.
4. The document has a white background color and uses a normal blend mode.
5. The document is owned by the user and can be viewed through the provided link.

Please note that this is a very basic interpretation based on the provided text. For a more accurate conversion of the Figma design to code, a direct link to the Figma file or a more detailed description of the design would be needed.
You: This is the link: https://www.figma.com/file/vl7p4WtOLFHQskJhLPZ6XX/Cookbook?type=design&node-id=1-2&mode=design can you tell me something more?
AI: I'm sorry for the confusion, but as an AI developed by OpenAI, I don't have the ability to directly access or view external links, including Figma files. I can only provide assistance based on the information provided to me in the text
You: Then why did you say "For a more accurate conversion of the Figma design to code, a direct link to the Figma file or a more detailed description of the design would be needed"?
AI: I apologize for the confusion. That statement was incorrect. As an AI developed by OpenAI, I don't have the ability to directly access or view external links, including Figma files. I can only provide assistance based on the information provided to me in the text. For a more accurate conversion of the Figma design to code, a more detailed description of the design would be needed

```

Figura 6.11. Screenshot in cui l'IA descrive ciò che realmente può desumere dal file Figma. Anche qui si può osservare una risposta inventata e subito dopo ritrattata dalla stessa intelligenza artificiale che l'ha fornita.

Un altro ragionamento utile è stato quello in merito al prompt di sistema: dal momento che istruzioni come "Code the whole page" sembravano insufficienti, sembrava inutile sprecare ogni volta diversi messaggi per ripetere le stesse direttive. L'evoluzione successiva ha riguardato dunque proprio il prompt di sistema, che dalla versione minimale del paragrafo 6.1.1 è stato arricchito con molte descrizioni, trasformandosi nella versione finale riportata nella figura 6.12.

Queste due modifiche apparentemente semplici hanno reso possibile generare un'applicazione Angular completa fin dalle prime iterazioni: l'intelligenza artificiale ha iniziato a generare risposte complete, fornendo sia i comandi per creare l'app e generare i componenti da terminale, che il codice completo di tutti i file. Ciò ha reso possibile concentrarsi su problemi di livello più alto. Pur generando pagine soddisfacenti, il programma non era però immune dagli errori:

- Spesso venivano generati solo l'HTML e il CSS. Anche se nel prompt era specificato di generare anche il codice TypeScript di Angular, quest'ultimo veniva aggiunto solo dopo averlo chiesto esplicitamente con un messaggio utente. Dopo aver iniziato a nominare TypeScript più volte nel prompt questo problema è stato risolto, ma rimane comunque un fatto curioso, dal momento che HTML e CSS sono sempre stati creati anche menzionandoli una volta sola.
- Una costante che dai primi tentativi con solo HTML e CSS è perdurata fino alla fine è

```

system_prompt_template = """You are expert coder Jon Carmack. Use the provided design
context to create Angular typescript code along with HTML and CSS as possible based
on the user request.
Write css and html code to ensure that:
- The navbar must be light grey, almost white with dark grey text. It must have "Recipe
  book" (with a big, bold font), "recipes" and "shopping list" to the left and
  "logout" and a div containing "manage" and a caret to the right.
- Everything in the navbar must be in the same line.
- Under the navbar there must be a form with two text inputs labeled as name and
  amount: they must occupy 6 and 3 columns each (use col-xs-x tags); they must
  be inside a same row in the form.
- Inside the form add a second row with two buttons called add and clear (green
  and blue), each occupying 3 columns (use the same classes as before). They
  must be aligned to the left.
- Under the form there is an item list. The items must have dark grey borders
  and no margin between them and they in must be the form "name (amount)". The
  first and last item must have rounded upper and lower corners, respectively.
Write typescript code to ensure that:
- When the app starts, the list must be already populated.
- When add is clicked add an item to the list; clear the list when clear is clicked.
- When an item is selected, the clear button must become red and get the name
  "delete", making the user able to delete the selected item. When the user
  clicks again on the item or somewhere else in the page, the button must return "clear".
Don't use AngularJS: use Angular 14.
Be sure to generate the typescript code to implement the requested functionalities
for Angular components.
Figma file nodes and metadata: {context}"""

```

Figura 6.12. Versione finale del prompt, che comprende tutte le direttive che è stato necessario aggiungere nel corso dei vari esperimenti eseguiti

stato il problema del margine dei componenti dal lato sinistro della pagina: pur facendo notare il problema, non si riesce a fare in modo che form, bottoni e lista degli ingredienti abbiano lo stesso margine rispetto al lato sinistro della pagina, creando un risultato poco piacevole alla vista.

- Anche se nel prompt è specificato, bisogna ricordare ogni volta al chatbot di implementare il deselezionamento dell'elemento corrente della lista quando si clicca in un punto vuoto: una volta ricordatoglielo, genera un `HostListener` (direttiva che permette di reagire con una funzione di callback a specifici eventi che riguardino un componente), ma prima di allora se ne dimentica tutte le volte. Altre volte fornisce una soluzione per niente elegante e soprattutto non funzionante che consiste nell'incapsulare tutta la pagina in un componente generico di tipo `<div>` che una volta cliccato deseleziona l'elemento corrente: questa strategia fa sì che sia impossibile selezionare un ingrediente della lista, perché ogni clic attiva il deselezionamento.
- Altre volte al problema precedente viene proposta una soluzione che funziona, ma che inizializza male gli stati dell'applicazione, rendendo invisibili il tasto Clear e la lista fino a quando non si interagisce in qualche modo con il form.

Il frequente verificarsi di questi errori ha permesso di fare luce sulla capacità di gpt-4.0 di auto-correggersi: se si copiava e incollava nella chat un messaggio di errore preso dal terminale

```
You: Now it works, but look better at the system prompt: aren't you forgetting some functionality?
AI: You're right; I overlooked the functionality where the "Clear" button changes to "Delete" and turns red when an item is selected, allowing the user to delete the selected item. Let's implement this functionality to complete the requirements.

### Step 1: Update the Item Form Component
We need to modify the 'ItemFormComponent' to handle the dynamic change of the "Clear" button to "Delete" and its color change based on whether an item is selected.

### Step 2: Update the AppComponent
The 'AppComponent' needs to manage the state of whether an item is selected and communicate this state to the 'ItemFormComponent'.

Here's the updated code:

### ItemFormComponent (item-form.component.html)
Update the buttons to dynamically change based on the selected item:
<form class="item-form">
  <div class="form-row">
    <div class="col-xs-6">
```

Figura 6.13. Screenshot che mostra come l'intelligenza artificiale abbia individuato la funzionalità mancante dopo averle fatto genericamente notare che una parte del prompt di sistema fosse stata trascurata.

di Angular, il problema veniva inquadrato correttamente e una soluzione corretta veniva fornita al primo tentativo. Più difficile era la correzione dei difetti che non generavano un errore da terminale, ma che erano solo sgradevoli per un utente umano, come il problema del margine sinistro dei componenti (mai risolto veramente) o quello del click-away nella lista, a cui spesso era l'utente a dover ricordare di usare l'HostListener come aveva fatto la prima volta al posto del <div> che scombinava tutto. Purtroppo, problemi come quest'ultimo o quello della lista invisibile richiedevano un certo livello di conoscenza del linguaggio impiegato, tale da poter suggerire all'agente virtuale la strategia da adottare ed erano quindi al di fuori della portata di un utente comune.

In un'occasione però, l'intelligenza artificiale ha rivelato una buona capacità di autocorrezione, ricordando accuratamente quanto scritto nel prompt e individuando l'aspetto che aveva trascurato senza che gli venisse esplicitamente detto quale fosse (vedi figura 6.13).

Capitolo 7

Osservazioni sui risultati precedenti e conclusioni

In questo ultimo capitolo commenteremo ciò che di più rilevante è emerso dalle osservazioni appena effettuate. Segue un pensiero riassuntivo sull'utilizzo del programma.

7.1 Osservazioni

Nel capitolo precedente è emerso come anche nelle versioni finali del chatbot basate su versioni di GPT più avanzate continuassero a verificarsi errori grossolani come margini dei componenti non particolarmente appaganti per l'occhio, soluzioni poco creative ed errori di programmazione (vedi paragrafo 6.5). Il comportamento più sgradito rimane però quello illustrato nei paragrafi 6.1.2 e 6.5: come in molte altre applicazioni basate su questa tecnologia, l'intelligenza artificiale inventa contenuti di sana pianta ogniqualvolta non sappia come andare avanti con la generazione di un contenuto, invece di ammetterlo con l'utente. Il problema è talmente ricorrente nel campo dell'intelligenza artificiale da aver ottenuto un nome, allucinazione, ed è ben riassunto dalle parole del professor Ethan Mollick dell'università di Wharton, che ha definito ChatGPT come *uno stagista onnisciente e ansioso di compiacerti che a volte ti mente* [49].

Le allucinazioni sono così diffuse da aver costretto Meta al ritiro di una piattaforma perché generava risposte errate facendole comunque suonare autorevoli [50] e nemmeno OpenAI, sulle cui API si basa questo progetto sembra esserne immune. In un esperimento condotto dalla ricercatrice svizzera Teresa Kubacka, ChatGPT ha infatti prodotto un testo su un fenomeno fisico inesistente il cui nome era stato inventato sul momento dalla scienziata, a cui durante il dottorato era capitato di dover studiare fisica. Il testo era così ben progettato da riportare anche citazioni di testi inesistenti attribuite però a personalità di spicco in quel campo di ricerca [49].

Nel paragrafo 6.5 si può osservare un altro fenomeno interessante: per quanto passare uno screenshot o aggiornare il modello dell'agente virtuale possano essere stati fattori che hanno contribuito, ciò che più ha permesso al programma di creare pagine soddisfacenti è stata sicuramente la sostituzione del prompt di base del paragrafo 6.1.1 con quello dettagliato mostrato al paragrafo 6.5. Appare allora evidente che, per quanto buone siano le sue prestazioni, la creatività dell'intelligenza artificiale non sia ancora sufficiente da sola: gran parte del risultato deriva dall'accuratezza del prompt.

Pur avendo chiaro *cosa* si voglia chiedere ad un agente virtuale, bisogna dunque prestare molta

attenzione a *come* lo si chiede, ed è perciò importante eseguire più esperimenti con prompt diversi; anche una conoscenza di base su ciò che si vuole generare è richiesta, in modo da sapere su cosa focalizzarsi nella stesura del prompt (in questo caso i linguaggi HTML, CSS e Angular). Questi due fattori e la diffusione sempre maggiore dell'intelligenza artificiale stanno rendendo quella del prompt engineer una figura molto in voga: questi professionisti specializzati nella stesura di prompt volti ad ottenere risultati specifici sembrano essere molti richiesti, anche se ci sono dubbi sul loro futuro. Ciò che crea perplessità su questo mestiere è la sua utilità in una prospettiva a lungo termine, visto l'incremento sempre più promettente delle prestazioni dei sistemi di intelligenza artificiale [51]: una ricerca svolta da Rick Battle e Teja Gollapudi presso l'azienda californiana VMware ha infatti evidenziato come il miglior creatore di prompt sia spesso (e quasi sempre il più rapido) la stessa intelligenza artificiale [52].

Al di là dei dubbi sul futuro dei prompt engineer, l'importanza di un prompt ben strutturato è al momento innegabile, come dimostrano i risultati di questa tesi e il fatto che la stessa OpenAI abbia pubblicato sul proprio sito una serie di linee guida su come ottenere il massimo dalle sue API ponendo le domande ai modelli di intelligenza artificiale in modo ottimale [53].

7.2 Conclusione

I risultati e le osservazioni raccolti finora mettono in luce il pubblico a cui questo lavoro può essere utile.

Al momento attuale, le capacità dell'intelligenza artificiale fanno sì che lo strumento descritto in queste pagine possa trovare impiego solo fra sviluppatori che abbiano competenze di programmazione (anche minime) e siano in grado di leggere con occhio critico le risposte generate, invece di prenderle ciecamente per buone.

Un possibile sviluppo futuro sarebbe invece quello di allargare il bacino di utenti alle aziende: un cliente che voglia far capire ai programmatori che tipo di prodotto si aspetti potrebbe trovare sicuramente comodo avere un prototipo con le funzionalità di base già implementate. Per ottenere un risultato del genere bisogna però attendere di potersi basare su delle API con un livello di intelligenza non ancora raggiunto dagli agenti virtuali odierni, che al momento non permettono di far generare codice di base ad utenti senza alcuna competenza in ambito informatico.

In sintesi, anche se suscettibile ad errori questo strumento rimane un valido aiuto in grado di far risparmiare ai programmatori una buona quantità di tempo e i suoi eventuali sviluppi non potranno che dipendere da un futuro in cui l'intelligenza artificiale promette prestazioni sempre migliori in fatto di creatività, comprensione dei desideri dell'utente e velocità di generazione delle risposte.

Bibliografia

- [1] Francesco Amigoni, Viola Schiaffonati e Marco Somalvico, "Intelligenza Artificiale", *Enciclopedia della Scienza e della Tecnica*, Treccani, 2008. Disponibile online: [https://www.treccani.it/enciclopedia/intelligenza-artificiale_\(Enciclopedia-della-Scienza-e-della-Tecnica\)/](https://www.treccani.it/enciclopedia/intelligenza-artificiale_(Enciclopedia-della-Scienza-e-della-Tecnica)/)
- [2] High-Level Expert Group on Artificial Intelligence, "A definition of AI: Main capabilities and scientific disciplines", 2018. Disponibile online: <https://digital-strategy.ec.europa.eu/it/node/2226>
- [3] Neil Bhutta, Aurel Hizmo e Daniel Ringo, "How Much Does Racial Bias Affect Mortgage Lending? Evidence from Human and Algorithmic Credit Decisions", *Finance and Economics Discussion Series 2022-067*, Washington: Board of Governors of the Federal Reserve System, 2022. Disponibile online: <https://doi.org/10.17016/FEDS.2022.067>
- [4] Yuchen Jiang, Xiang Li, Hao Luo, Shen Yin e Okyay Kaynak, "Quo vadis artificial intelligence?", *Discover Artificial Intelligence* 2, 4, 2022. Disponibile online: <https://doi.org/10.1007/s44163-022-00022-8>
- [5] Henry Shevlin, Karina Vold, Matthew Crosby e Marta Halina, "The limits of machine intelligence", *EMBO reports*, Vol.20, e49177, 2019. Disponibile online: <https://www.embopress.org/doi/abs/10.15252/embr.201949177>
- [6] Shane Legg e Marcus Hutter, "Universal intelligence: A definition of machine intelligence.", *Minds and machines* 17, 2007: 391-444. Disponibile online: https://users.umiacs.umd.edu/~hal/courses/2016F_RL/Legg07.pdf
- [7] Gopinath Rebala, Ajay Ravu e Sanjay Churiwala, "An Introduction to Machine Learning", Springer International Publishing, 2019. Disponibile online: <https://doi.org/10.1007/978-3-030-15729-6>
- [8] Hannes Schulz e Sven Behnke, "Deep learning: Layer-wise Learning of Feature Hierarchies", *KI-Künstliche Intelligenz*, vol. 26, pp. 357-363, 2012. Disponibile online: https://www.researchgate.net/publication/230690795_Deep_Learning_Layer-wise_Learning_of_Feature_Hierarchies
- [9] Yann Lecun, Yoshua Bengio e Geoffrey Hinton, "Deep learning". *Nature*, 2015, 521 (7553), pp.436-444. Disponibile online: <https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf>
- [10] Shai Shalev-Shwartz e Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms". Cambridge University Press, 2014, p 268
- [11] Zhihua Zhang, "Multivariate Time Series Analysis in Climate and Environmental Research". Springer, 2018. Disponibile online: <https://dx.doi.org/10.1007/978-3-319-67340-0>
- [12] Simon Haykin, "Neural Networks and Learning Machines". Terza edizione, Pearson, 2009.
- [13] David Foster, "Generative Deep Learning - Teaching Machines to Paint, Write, Compose and Play". O'Reilly Media, 2019.

-
- [14] Wayne Xin Zhao e altri, "A Survey of large language models". arXiv preprint arXiv:2303.18223, 2023. Disponibile online: <https://doi.org/10.48550/arXiv.2303.18223>
- [15] Enkelejda Kasneci e altri, "ChatGPT for good? On opportunities and challenges of large language models for education". Learning and Individual Differences, 103, 2023. Disponibile online: <https://doi.org/10.1016/j.lindif.2023.102274>
- [16] Jean Kaddour e altri, "Challenges and applications of large language models". arXiv preprint arXiv:2307.10169, 2023. Disponibile online: <https://doi.org/10.48550/arXiv.2307.10169>
- [17] Omer Shahab, Bara El Kurdi, Aasma Shaukat, Girish Nadkarni e Ali Soroush, "Large language models: a primer and gastroenterology applications". Therapeutic Advances in Gastroenterology, 17, 2024. Disponibile online: [doi:10.1177/17562848241227031](https://doi.org/10.1177/17562848241227031)
- [18] "Israeli AI code developer Codota buys Canada's TabNine". Disponibile online: <https://en.globes.co.il/en/article-israeli-startup-codota-buys-canadas-tabnine-1001311136>
- [19] "Tabnine Reaches 1M+ Monthly Users, Expands on Vision to Support Entire Software Development Lifecycle". Disponibile online: <https://www.globenewswire.com/news-release/2023/02/22/2613342/0/en/Tabnine-Reaches-1M-Monthly-Users-Expands-on-Vision-to-Support-Entire-Software-Development-Lifecycle.html>
- [20] Anthony Alford, "OpenAI Announces 12 Billion Parameter Code-Generation AI Codex". Dal sito InfoQ, 31/08/2021. Disponibile online: <https://www.infoq.com/news/2021/08/openai-codex/>
- [21] Nat Friedman, "Introducing GitHub Copilot: your AI pair programmer". Dal blog di GitHub, 29/06/2021. Disponibile online: <https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/>
- [22] "We've filed a lawsuit challenging GitHub Copilot, an AI product that relies on unprecedented open-source software piracy. Because AI needs to be fair & ethical for everyone." Dal sito GitHub Copilot litigation, url: <https://githubcopilotlitigation.com/>
- [23] David Ramel, "Copilot by the Numbers: Microsoft's Big AI Bet Paying Off". Visual Studio Magazine, 02/05/2024. Disponibile online: <https://visualstudiomagazine.com/Articles/2024/02/05/copilot-numbers.aspx>
- [24] Dave Gavigan, "The History of Angular". Dal sito Medium, 03/04/2018. Disponibile online: <https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7>
- [25] "Angular components overview". Dalla documentazione di Angular: <https://v17.angular.io/guide/component-overview>
- [26] "Ahead-of-time (AOT) compilation". Dalla documentazione di Angular: <https://v17.angular.io/guide/aot-compiler>
- [27] "Property binding". Dalla documentazione di Angular: <https://v17.angular.io/guide/property-binding>
- [28] "Event binding". Dalla documentazione di Angular: <https://v17.angular.io/guide/event-binding>
- [29] "Two-way binding". Dalla documentazione di Angular: <https://v17.angular.io/guide/two-way-binding>
- [30] "Dependency injection in Angular". Dalla documentazione di Angular: <https://v17.angular.io/guide/dependency-injection-overview>
- [31] "Angular Routing". Dalla documentazione di Angular: <https://v17.angular.io/guide/routing-overview>
- [32] "Built-in directives". Dalla documentazione di Angular: <https://v17.angular.io/guide/built-in-directives>

-
- [33] "Directive composition API". Dalla documentazione di Angular: <https://v17.angular.io/guide/directive-composition-api>
- [34] "CLI Overview and Command Reference". Dalla documentazione di Angular: <https://v17.angular.io/cli>
- [35] "Who Uses Angular in 2021? 10 Global Websites Built With Angular JS". Dal sito SoftProdigy. Disponibile online: <https://softprodigy.com/who-uses-angular-in-2021-10-global-websites-built-with-angular-js/>
- [36] Michelle Cheng, "Why Teams at Uber and Slack Use This Powerful Design Tool". Inc., 18/04/2019. Disponibile online: <https://www.inc.com/michelle-cheng/figma-design-cloud-collaboration-google-docs-30-under-30-2019.html>
- [37] Josh Constine, "23-Year-Old's Design Collaboration Tool Figma Launches With \$14M To Fight Adobe". TechCrunch, 3/12/2015. Disponibile online: https://techcrunch.com/2015/12/03/figma-vs-goliath/?guccounter=1&guce_referrer=aHR0cHM6Ly91bi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAAGq6Hq7150kLVKYznEdCQZns945I9SH8o0U8DwMugZNZB18xBt3_yYktDH5FAJvh0aqE01d_yJAnlu1UducNDunDSBS9cAOXua9d8wrh6_pKZf2mwbMHLAQDhKcpiY79IYLcVtFMJd_vEhoUafgjBSdmyG1Q9sriQPQYjsyXt4C9
- [38] Robbie Gonzalez, "Figma Wants Designers to Collaborate Google-Docs Style". Wired, 25/07/2017. Disponibile online: <https://www.wired.com/story/figma-updates/>
- [39] "Guide to components in Figma", dalla documentazione di Figma. Disponibile online: <https://help.figma.com/hc/en-us/articles/360038662654-Guide-to-components-in-Figma>
- [40] "Create and use variants", dalla documentazione di Figma. Disponibile online: <https://help.figma.com/hc/en-us/articles/360056440594-Create-and-use-variants>
- [41] "Styles in Figma", dalla documentazione di Figma. Disponibile online: <https://help.figma.com/hc/en-us/articles/360039238753-Styles-in-Figma>
- [42] "Q&A with RAG", dalla documentazione di LangChain. Disponibile online: https://python.langchain.com/v0.1/docs/use_cases/question_answering/
- [43] "Retrievers", dalla documentazione di LangChain. Disponibile online: https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/
- [44] "LangChain Expression Language (LCEL)", dalla documentazione di LangChain. Disponibile online: <https://python.langchain.com/v0.2/docs/concepts/#langchain-expression-language-lcel>
- [45] "Output parsers", dalla documentazione di LangChain. Disponibile online: <https://python.langchain.com/v0.2/docs/concepts/#output-parsers>
- [46] "What are tokens and how to count them?". Dal sito di OpenAI: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>
- [47] "Do Big Companies Use AngularJS?". Dal sito Medium, 18/09/2023: <https://medium.com/@techedtalkhub/do-big-companies-use-angularjs-20431003aa71#:~:text=Amazon%20uses%20AngularJS%20for%20its,complex%20and%20demanding%20web%20applications.>
- [48] "Figma". Pagina del sito di langchain che illustra l'integrazione Figma: https://python.langchain.com/v0.2/docs/integrations/document_loaders/figma/
- [49] Emma Bowman, "A new AI chatbot might do your homework for you. But it's still not an A+ student". Dal sito NPR, 19/12/2022: <https://www.npr.org/2022/12/19/1143912956/chatgpt-ai-chatbot-homework-academia>
- [50] "New Meta AI demo writes racist and inaccurate scientific literature, gets pulled". Dal sito Ars Technica, 19/11/2022: <https://arstechnica.com/information-technology/2022/11/after-controversy-meta-pulls-demo-of-ai-model-that-writes-scientific-papers/>

- [51] Oguz A. Acar, "AI Prompt Engineering Isn't the Future". Harvard Business Review, 06/06/2023. Disponibile online: <https://hbr.org/2023/06/ai-prompt-engineering-isnt-the-future>
- [52] Dina Genkina, "AI Prompt Engineering Is Dead". IEEE Spectrum, 06/03/2024. Disponibile online: <https://spectrum.ieee.org/prompt-engineering-is-dead>
- [53] "Prompt engineering". Dal sito di OpenAI: <https://platform.openai.com/docs/guides/prompt-engineering>