

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

**CHESHIRE CAT AI:
FRAMEWORK PER LA REALIZZAZIONE
DI CHATBOT SPECIALIZZATI**

Relatore:
Chiar.mo Prof.
Giancarlo Succi

Presentata da:
Francesco Testa

II Sessione
Anno Accademico 2023/2024

心臓を捧げよ!

(Shinzō wo sasageyo! - Dedica il tuo cuore!)

Abstract

L'uso dei chatbot basati su Large Language Models (LLM) sta rivoluzionando il mondo del lavoro. La creazione di assistenti virtuali specializzati in un singolo compito risulta fondamentale per ricoprire la "coda lunga", ovvero casi d'uso ampiamente specifici. Questo studio mira ad analizzare le metodologie allo stato dell'arte per la verticalizzazione degli LLM, confrontandone aspetti negativi e positivi. Le soluzioni attualmente disponibili richiedono specialisti, spesso rendendo il processo costoso. Tuttavia, progetti come il *Cheshire Cat AI* permettono di modificare il comportamento standard di un modello implementando plugin specifici. È stata analizzata l'architettura del framework ed è stato codificato un plugin, per dimostrarne la semplicità d'uso. Gli sviluppatori del *Cheshire Cat AI* prevedono l'implementazione della multimodalità e la riduzione delle allucinazioni attraverso l'uso di Knowledge Graphs. Inoltre, essendo il progetto open source, è possibile contribuire al *core*, oppure caricare i propri plugin nel registro pubblico. Il plugin *ccat_groups* realizzato per lo studio è disponibile e può essere installato facilmente.

Indice

1	Introduzione	1
2	Prerequisiti	3
2.1	Rete Neurale	3
2.2	Large Language Model	7
2.2.1	Attention mechanism	7
2.2.2	Transformer	7
3	Verticalizzazione LLM	11
3.1	Fine-tuning	11
3.2	RAG	12
3.3	Uso delle Ontologie	19
3.3.1	Question Answering (QA) su dati privati	21
3.4	Confronto fra i metodi	24
4	Cheshire Cat AI	26
4.1	Architettura	27
4.2	Messaggio	32
4.3	Plugin	34
5	Implementazione di un plugin	38
5.1	Demo	43
6	Conclusioni	45
A	Test group	51

Elenco delle figure

2.1	Neurone artificiale	3
2.2	Rete neurale artificiale	4
2.3	Architettura Encoder-Decoder	6
2.4	Architettura Transformer	8
3.1	Architettura di un'applicazione basata su in-context learning	14
3.2	Database Vettoriale	15
3.3	Architettura che combina diversi approcci RAG	19
3.4	Architettura di un sistema per QA	23
4.1	Architettura del core del Cheshire Cat AI	29
4.2	Flowchart dell'endpoint <i>/message</i>	32

Code Listings

4.1	Implementazione singleton in Python	26
4.2	Instruction Prompt	30
4.3	Prefisso del Main Prompt	31
4.4	Esempio Form: classe per il contenuto JSON da inviare all'API . . .	35
4.5	Esempio Form: classe base che gestisce il riempimento del JSON . . .	35
5.1	Implementazione del modello per il form di aggiunta/rimozione di utenti dal gruppo	39
5.2	Implementazione del form per l'aggiunta di un utente al gruppo. . . .	40
5.3	Implementazione Rabbit Hole Hooks	41
5.4	Comandi disponibili per la gestione dei gruppi	42
5.5	Implementazione hook <i>agent_fast_reply</i>	42

Capitolo 1

Introduzione

Negli ultimi anni, il ruolo dell'intelligenza artificiale è diventato fondamentale in molti ambiti. Lo sviluppo è stato trainato dai Large Language Model, i quali si basano, per la grande maggioranza, sull'architettura a Transformer. Questa architettura ha giocato un ruolo cruciale nella comprensione delle dipendenze all'interno del testo, conducendo passi avanti nello studio del Natural Language Processing (NLP). Gli LLM sono alla base degli assistenti virtuali, adottati in tantissimi ambiti - come la sanità, il campo legale o quello economico - per migliorare l'accessibilità alle informazioni o avere l'automazione di pratiche ripetitive. Nonostante gli LLM offrano una conoscenza elevata in numerosi ambiti, la verticalizzazione è la soluzione che permette di raggiungere una conoscenza maggiore in un campo specifico.

L'obiettivo della tesi è fornire le conoscenze sull'architettura a Transformer, per comprendere il funzionamento degli LLM, ed analizzare il problema della verticalizzazione. Segue, inoltre, la presentazione di un progetto open-source italiano, ovvero il framework Cheshire Cat AI¹, che offre la possibilità di realizzare chatbot verticali in modo semplice. Il framework aspira a diventare, nel mondo degli assistenti virtuali, ciò che Wordpress rappresenta per lo sviluppo di siti web².

La tesi è divisa nel seguente modo. Il capitolo 2 fornisce delle conoscenze sulla base dei chatbot, ovvero i Large Language Model e analizza il funzionamento dei transformer, l'architettura utilizzata per la costruzione di reti neurali che capiscano le dipendenze nel testo.

Il capitolo 3 esplora diverse soluzioni per verticalizzare un LLM, come la RAG o il fine-tuning. Poiché la creazione di modelli comporta costi molto elevati, sono poche le aziende che possono permetterselo. Le piccole imprese devono quindi par-

¹<https://github.com/cheshire-cat-ai/core>

²Wordpress è un progetto open-source che permette di realizzare, in modo semplice, siti web. È diventato molto popolare: il 43,5% dei siti nel mondo utilizza Wordpress[15]. Il link per il codice sorgente è il seguente:<https://github.com/wordpress>.

tire da modelli già esistenti e cercare di verticalizzarli nel loro specifico settore di competenza.

Il capitolo 4 analizza il Cheshire Cat AI: un framework che si ispira al modello Wordpress per la realizzazione di assistenti virtuali verticali in un settore. Attraverso la scrittura di plugin si possono personalizzare alcune componenti in modo da allontanarsi dal comportamento standard.

Il capitolo 5 mostra l'implementazione di un semplice plugin, ideato per aiutare gli studenti a preparare test orali.

Capitolo 2

Prerequisiti

2.1 Rete Neurale

Una rete neurale artificiale (ANN) è un paradigma di computazione di informazioni ispirato al mondo della biologia, in particolare al cervello umano.[11] L'obiettivo dei ricercatori è sempre stato quello di automatizzare task difficili da risolvere mediante algoritmi classici, come la pattern recognition, ma semplici da risolvere per un umano. Con questo intento arriva il tentativo di simulare i processi del cervello, ricreando una struttura che somigli ad un neurone biologico.

La singola unità di computazione è il neurone artificiale, chiamato anche nodo (figura 2.1). La computazione può essere divisa in due fasi: la prima riguarda la combinazione lineare calcolata a partire dagli input ($b + \sum_{i=1}^n x_i w_i$); nella seconda si prende l'output generato e lo si usa come input per una funzione di attivazione, il cui scopo è quello di attivare o meno il neurone.

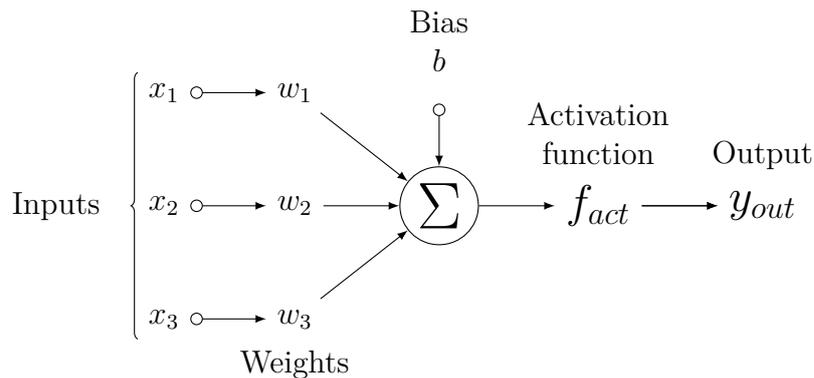


Figura 2.1: Neurone artificiale

Il bias è una costante che permette di shiftare la funzione di attivazione. Quest'ultima è una funzione non lineare: la non linearità è un requisito fondamentale per aumentare l'espressività del modello; senza di essa si potrebbe considerare l'intera rete come una grossa combinazione lineare.

Componendo assieme più neuroni, si viene a creare un grafo chiamato Artificial Neural Network (ANN). In particolare, la rete si chiamerà "deep" se ci sono uno o più layer nascosti, dove il layer nascosto è un qualsiasi strato esistente tra quello di input e quello di output (figura 2.2).

I parametri della rete che possono essere allenati sono i pesi e il bias. Nella fase di training, quindi, sono modificati questi valori, in un processo che continua fino ad un risultato desiderato.

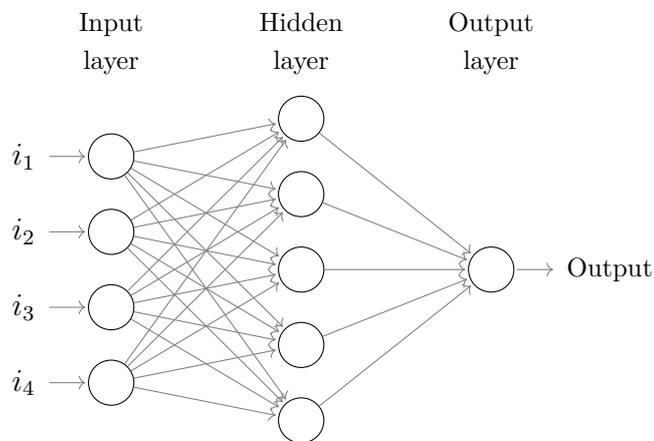


Figura 2.2: Rete neurale artificiale

L'allenamento può essere diviso in 3 categorie:

- **Supervisionato:** la rete è allenata a partire da dataset formati da coppia (input - risultato atteso). La modifica dei parametri avviene calcolando la distanza tra l'output e il risultato atteso; l'errore così generato è usato nell'algoritmo del backpropagation, che punta a diminuire l'errore cercando i minimi della funzione.
- **Non supervisionato:** la rete è allenata per scoprire attributi e classificare pattern riconosciuti negli input. Non sono definite delle categorie a priori.
- **Apprendimento per rinforzo:** si può considerare un allenamento intermedio tra i due citati precedentemente. Non si parte da un dataset specifico, ma si basa sul compimento di azioni in un determinato ambiente da parte del modello. Se l'azione è giudicata in modo positivo allora si darà una ricompensa, altrimenti una punizione. In base al giudizio ottenuto, si modificheranno i

parametri. L'obiettivo è spingere il modello a compiere l'azione giusta in quel determinato ambiente. Un esempio in cui usare il Reinforcement Learning è quello di insegnare, ad un modello, a completare un gioco. Quando il modello eseguirà la prima azione giusta, sarà ricompensato e spinto a eseguirla nuovamente.

Lo studio che utilizza le reti neurali profonde è definito Deep Learning, ed è un sottoinsieme del Machine Learning.[17]

Le tecniche esplorate nel Deep Learning e la capacità delle ANN di imparare a risolvere problemi complessi si sono intrecciate con l'ambito delle NLP.

Per NLP (acronimo di Natural Language Processing) o elaborazione del linguaggio naturale si intendono algoritmi di Intelligenza Artificiale in grado di analizzare, rappresentare e quindi comprendere il linguaggio naturale. Le finalità possono variare dalla comprensione del contenuto, alla traduzione, fino alla produzione di testo in modo autonomo a partire da dati o documenti forniti in input.[30]

Lo studio delle NLP parte dagli anni '50. La prima grande rivoluzione avvenne con l'adozione di sistemi statistici, negli anni '90: i modelli probabilistici si basano sulla probabilità che una parola occorra in un determinato punto.

Negli ultimi anni il Deep Learning si è affermato come metodo principale nello studio dei problemi di elaborazione del linguaggio naturale.

Architetture di rete utilizzate in questo ambito sono la CNN e la RNN[19].

Una CNN (Convolutional Neural Network) è un tipo di rete capace di estrarre autonomamente degli attributi dagli input attraverso strutture convoluzionali[22]. L'estrazione dei pattern è fortemente condizionata dal kernel convoluzionale: è un parametro della rete, quindi modificabile o addestrabile.

Nonostante, in un primo momento, siano state utilizzate soltanto per la computer vision e l'elaborazione di immagini, le reti neurali convoluzionali hanno poi trovato impiego anche nello studio degli NLP, in quanto capaci di catturare, nella struttura del testo, pattern locali e dipendenze globali nello stesso momento[19].

Una RNN (Recurrent Neural Network) è un tipo di rete realizzato per la lavorazione di dati sequenziali. Infatti, in molti task, come ad esempio la predizione di una parola in una frase, l'uso di una rete neurale classica porterebbe a scarse performance poiché gli input sono considerati indipendenti tra loro. Una RNN è, per definizione, ricorrente, quindi si eseguirà lo stesso task su ogni elemento della sequenza in input, e ogni output dipende da quelli precedenti.[23]

L'architettura RNN si è evoluta nella LSTM(Long Short-Term Memory) per ridurre il problema della scomparsa del gradiente. Questa architettura introduce un nuovo componente, ovvero l'utilizzo di celle di memoria. Ogni cella è composta da tre gate: gate di input, che decide quali informazioni aggiornare e come; gate di forget, che decide quali informazioni devono essere eliminate; gate di output, che decide quali informazioni devono essere inviate come output.[29] In questo modo è possibile

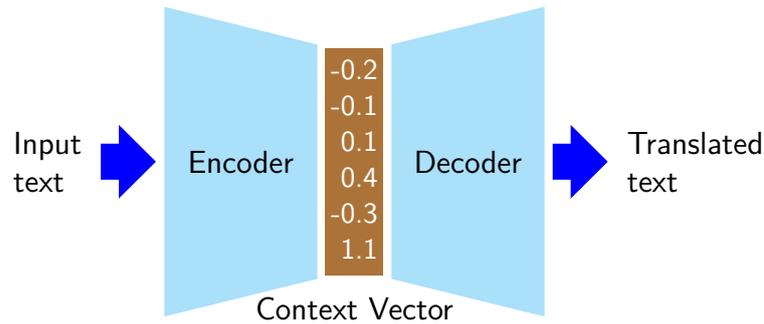


Figura 2.3: Architettura Encoder-Decoder

catturare dipendenze anche distanti nell'input, riuscendo a mitigare il problema della scomparsa del gradiente e offrire performance migliori rispetto alle RNN classiche. Un problema che persisteva riguardava la gestione degli input di lunghezza variabile e la generazione di output di lunghezza variabile. La struttura Encoder-Decoder rappresenta l'evoluzione che elimina questo problema: la separazione tra elaborazione dell'input (encoder) e generazione dell'output (decoder) permetteva una maggiore flessibilità in task come la traduzione o la sintetizzazione dell'input.[13] Tuttavia, permane il problema della grandezza fissata del Context Vector, che difficilmente riesce a catturare dipendenze distanti in dati sequenziali. Ciò che ha portato alla creazione e all'esplosione degli LLM è il meccanismo dell'attenzione e l'architettura a Transformer.

2.2 Large Language Model

2.2.1 Attention mechanism

Il meccanismo dell'attenzione è stato introdotto nel paper *Neural Machine Translation by Jointly Learning to Align and Translate*[6] per risolvere il problema della codifica dell'input in un vettore di dimensione fissa. Le poche informazioni sull'input a disposizione del Decoder erano una limitazione, soprattutto con frasi complesse, codificate anch'esse nella stessa dimensionalità di frasi semplici.

Il meccanismo dell'attenzione permette al decoder di "decidere" su quale parte della frase in input concentrarsi. Si può raggiungere questo risultato attraverso un encoder bidirezionale (BiRNN), che permette di cogliere le dipendenze di una parola non soltanto con il testo precedente, ma anche con quello successivo. Le dipendenze (hidden state o annotazioni) sono pesate in base alla posizione, attraverso un modello di allineamento: questa combinazione lineare genererà il context vector, utilizzato dal decoder nella produzione di testo.

Ad esempio, se si volesse tradurre la frase "Black cat ate the mouse" in francese, il modello si dovrebbe focalizzare prima sulla parola "cat" e poi "black". Il meccanismo dell'attenzione sposta la concentrazione sulla seconda parola, per poi tornare sulla prima. Quindi la traduzione sarà "Chat noir" e non "Noir chat". Proseguendo, il modello di allineamento permette al decoder di soffermarsi sulla parola "ate" per la generazione di due parole, ovvero "a mangé", in quanto la traduzione di una singola parola inglese richiede, in questo caso, due parole francesi.

Questa architettura migliora i problemi riscontrati precedentemente, ma ne introduce di nuovi, tra i quali spicca la complessità computazionale.

2.2.2 Transformer

Il nome all'architettura presentata nel paper *Attention Is All You Need*[34] indica la sua applicazione in tutti i campi che necessitano di trasformare una sequenza di input in una sequenza di output.

"A transformer is a type of artificial intelligence model that learns to understand and generate human-like text by analyzing patterns in large amounts of text data."[12] L'architettura Transformer mantiene il concetto di Encoder-Decoder. In particolare, un Transformer contiene, specularmente, N encoder/decoder: ogni encoder prenderà l'input originale e l'input dal layer precedente; ogni decoder prenderà, in input, l'output dell'ultimo encoder e l'output del decoder precedente.

Osservando la figura 2.4, si può dividere l'analisi dell'architettura in due parti: la fase di encoding, sulla sinistra, e la fase di decoding, sulla destra.

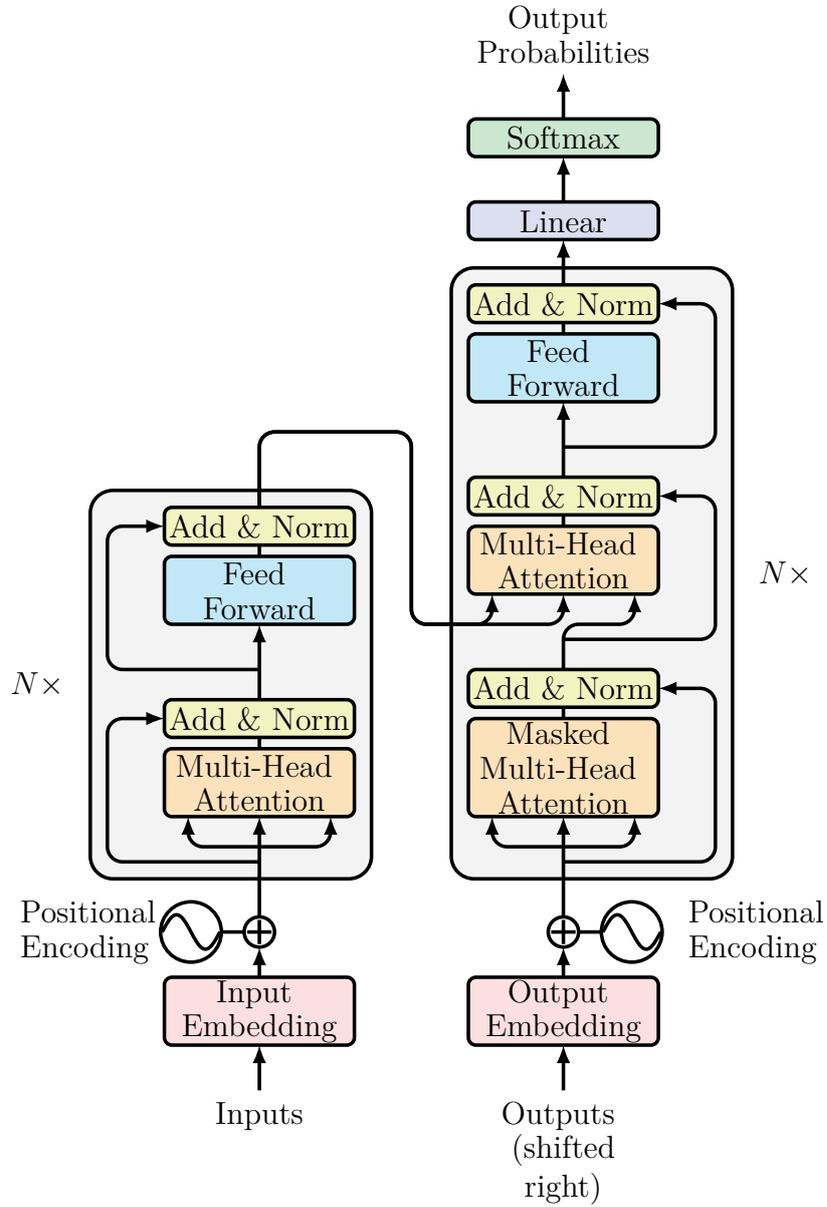


Figura 2.4: Architettura Transformer

encoding La prima fase riguarda la creazione di vettori che catturino la semantica di ogni token (parola), attraverso l'embedder. Ogni vettore è di 512 elementi, e questa sequenza di vettori sarà usata come input per ognuno degli N encoder.

Per avere informazioni riguardo la posizione di ogni token all'interno della frase, si aggiunge, al vettore generato precedentemente, un "positional encoding". In questo modo si sopperisce alla mancanza dell'utilizzo di reti ricorrenti (RNN) che riuscivano a catturare in modo semplice questo aspetto.

In seguito, si passa all'elaborazione da parte degli N encoder. L'obiettivo di ogni encoder è la rappresentazione astratta delle informazioni acquisite sull'input. Questa fase si suddivide in due sottofasi:

- **Multi-Headed Self-Attention Mechanism:** il self-attention permette al modello di legare ogni token in input con altre parole. Il transformer, a partire dall'embedding in input, genera tre vettori: Query, che rappresenta la parola su cui stiamo cercando informazioni; Key, che rappresenta un'altra parola all'interno della frase in input; Value, che, associato ad ogni chiave, rappresenta le informazioni della parola stessa. Se la Query e Key faranno match, verrà data maggiore enfasi al Value in output, con lo scopo di spostare l'attenzione del modello.

Calcolate Key e Query, si esegue una moltiplicazione tra matrici per ottenere un punteggio, che indica la relazione tra ogni parola. In seguito all'applicazione di normalizzazione e softmax, si moltiplica la matrice ottenuta con quella dei valori. In questo modo si avrà l'output del meccanismo di attenzione.

Per calcolare Query, Key e Value si usano tre matrici di pesi, le quali contribuiscono al totale dei parametri addestrabili della rete.

Nel sistema Multi-Head, si utilizzano più meccanismi di attenzione in modo parallelo e indipendente. I risultati verranno concatenati e trasformati attraverso un'ulteriore matrice dei pesi. Questa metodologia permette di analizzare lo stesso input da diversi punti di vista, in quanto le matrici dei pesi sono indipendenti, e dunque potranno essere composte da valori diversi e creare delle attenzioni diverse.[14]

- **Feed Forward Network** L'output precedente è usato come input in una rete neurale composta da due trasformazioni lineari¹, separate da funzione di attivazione ReLU. Il primo layer espande la dimensionalità di ogni punto dell'input; il secondo riporta ogni punto alla dimensione originale. Poiché si lavora indipendentemente su ogni punto dell'input, è possibile effettuare la computazione in parallelo, migliorando di molto le prestazioni. [28] Questo tipo di

¹Una trasformazione lineare è una funzione che conserva le operazioni di somma tra vettori e di prodotto di un vettore con uno scalare. Una trasformazione lineare preserva le combinazioni lineari, quindi si mantengono le relazioni, generate precedentemente, tra i token in input.

reti, introdotte nel paper *Neural Machine Translation by Jointly Learning to Align and Translate* hanno rivoluzionato il mondo delle NLP.

L'output dell'encoder sarà un insieme di vettori che rappresentano l'input, arricchiti da informazioni che dipendono dal contesto, come le relazioni tra le diverse parole.

decoding La seconda parte dell'architettura (la parte destra nella figura 2.4) si occupa della generazione del testo.

Come input usa l'output della generazione fino a quel momento: per generare il token i -esimo si sfruttano tutti i token generati fino all' $i-1$ -esimo. Come nella fase di codifica, l'input subisce una modifica dovuta al positional encoding. Successivamente è usato in un layer che calcola la multi-head attention, che differisce da quella dell'encoding per la presenza di una maschera. La maschera sposta l'attenzione del modello: banalmente, inserendo - *infinito* come valore nelle celle della matrice che non si vogliono considerare, il modello non si concentrerà lì.

Successivamente, si segue la struttura usata nella fase di encoding, già descritta. L'unica differenza è rappresentata dalla scelta dell'input: Query e Key sono presi dall'output dell'encoder, mentre la matrice Value è l'output dell'attention effettuata, come primo step, nel decoder. L'allineamento tra Encoder e Decoder, chiamato anche **Cross Attention**, permette al Decoder di enfatizzare le parti più rilevanti dell'input [12].

Infine, l'output generato nella fase di decoding è proiettato, attraverso una trasformazione lineare, in un vettore. L'applicazione lineare avviene con una matrice di "unembedding"², formata da tante righe quante sono le parole nel vocabolario, quindi il vettore avrà dimensione pari alla cardinalità di vocaboli utilizzati. Il layer Softmax trasforma tutti i valori in probabilità, ovvero valori tra 0 e 1, da cui si estrae quello maggiore. Questo token è aggiunto all'output già generato precedentemente e ricomincia il ciclo di generazione della prossima parola. Soltanto alla predizione di un token speciale il ciclo si interromperà.

²La matrice è formata da parametri addestrabili della rete.

Capitolo 3

Verticalizzazione LLM

Attualmente, lo sviluppo dell'intelligenza artificiale è fortemente centralizzato. Difatti, si possono evidenziare i numerosi passi avanti che stanno compiendo OpenAI (Microsoft) e Gemini(Google), con i rilasci di nuovi modelli che riescono a migliorare quelli precedenti. Ora, dunque, è difficile ottenere dei risultati ottimi usando modelli propri o modelli Open-Source (come può essere LLama), in quanto il capitale investito, necessario per la ricerca, l'addestramento e la collezione di dati, è inferiore, se messo in confronto con le aziende precedentemente citate[3].

L'obiettivo della ricerca e delle piccole aziende è dunque il riempimento della coda lunga, ovvero quella parte, prevalentemente di nicchia, che si specializza verticalmente in un settore. La coda lunga fa riferimento al gran numero di piccoli ricercatori e aziende che non hanno accesso a risorse computazionali per gestire numerosi dati e calcoli[20]. In questo caso si tratta di modelli di intelligenza artificiale che occupano uno specifico ruolo, in uno specifico campo. Per raggiungere questo scopo, i ricercatori hanno studiato alcune metodologie in cui, per i motivi elencati precedentemente, non si parte da 0, ma da Large Language Model già addestrati.

3.1 Fine-tuning

Per fine-tuning si intende il processo di ulteriore addestramento di un LLM su un piccolo dataset, affinché migliori le proprie capacità e performance su un dominio specifico. [27] È un metodo che è stato, e risulta ancora oggi, fondamentale, usato in moltissime AI. Per illustrare l'importanza di questa pratica si tratterà brevemente del suo utilizzo sul popolare ChatGPT. In seguito al primo addestramento della rete, era necessario rendere il chatbot politicamente corretto, affinché fosse usufruibile dalla massa. Per raggiungere questo obiettivo è stato usato il Reinforcement Learning from Human Feedback (RLHF). Questo approccio, che si basa sul Reinforcement Learning classico, si avvale della partecipazione umana per giudicare il

comportamento del modello, rendendolo affine alle preferenze e ai valori delle persone. L'esito di questa fase é la generazione di uno score che riassume i risultati ottenuti. Dall'insieme delle risposte si elabora un modello di ricompense che premierá o punirá le scelte successive della rete. Il processo si ripete iterativamente per migliorare le performance, fino all'ottenimento dei risultati sperati. [24]

SFT Nel caso in cui il processo sia supervisionato, si tratta di **Supervised fine-tuning** (SFT): preso il modello già allenato, si prepara un piccolo dataset etichettato e si esegue una seconda fase di addestramento. Le etichette sono fondamentali per calcolare l'errore e, di conseguenza, aggiustare i pesi della rete neurale[25]. È un approccio utilizzato per la classificazione del testo, riconoscimento di entità o ogni altro task supervisionato.

Instruction fine-tuning L'obiettivo è migliorare le abilità del LLM di eseguire determinate istruzioni. Un esempio potrebbe essere quello di tradurre una frase da una lingua ad un'altra. Si tratta di un metodo che sfrutta i principi del prompt engineering nel SFT.

Full fine-tuning È un processo basato sulla modifica di ogni parametro della rete di partenza. Tuttavia, è poco realizzabile in quanto richiede numerosa potenza di calcolo, necessaria per l'elaborazione e la memorizzazione di tutti i gradienti per la modifica dei parametri. Ad esempio, per effettuare un full fine-tuning di LLama 65B (Modello di Linguaggio Open-Source) sarebbero necessari almeno 780GB di memoria della scheda video [10].

PEFT Sviluppato per migliorare le prestazioni e diminuire i costi, il Parameter-Efficient Fine-Tuning(PEFT) si focalizza nell'addestrare soltanto un piccolo numero di parametri nel modello. Le tecniche allo stato dell'arte permettono di ottenere risultati molto simili a quelli del full fine-tuning, con costi e tempi decisamente minori. In queste tecniche rientrano sicuramente LoRA (Low-Rank Adaptation) e QLoRA (Quantized LoRA). Nel Paper di presentazione di LoRA [36] si effettua fine-tuning a partire da BERT. La quantità di parametri aggiustati risulta essere solamente lo 0.08%, con il congelamento di tutti gli altri.

3.2 RAG

Un altro approccio è basato sull'arricchire, nel miglior modo possibile, il prompt da dare come input al Large Language Model, affinché la risposta venga condizionata. Fornendo un contesto alla conversazione è possibile ricevere una risposta attinente alla domanda e che dunque esaudisca, con maggiore accuratezza, la richiesta. Per

effettuare un'operazione simile, limitandosi al solo uso del Modello di Linguaggio, si dovrebbero aggiungere manualmente tutte le informazioni necessarie al prompt, e infine inserire la domanda. Questo approccio può funzionare nel caso in cui le informazioni da passare siano poche, ma diventa un processo laborioso in caso contrario. Dunque, la soluzione è lavorare per la creazione di un sistema più complesso, alla cui base è eseguito un LLM.

Questa metodologia prende piede nel 2020, con il contributo del paper *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*[21], in cui è stata introdotta la Retrieval Augmented Generation(**RAG**). Attraverso questo modello, si cerca di risolvere, contemporaneamente, due problemi degli LLM: le allucinazioni e la difficoltà di aggiunta di conoscenza in seguito al primo addestramento.

Per farlo si sfrutta la metodologia dell'In-Context Learning[8], la cui definizione è *"a method of prompt engineering where the model is shown task demonstrations as part of the prompt in natural language."*¹

L'idea alla base è prendere un LLM che non abbia subito fine-tuning, e condizionare le risposte fornendo dei prompt maggiorati con dati derivanti dal contesto. L'immagine 3.1 mostra, a grandi linee, l'architettura, che può essere suddivisa in 3 parti:

- **embedding**: la prima fase riguarda la memorizzazione di dati e informazioni specifiche, che possono risultare utili ed essere recuperate e impiegate successivamente. Fondamentali in questa fase sono l'embedder e il database vettoriale.

L'embedder è un componente che prende in input una stringa (i.e. token), e restituisce in output un vettore. Questo vettore mantiene il significato semantico dell'input, quindi stringhe di significato simili saranno codificate in vettori vicini nello spazio. Tutti questi vettori vengono memorizzati in un database vettoriale. Il database vettoriale è un componente definito in questo modo: *... stores, manages and indexes high-dimensional vector data. Data points are stored as arrays of numbers called "vectors", which are clustered based on similarity. This design enables low-latency queries, making it ideal for AI applications.*² La definizione si focalizza su tre concetti: vettori; similarità, fondamentale per restituire valori semanticamente vicini³; efficienza, di primaria importanza in quanto spesso sono memorizzati milioni di vettori. La figura 3.2 mostra un semplice database vettoriale a due dimensioni. Si tratta di una visione semplicistica perché, in realtà, si usa un numero di dimensioni che non sono concepibili dalla mente umana.

¹<https://www.hopsworks.ai/dictionary/in-context-learning-icl>

²Definizione presa da: <https://www.ibm.com/topics/vector-database>

³È possibile calcolare la similarità tra vettori in diversi modi. Uno di questi, ad esempio, è la similarità del coseno.

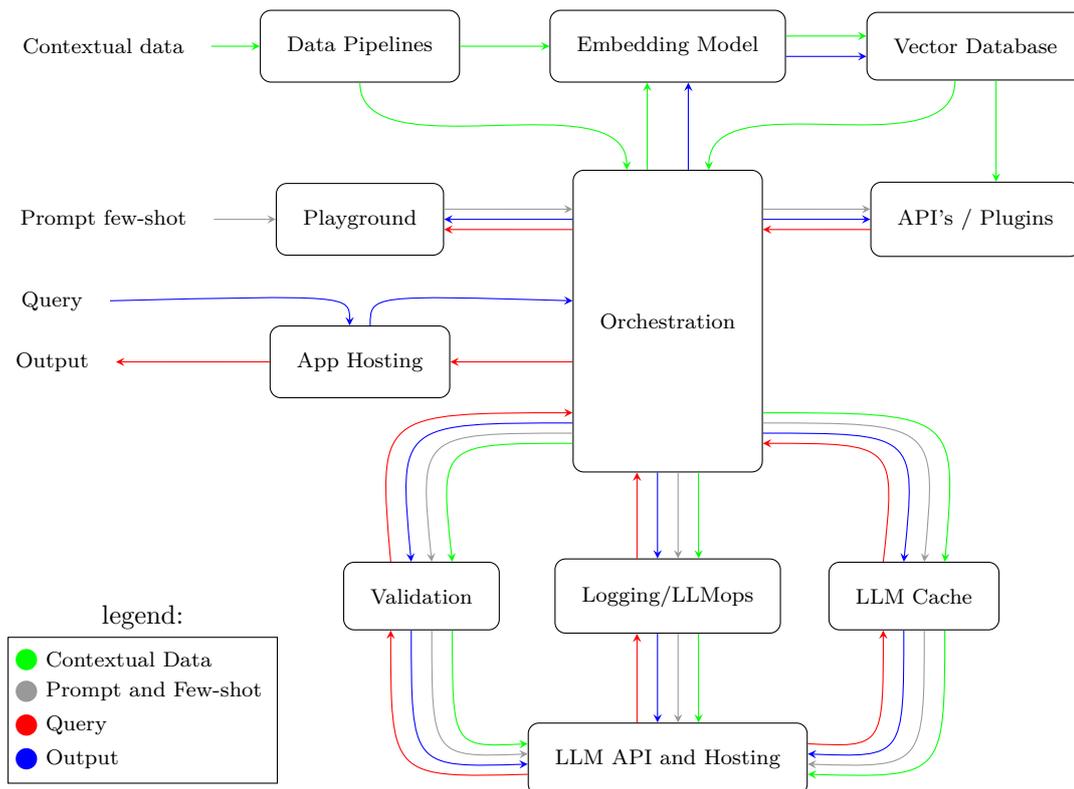


Figura 3.1: Architettura di un'applicazione basata su in-context learning. Immagine copiata e rielaborata da: Matt Bornstein e Rajko Radovanovic. *Emerging Architectures for LLM Applications*. Accessed: 09/06/2024. 2024. URL: <https://a16z.com/emerging-architectures-for-llm-applications/>

- retrieval:** quando un utente effettua una domanda, l'applicazione costruisce una serie di prompt contenenti esempi few-shot, e ogni altra informazione proveniente da API esterne o dal database vettoriale. Per ottenere dal database informazioni che siano semanticamente coerenti alla domanda, quest'ultima è usata come input nell'embedder; il vettore risultante sarà poi utilizzato per recuperare le altre informazioni attraverso il calcolo della distanza con gli altri vettori (ad esempio usando la similarità del coseno). Le strategie per la costruzione di prompt sono numerose, e varia l'utilizzo in base allo scopo da raggiungere. Fondamentale è dunque il componente centrale, ovvero il framework di orchestrazione (come può essere LangChain), il quale fornisce un'astrazione che permette la creazione di un prompt a partire da diversi input, come sono gli output delle API esterne o i dati derivati dal database vettoriale, mantenendo sempre traccia delle ultime chiamate al LLM.

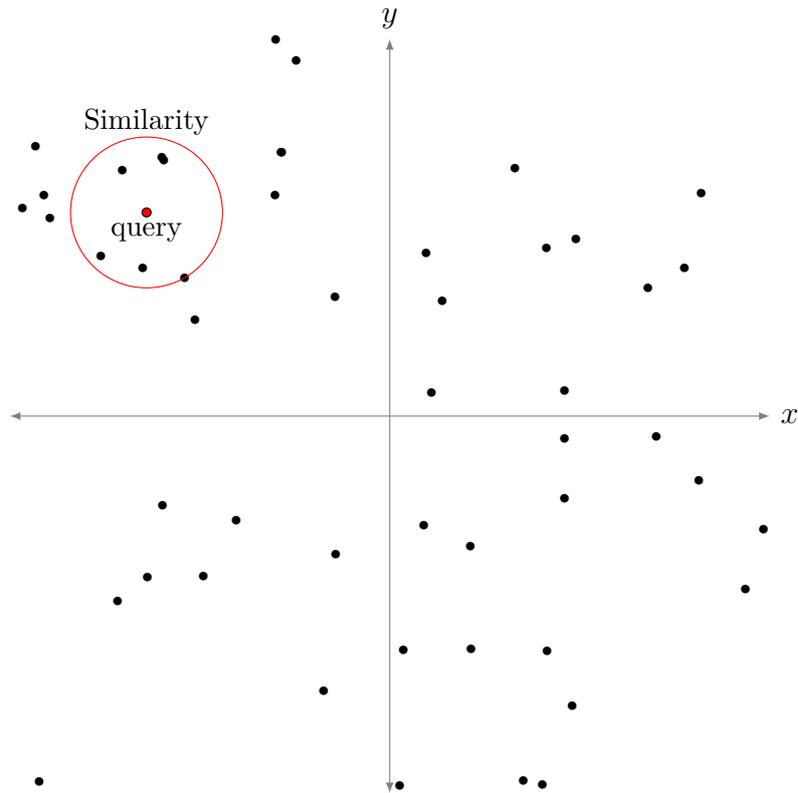


Figura 3.2: Database vettoriale a due dimensioni. Il punto etichettato indica il vettore, restituito in seguito all'embedding, relativo alla domanda dell'utente. Ipotezzando un qualsiasi calcolo per la similarità, i punti all'interno del cerchio sono quelli attinenti alla domanda, e dunque possono essere utilizzati.

- **inference:** la fase di inference è la generazione di un output a partire da dati diversi da quelli di addestramento. La realizzazione di un Modello di Linguaggio potrebbe infatti dividersi in due fasi: la prima riguarda il training su un dataset; la seconda riguarda la messa in produzione del modello e la generazione di risposte (predizioni) su input nuovi, non visti precedentemente. Ad esempio, la creazione e l'addestramento di un modello in un'azienda sarebbe svolta dagli specialisti nel settore (training); una volta pronto si potrebbe mettere in produzione ed essere usato dai lavoratori per aiutarli nel loro lavoro (inference).

Tuttavia, la RAG vanilla presenta alcune limitazioni: riduce la versatilità degli LLM e, poichè la fase di recupero di dati (retrieval) avviene indiscriminatamente, ovvero senza verificare se effettivamente l'informazione è utile, ci può essere la generazione di risposte che sono off-topic o di scarsa qualità.

Per ovviare a questi problemi, sono nate recentemente alcune soluzioni, che possono anche essere combinate.

Self-RAG Il paper *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*[5] presenta un framework che realizza una RAG potenziata con self-reflection. Quest'ultima è la capacità di un modello di analizzare e migliorare le proprie risposte in modo autonomo. Per questa caratteristica prende il nome di Self-Reflective Retrieval-augmented Generation (**Self-RAG**).

Il punto principale dello studio è la creazione di un metodo ibrido, che unisca l'information-retrieval con i processi generativi; mentre i modelli antecedenti a questo studio si focalizzavano su uno o sull'altro.

Questo framework esegue il recupero di informazioni on-demand, ovvero può recuperare informazioni in diversi momenti durante la generazione, o non farlo affatto. In seguito, si valuta l'utilità delle generazioni (eseguite a partire dai dati recuperati), utilizzando un modello critico che genera dei *reflection token*, attraverso i quali si può controllare il LM. In particolare, dato un segmento (una frase), il framework può eseguire:

- **Retrieve:** genera un *retrieval token*, con il quale si valuta se è utile recuperare delle informazioni. Nel caso queste siano richieste, il LM chiama un modulo esterno per recuperarle (esempio: Database Vettoriale).
- **Generate:** se il recupero non è necessario, il modello genera il prossimo output, come un qualsiasi LM. Altrimenti, se il recupero di informazioni è necessario, il modello genera dei *critique token* per valutare l'affinità dei dati recuperati, e in seguito continua la generazione dell'output condizionando la risposta con le informazioni ottenute precedentemente.
- **Critique:** è la parte centrale del self-reflective. Si generano dei *critique token* per valutare i propri output e scegliere i migliori.

I risultati dimostrano come questo approccio batta gli LM classici, utilizzando LM con meno parametri, e i metodi convenzionali di RAG.

Corrective RAG Lo studio *Corrective Retrieval Augmented Generation*[35] analizza, per la prima volta, uno scenario precedentemente trascurato. Se da un lato il Self-RAG si concentra sul capire quando recuperare informazioni e valutare le risposte generate dal LM stesso, dall'altro lato CRAG analizza il caso in cui il recupero di documenti restituisca informazioni non accurate.

Il punto focale del framework Corrective RAG (**CRAG**) è il *Retrieval Evaluator*. Si tratta del componente designato per valutare, a partire dalla coppia query e documento, l'affinità di quest'ultimo. In particolare, ogni valutazione restituisce in

output un *relevance score*. Nel paper si utilizza un modello con 0.77B parametri, quindi si tratta di un modello molto leggero, se confrontato con gli LLM. Ad esempio, il modello critico del self-RAG [5] utilizza LLaMA-2 da 7B di parametri.

Dunque, in seguito al recupero di informazioni, possono essere innescate 3 azioni:

- **Correct:** un recupero di dati viene definito Correct se *almeno un* documento ottenuto restituisce un relevance score maggiore di un upper threshold. Anche se il documento risulta corretto, possono comunque esserci parti meno attinenti di altre. Per questo è eseguito un metodo di rifinitura della conoscenza, in cui il documento è diviso in strip. Ogni strip è valutato secondo il retrieval evaluator e infine il documento è ricomposto con le informazioni maggiormente affini, mentre gli strip meno significativi sono scartati.
- **Incorrect:** un recupero di dati viene definito Incorrect se *tutti* i documenti ottenuti restituiscono un relevance score minore di un lower threshold. In questo caso, per ovviare alla mancanza di conoscenza, si effettua una ricerca web. Anche in questo caso, i dati ottenuti sono rifiniti con il metodo descritto precedentemente.
- **Ambiguous:** è il caso alternativo ai due già citati. In questo caso si eseguono entrambe le azioni, ovvero il processo di rifinitura per i documenti recuperati e la ricerca web. I risultati poi vengono concatenati.

In seguito a queste azioni, avviene la costruzione del prompt con le informazioni acquisite.

Questo framework non è un'alternativa al self-RAG. Difatti, gli stessi ricercatori hanno effettuato degli esperimenti utilizzando un modello definito Self-Reflective Corrective RAG (**Self-CRAG**), il quale ha ottenuto i risultati migliori. Questo studio quindi ha realizzato un modello plug-and-play che può essere aggiunto a modelli già esistenti per migliorarne le performance.

Adaptive-RAG Il paper *Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity*[18] descrive un modello adattativo per task di Question Answering. Lo studio esplica un approccio che si adegua alla complessità della domanda in input. L'obiettivo della ricerca è indagare la possibilità di evitare l'eccessivo overhead di risorse e tempo. I metodi utilizzati precedentemente, infatti, si basano su approcci multi-step per rispondere alle domande in modo opportuno. Questa metodologia causa inefficienza in quanto vengono utilizzati eccessivamente il Retrieval e il LLM. Tuttavia, la complessità delle domande non è sempre la stessa, quindi l'approccio multi-step non è sempre richiesto. Ad esempio, per domande semplici come "Qual è la capitale della Francia?" è superfluo

attingere dalla conoscenza esterna, ma sarebbe sufficiente sfruttare la conoscenza interna del LLM.

Il punto centrale dello studio è l'utilizzo di un *classifier* che, in base alla complessità, restituisca A, B o C. In base al risultato si svolge un'operazione diversa:

- **Non Retrieval (A)**: non è necessario l'uso di informazioni recuperate da una fonte esterna. In questo caso, quindi, l'output sarà

$$\bar{a} = LLM(q)$$

dove \bar{a} è la risposta generata, e q la query inserita.

- **Single-step (B)**: si utilizza conoscenza esterna \mathbf{d} da una sorgente di conoscenza \mathbf{D} . In questo caso è necessario, ovviamente, un *Retriever* tale che $d = \text{Retriever}(q; \mathbf{D})$. Dunque, la risposta generata sarà:

$$\bar{a} = LLM(q, d)$$

- **Multi-step (C)**: è il caso in cui la domanda è molto complessa, e il LM necessita di informazioni esterne. In questo caso, l'interazione tra LLM e *Retriever* avviene in passaggi iterativi, utili affinché il modello comprenda la domanda e risponda in modo adeguato. Dunque, partendo dalla query iniziale \mathbf{q} , ad ogni iterazione i sono estratti d_i documenti e aggiunti al prompt di input. Quindi l'output generato ad ogni iterazione è

$$\bar{a}_i = LLM(q, d_i, c_i)$$

dove c_i rappresenta il contesto raggiunto all'iterazione i -esima. La composizione di c può variare; in questo studio si utilizza la concatenazione di tutte le precedenti generazioni e i documenti recuperati, quindi $c_i = (d_1, d_2, \dots, d_{i-1}, \bar{a}_1, \bar{a}_2, \dots, \bar{a}_{i-1})$. Anche il recupero di documenti varia in base al contesto, quindi $d_i = \text{Retriever}(q, c_i; \mathbf{D})$

I risultati sottolineano come questo modello abbia le stesse performance del Multi-step, riducendo sensibilmente l'overhead. La Figura 1 di Soyeong Jeong et al. *Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity*. Accessed: 14/07/2024. 2024. arXiv: 2403.14403 [cs.CL]. URL: <https://arxiv.org/abs/2403.14403> presenta un grafico in cui il tempo medio per query si avvicina sensibilmente al tempo del Non Retrieval.

Questi 3 approcci presentati, che ora rappresentano lo stato dell'arte della ricerca, possono essere combinati. Un esempio è quello di MistralAI⁴, che sottolinea come si possano prendere i concetti base di questi 3 framework ed unirli (Figura 3.3).

⁴https://github.com/mistralai/cookbook/tree/main/third_party/langchain

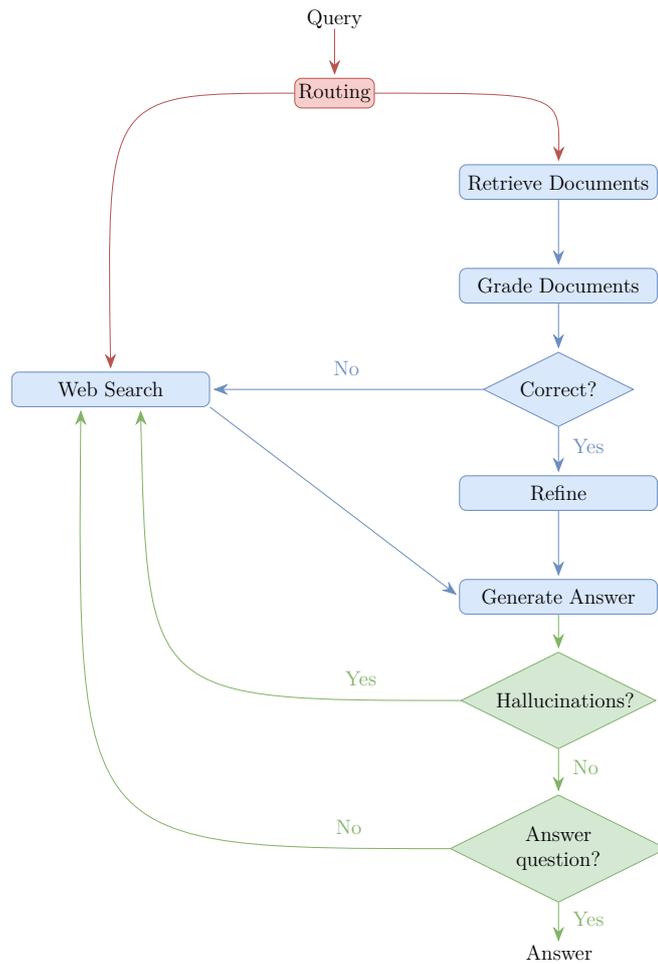


Figura 3.3: Rappresentazione di una possibile architettura che combini gli approcci di RAG descritti. In particolare, in rosso è la parte realizzabile a partire dall'Adaptive-RAG, in blu la parte del Corrective RAG (CRAG), in verde la parte del Self-RAG. Questo dimostra come i framework presentati non devono essere presi esattamente nel modo in cui sono descritti, ma possono essere adattati. L'architettura presentata è una rielaborazione dell'immagine presa da https://github.com/mistralai/cookbook/tree/main/third_party/langchain

3.3 Uso delle Ontologie

Lo studio dell'Ontologia parte dalla filosofia. In questo ambito, prende il significato di "the study or theory of the explanation of being"[33] (definizione che deriva direttamente dal greco: ὄντος e λόγος, dunque letteralmente "discorso sull'essere"). Tuttavia, non si ha una definizione univoca di Ontologia, ma varia in base al settore in cui si tratta.

Nel campo dell'informatica, questa materia prende piede negli anni Ottanta del XX secolo nel campo dell'intelligenza artificiale, e negli anni Novanta viene applicata nello studio dell'interoperabilità tra sistemi di conoscenza ⁵.

In questo caso, la definizione è "a formal explicit specification of a shared conceptualization"[33]. L'enunciazione racchiude diversi concetti fondanti: tutta la conoscenza espressa dalle ontologie deve passare attraverso specificazioni esplicite; l'ontologia deve essere una concettualizzazione, ovvero un concetto comprensibile universalmente; l'ontologia deve essere condivisa, ovvero il significato di un concetto deve essere accettato universalmente; l'ontologia deve essere formale, ovvero rappresentata in una logica processabile dalle macchine.

L'obiettivo dell'Ontologia è raggiungere una conoscenza comune e condivisa, che possa essere trasmessa tra persone e sistemi applicativi differenti. In questo modo è possibile creare la semantica in modo esplicito e generale.

L'Ontologia include 4 componenti:

- un **concetto** (classe o termine): è un gruppo astratto, un insieme di oggetti. È l'elemento base di un dominio e rappresenta un insieme che condivide delle proprietà. Segue una strutturazione simile a quella dell'Object Oriented, in quanto ogni classe può avere delle sottoclassi. Ad esempio, un'università (classe, concetto) può essere rappresentata come unione di più facoltà (sottoclassi).
- un'**istanza**: come nell'Object Oriented, rappresenta un'istanza specifica di una classe.
- una **relazione**: descrive la relazione tra un oggetto di partenza e un oggetto di arrivo.
- un **assioma**: utilizzato per imporre dei vincoli sulle classi o sulle istanze. Sono spesso formalizzati in logica del prim'ordine.

Nell'ambito informatico è necessario che l'Ontologia sia descritta in modo rigorosamente formale, affinché sia processabile da un calcolatore. Un'Ontologia, per essere definita in modo rigorosamente formale, deve fornire termini definiti con una semantica formale, teoremi e dimostrazioni delle proprietà, tra cui figurano completezza e correttezza. [33]

L'Ontologia ha come punto di forza la descrizione di modelli generalizzati, ma come limite l'assenza di informazioni specifiche in un dominio.

Per risolvere questo problema, è possibile realizzare un Knowledge Graph, basandosi sull'Ontologia. A partire da dati specifici (il cui modello generale è descritto attraverso un'Ontologia), è realizzabile un'istanza specifica di ogni relazione ontologica.

⁵Preso dalla definizione Treccani: [https://www.treccani.it/enciclopedia/ontologia-infornatica_\(Lessico-del-XXI-Secolo\)/](https://www.treccani.it/enciclopedia/ontologia-infornatica_(Lessico-del-XXI-Secolo)/).

Dunque, descrivendo tutte le relazioni, si verrà a creare una vera e propria rete, ovvero il Knowledge Graph.⁶

Sarà possibile utilizzarlo in sostituzione delle tabelle originarie che contenevano i dati: con il linguaggio SPARQL si potranno effettuare query, o creare nuove connessioni all'interno del grafo.

Come descritto precedentemente, il problema principale degli LLM (e dei Pre-trained Language Model) riguarda le allucinazioni. Difatti, essendo sostanzialmente macchine statistiche, potrebbero produrre risultati non veritieri. Gli errori mettono in evidenza l'assenza di ragionamento degli LLM, dovuti anche alle poche fonti esterne di conoscenza. Un'idea per risolvere questo problema è utilizzare i Knowledge Enhanced Pre-trained Language Models (KE-PLMs)[16]

L'idea è di sfruttare il valore semantico offerto dalle Ontologie e dai Knowledge Graphs per incorporare conoscenza negli LLM, al fine di raggiungere un ragionamento migliore. Nel paper si mostra ad esempio come si possa creare un Knowledge Aggregator che fonda conoscenza esterna con i token di input (KLMO), oppure creare un knowledge injector che integri la conoscenza in modo esplicito (KERM).

Un punto di unione tra Fine-tuning e conoscenza offerta dalle Ontologie è offerto dal paper *Fine-tuning Large Enterprise Language Models via Ontological Reasoning*[7]. Spesso il fine-tuning è effettuato sfruttando un dataset pubblico, oppure dati privati provenienti da database. Tuttavia, questo approccio ignora la "domain experience", ovvero quella conoscenza specifica del dominio su cui si sta lavorando. Questa conoscenza può essere invece catturata dagli *Enterprise Knowledge Graphs* (EKGs), poi utilizzati per effettuare ragionamento ontologico, da cui estrarre ulteriori dati per il fine-tuning. L'architettura descritta dallo studio utilizza VADALOG per computare task di ragionamento complessi a partire dai Knowledge Graphs. Il risultato è un miglioramento visibile sul dominio specifico sul quale si è svolto fine-tuning.

3.3.1 Question Answering (QA) su dati privati

L'uso di un LLM per interagire e ottenere informazioni a partire da dati privati (i.e. database SQL) è un problema noto come Question Answering. L'obiettivo è ricevere i giusti record senza l'utilizzo di alcuna query, ma facendo interfacciare LLM con il database SQL. L'utente quindi dovrà soltanto descrivere a parole i risultati che vuole ottenere. Tuttavia, sottoponendo una query ad un LLM, senza l'uso di esempi o dimostrazioni aggiuntive nel prompt, il risultato sarà deludente (i.e. usando GPT-4 si avrà un'accuratezza del 16,7% [32]).

⁶Tratto da <https://enterprise-knowledge.com/whats-the-difference-between-an-ontology-and-a-knowledge-graph/>

Un possibile approccio è quello di creare un ulteriore strato sul database, in modo da aggiungere contesto e dare valore semantico allo schema SQL. In particolare, questo layer è composto da due parti:

- **ontologia:** fornire valore semantico agli attributi e alle relazioni dello schema SQL.
- **mapping:** tradurre tutte le regole dello schema originale SQL nella corrispondente ontologia, definita precedentemente.

Lo schema descritto potrebbe essere definito usando RDF; in particolare le ontologie in OWL e il mapping in R2RML. È dunque possibile realizzare un Knowledge Graph, fondamentale per convertire la query SPARQL in una SQL e ottenere i risultati dal database.

Dunque, aggiungendo l'ontologia OWL al prompt, si riusciranno ad ottenere dei risultati di gran lunga migliori, con un'accuratezza che arriva al 54,2% [32].

Poichè questo approccio è molto semplice ed è stato utilizzato soltanto per dimostrare il possibile utilizzo delle ontologie in questo settore (e nonostante ciò ottenga buoni risultati), sono stati eseguiti ulteriori studi per raggiungere esiti migliori.

In particolare lo studio *Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue!*[4] aggiunge due ulteriori componenti in questa architettura, che si focalizzano sulla correzione dei risultati provenienti da un LLM. In questo caso, rilevando la non correttezza di una query SPARQL, sarà possibile correggerla, prima ancora di utilizzarla per richiedere informazioni al database SQL.

La figura 3.4 mostra l'architettura, basata principalmente su due componenti:

- **Ontology-based Query Check (OBQC):** prende in input la query SPARQL generata dal LLM e l'ontologia del Knowledge Graph, e verifica se la query è valida usando le regole basate sulla semantica dell'ontologia (il capitolo 3.1 del paper [4] esplora quali sono le regole utilizzate). Gli errori riscontrati possono riguardare: un path sbagliato, nel caso in cui la query non segua il percorso descritto dalle proprietà dell'ontologia; oppure una direzione sbagliata, nel caso in cui la query inverta la direzione di alcuni archi del grafo.
- **LLM Repair:** una volta individuato il problema della query, si utilizza un prompt formato dalla descrizione dell'errore e dalla query errata, e si chiede al LLM di risolverla. Questo tentativo di risoluzione potrebbe durare all'infinito, dunque è sufficiente impostare un limite, in seguito al quale si restituisce all'utente l'errore.

Usando questo metodo, l'accuratezza sale fino al 72,55%, con un miglioramento superiore del 50% rispetto al non utilizzare per nulla ontologie e KG. Inoltre si cerca

anche di eliminare le allucinazioni, il problema attualmente più grande degli LLM. Essendo oggetti probabilistici, il risultato generato potrebbe non essere corretto, e, in un contesto come il Question Answering, è un limite che dovrebbe essere superato. L'utilizzo di un OBQC permette la rilevazione delle allucinazioni. In questo modo è possibile notificare all'utente l'errore, e non restituire dati sbagliati o inventati.

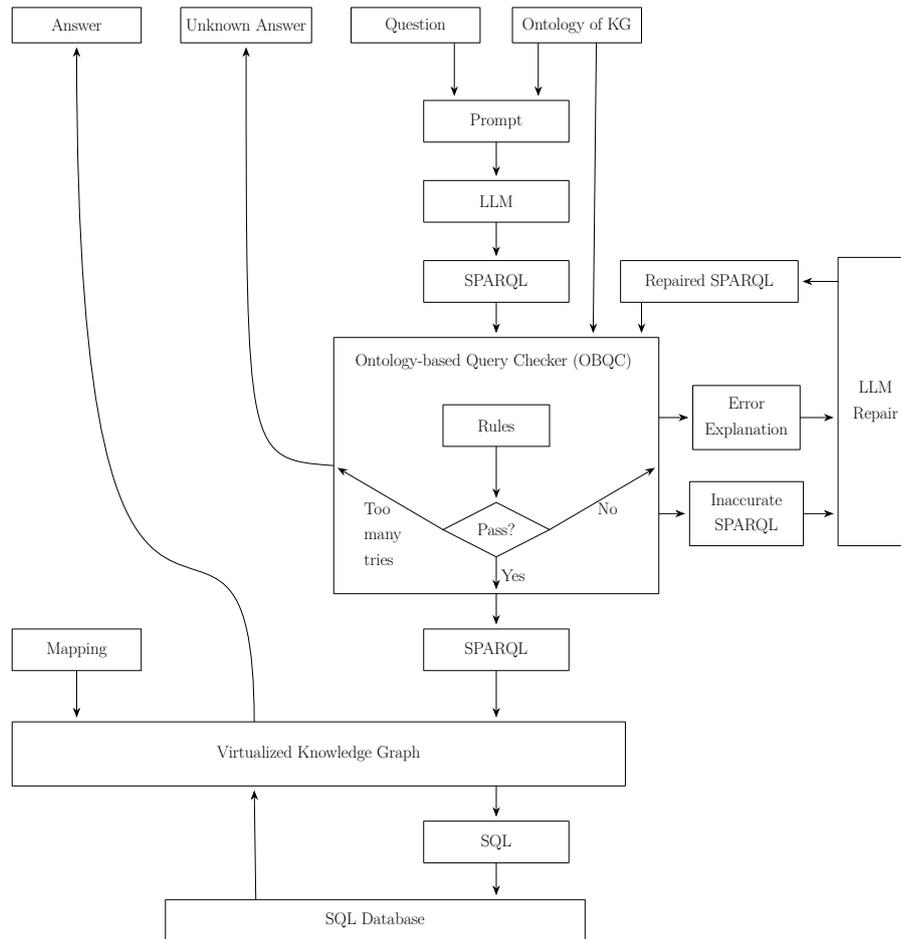


Figura 3.4: Architettura di un sistema per QA, basato sull'utilizzo di ontologie e KG. Immagine copiata e rielaborata da: Dean Allemang e Juan Sequeda. *Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue!* 2024. arXiv: 2405.11706 [cs.AI]. URL: <https://arxiv.org/abs/2405.11706>

3.4 Confronto fra i metodi

Lo studio che ha generato notevole interesse per l'intelligenza artificiale è *Language Models are Few-Shot Learners*[9], pubblicato nel 2020. Nel paper si dimostra come i modelli siano *Few-shot learners*. Rispetto al fine-tuning, in cui avviene l'aggiustamento dei pesi attraverso la discesa del gradiente per ogni esempio nel dataset, sono stati testati lo Zero-shot, il One-shot e il Few-shot, 3 approcci facenti parte dell'in-context-learning. Il metodo che ottiene i risultati migliori è il few-shot, in cui, oltre alla descrizione del task, vengono aggiunti una piccola quantità di esempi, e in seguito il prompt.

È stato raggiunto un risultato molto importante: è stato dimostrato che, inserendo nel prompt informazioni aggiuntive (come contesto e/o esempi), si riesce a far performare meglio gli LLM.

Recentemente, d'altro canto, fine-tuning e in-context learning sono stati confrontati ad armi pari, utilizzando stessi modelli ed esempi. [26]

Entrambe le metodologie hanno pro e contro. L'in-context learning è potenzialmente limitato dalla context size, ovvero la quantità di token processabili per ogni richiesta (in GPT-3 era 2048, ora diventata 8192); inoltre la lunghezza dei prompt influisce sul tempo di generazione della risposta. Necessita infine di un modello che sia già abbastanza efficace, altrimenti, non effettuando altre fasi di training, si otterranno dei risultati mediocri. Il fine-tuning presenta risultati simili all'in-context learning. L'aggiunta di parametri e l'aggiustamento di quelli già presenti, caratteristici del fine-tuning, permettono un miglior funzionamento, a parità di grandezza della rete. Per applicarlo è necessario l'intervento di un esperto che sappia fare l'allenamento di una rete.

L'in-context learning ha come punti di forza importanti la generalità e il riutilizzo, permesso dall'immutabilità dei pesi del proprio Modello. Il ramo dell'in-context learning con cui si raggiungono buoni risultati è quello del RAG.

Analizzando i problemi degli LLM, si può comprendere come agiscono questi due approcci. I problemi più rilevanti sono le allucinazioni e le mancanze di conoscenza esterna. Infatti, la conoscenza che è posseduta da un LLM è quella insita nei parametri, quindi si potrebbe considerare il fine-tuning come una modifica/aggiunta di conoscenza, il che lo rende verticale su un particolare campo. Queste caratteristiche rendono meno versatile il modello.

Dall'altro lato nell'in-context learning, e in particolare nella RAG, per ovviare a questo problema, viene aggiunta conoscenza proveniente dall'esterno. Tuttavia, prendendo in considerazione la RAG "vanilla", il problema della perdita di versatilità si verificherebbe comunque. Sono nati, quindi, altri numerosi metodi che permettono di non perdere generalità e, contemporaneamente, migliorare le performance nei casi di domande verticali in un settore, di cui si dispone conoscenza esterna. I modelli

analizzati in questo studio sono l'Adaptive-RAG, la Self-RAG e la Corrective-RAG, che ottengono buoni risultati e possono, inoltre, essere combinate assieme.

Il problema delle allucinazioni è il problema principale da superare, specialmente in un contesto di Question Answering. Tutti i metodi analizzati precedentemente cercano di superare questo limite. La soluzione migliore per raggiungere questo obiettivo è quello di dare semantica e significato alle parole. Questo processo è alla base del ragionamento, il che permetterebbe ai Modelli di Linguaggio di non commettere errori semantici. Per raggiungere questo scopo, si potrebbero usare Ontologie e grafi di conoscenza (i.e. Knowledge Graph). Tuttavia la ricerca non ha ancora esplorato soluzioni, oltre quelle descritte in precedenza.

Capitolo 4

Cheshire Cat AI

Il framework Cheshire Cat AI¹ è un progetto Open Source, fondato in Italia da Piero Savastano, per la realizzazione di assistenti AI verticali in un settore. Il cuore del progetto è il *core*², ovvero il backend dell'applicazione, che supporta un Dockerfile di default per velocizzare la messa in produzione dell'assistente.

L'idea nasce dalla volontà di costruire un framework che permettesse di creare chatbot con una semplicità paragonabile a quella di Wordpress nella realizzazione di siti web. Per riuscire nell'intento, il Cheshire Cat AI sfrutta gli hooks, ovvero un modo per modificare il comportamento standard nelle varie fasi della costruzione della risposta. Per comprendere cosa sono è necessario analizzare l'architettura, visibile nell'immagine 4.1.

I componenti core, ovvero il Mad Hatter, il White Rabbit, il RabbitHole, e la classe principale che contiene tutto, ovvero CheshireCat, sono stati implementati con il design pattern singleton. Questo è stato realizzato attraverso un decorator che modifica il metodo `__new__` di una classe³:

```
1 class singleton:
2     instances = {}
3
4     def __new__(cls, class_):
5         def getinstance(*args, **kwargs):
6             if class_ not in cls.instances:
7                 cls.instances[class_] = class_(*args, **kwargs)
8             return cls.instances[class_]
9
10        return getinstance
```

Code Listing 4.1: Implementazione singleton in Python

¹<https://github.com/cheshire-cat-ai>

²<https://github.com/cheshire-cat-ai/core>

³<https://github.com/cheshire-cat-ai/core/blob/main/core/cat/utils.py>

In Python, `__new__` è invocato nel momento della creazione di una nuova istanza della classe. Andando a modificare questo metodo, come mostrato, si crea un'istanza della classe solo se non esiste già.

Per gestire autenticazione e multiutenza è stata implementata la classe `StrayCat`. È incaricata di eseguire gli hook e i plugin selezionati, e tenere traccia del contesto e della Working Memory del singolo utente. Rappresenta la classe fondamentale del framework in quanto è quella acceduta da ogni endpoint, e quella che l'utente può usare nella scrittura di plugin. Contiene, inoltre, riferimenti all'unica istanza del `CheshireCat` (singleton), quindi può accedere ad ogni suo campo, come il LLM o il rabbit hole.

4.1 Architettura

L'utente può interagire con il backend del framework attraverso degli Endpoint HTTPS. Il più importante, che permette la conversazione con il chatbot, è `POST /message`, che sarà analizzato in seguito.

LLM e Embedder Componenti significativi sono il Large Language Model e l'Embedder. Il primo è il modello di linguaggio che genererà le risposte, il secondo si occupa di convertire delle stringhe in vettori, come analizzato nella pagina 13. Costituendo il nucleo del chatbot, le performance dipendono strettamente dal modello e dall'embedder utilizzato. Un punto a favore del framework è l'indipendenza dal LLM e dall'Embedder: essendo componenti esterne, ovvero non gestite dal framework stesso, è possibile collegare qualsiasi Modello o Embedder, in base alle proprie esigenze e capacità economiche. Per raggiungere questo livello di astrazione e indipendenza è stato usato il design pattern Factory. Rientra nella categoria dei design pattern creazionali, ovvero dei metodi per creare nuovi oggetti nascondendo informazioni sulla creazione e astraendo dall'istanziamento. In particolare, il design pattern Factory si utilizza quando si hanno le informazioni necessarie per la costruzione dell'oggetto soltanto a runtime. Nella pratica, si crea un'interfaccia per creare l'oggetto e lasciare alla sottoclasse la decisione di quale classe istanziare. In questo modo è possibile delegare la scelta del LLM da utilizzare a runtime, in base alle scelte dell'amministratore.

Attualmente, di default, sono supportati: OpenAI and Azure OpenAI, Cohere Ollama (soltanto per gli LLM), HuggingFace TextInference API (soltanto per gli LLM), Google Gemini, Qdrant FastEmbed (soltanto per l'Embedder); ma possono essere ampliati attraverso la scrittura di plugin.

Ad esempio, se si considerasse un ambito aziendale, sarebbe di fondamentale importanza conservare ed utilizzare i dati soltanto all'interno dell'azienda stessa. Non si potrebbe, quindi, usare ChatGPT di OpenAI, ma è necessario l'utilizzo di un model-

lo locale, hostato dall'azienda stessa. Per comprenderne l'importanza, si potrebbe effettuare un paragone con una macchina: il modello rappresenterebbe il motore, il framework tutti gli altri componenti.

La presenza di questi componenti esterni dona flessibilità al framework, di fondamentale importanza nel ricoprire più casi d'uso possibili.

Vector Database Il Database vettoriale è fondamentale nella memorizzazione dei vettori e nel recupero di informazioni memorizzate precedentemente. Il framework utilizza Qdrant, un database open source progettato appositamente per le AI, che utilizza l'algoritmo Hierarchical Navigable Small World (HNSW) per la ricerca di vettori simili (approximate nearest neighbors - ANN).

Rabbit Hole Il rabbit hole si occupa dell'acquisizione di informazioni. Possono essere codificate attraverso un JSON (endpoint `/rabbithole/memory`), un file (endpoint `/rabbithole`) di estensione .html, .md, .txt o .pdf, o una pagina web (endpoint `/rabbithole/web` con un URL come input). I dati, come ad esempio il testo contenuto in un pdf, sono ottenuti dai file attraverso opportuni parser. Una volta completata l'estrazione, si procede con la separazione dell'intero contenuto in chunks, la vettorizzazione (attraverso l'embedder) e la memorizzazione di ogni singolo chunk⁴. Attraverso la scrittura di plugin avviene la personalizzazione del rabbit hole: ad esempio, è possibile modificare lo splitter del testo di default per cambiare la dimensione o l'overlap dei chunks, oppure modificare o aggiungere parser, per variare l'analisi delle informazioni o permettere l'estrapolazione di informazioni da file di estensioni diverse.

Tutte le informazioni, elaborate dal rabbit hole e memorizzate, potranno essere richiamate ed inserite al prompt nella generazione della risposta di una domanda dell'utente.

Long Term Memory La Long Term Memory (LTM) rappresenta la memoria persistente del framework. È formata da tre parti:

- **Episodic Memory:** contiene tutto ciò che l'utente ha chiesto al chatbot, con le relative risposte.
- **Declarative Memory:** contiene tutte le informazioni memorizzate attraverso il rabbit hole. Verranno estratte da questa memoria le informazioni rilevanti e aggiunte al prompt per condizionare la risposta.

⁴La fase di divisione dei dati in piccole porzioni è detta chunking. Lo splitter è il componente delegato nella separazione del testo, ovvero nella creazione dei chunks.

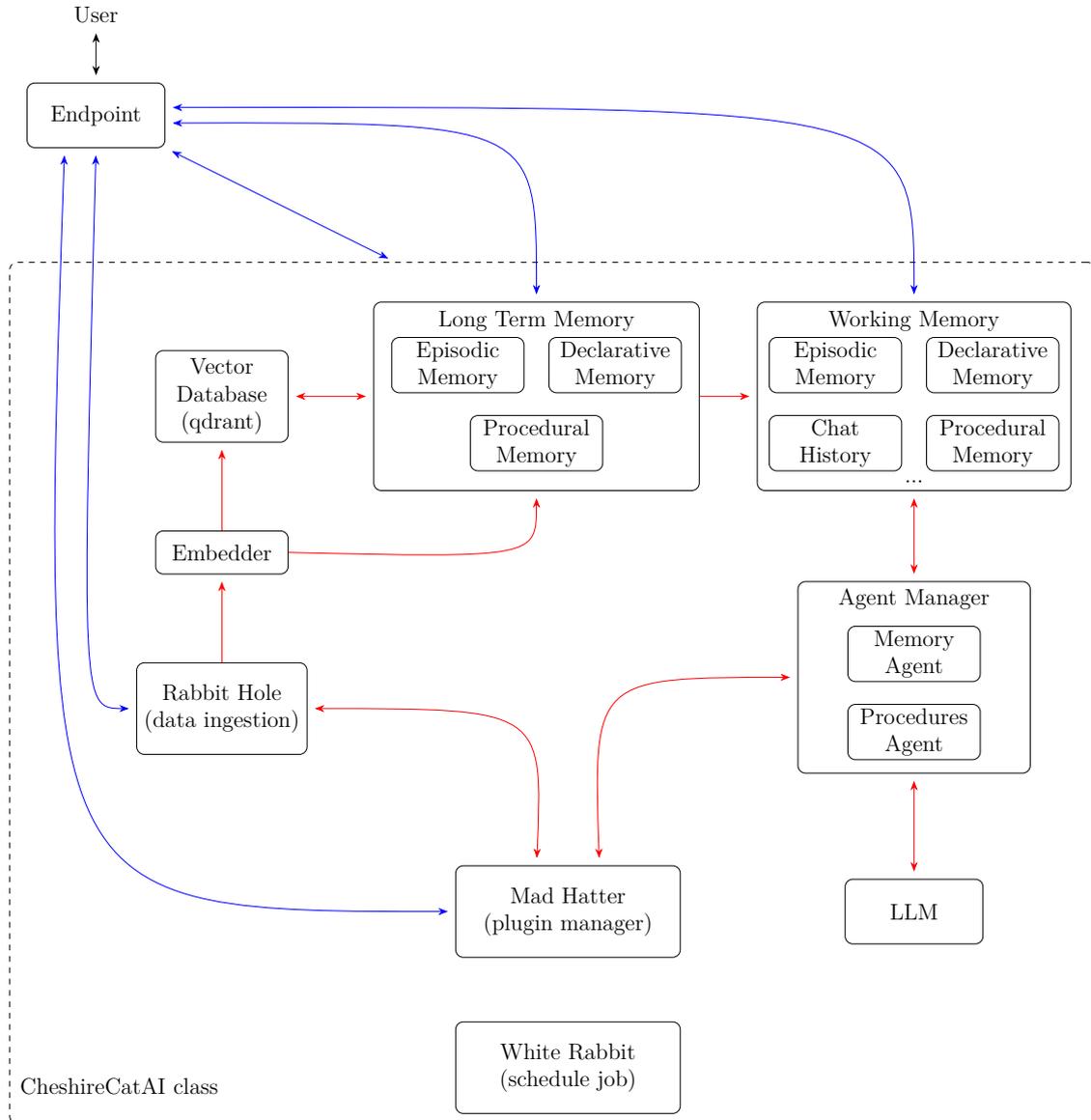


Figura 4.1: Architettura del core del framework Cheshire Cat AI. Le frecce blu indicano gli endpoint per effettuare richieste HTTP (ad esempio è possibile passare documenti attraverso una chiamata POST all'endpoint `/rabbithole/`). Le frecce rosse indicano quali componenti si interfacciano tra loro (ad esempio a interagire con il Large Language Model sono soltanto gli Agenti). Il bordo punteggiato indica la presenza di ogni componente nella classe Python CheshireCatAI.

- **Procedural Memory:** contiene i tools e come attivarli. Un tool è una funzione python che il framework esegue in seguito ad un evento scatenante, descritto in docstring.

Dato l'embedding della domanda dell'utente, da ognuna delle tre memorie vengono estratte le informazioni e salvate nella Working Memory, che, oltre a questi dati, conterrà anche la storia della conversazione. Le tre domande e risposte più recenti saranno usate come maggiorazione del prompt.

Mad Hatter Mad Hatter è il componente predisposto per la gestione dei plugin. In particolare si occupa di caricarli, dare la proprietà ed eseguirli. Inoltre accede al registro (<https://registry.cheshirecat.ai>) contenente tutti i plugin pubblicati dagli utenti, che si possono scaricare ed eseguire.

White Rabbit Il White Rabbit è il componente che si occupa di schedulare computazioni da eseguire in un tempo preciso. Queste comprendono eventi da scatenare soltanto una volta, oppure periodicamente usando i cronjob.

Agent Manager L'Agent Manager è il componente che si interfaccia con il Large Language Model. In particolare esegue la pipeline composta da: costruzione del prompt, in seguito al recupero e alla formattazione di alcuni dati; invio del prompt al Modello e analisi dell'output.

È composto da due sottoagenti. Il primo è il Procedural Agent, che è invocato per eseguire la tool chain. Una volta recuperati i tool dalla memoria procedurale, l'agente costruisce l'istruzione prompt, usato come input del LLM per scegliere quale, tra i tool estratti, utilizzare. Si utilizza il prompt di default di langchain⁵:

```

1 instructions = """
2 To use a tool, please use the following format:
3
4 Thought: Do I need to use a tool? Yes
5 Action: the action to take, should be one of [{tool_names}]
6 Action Input: the input to the action
7 Observation: the result of the action
8
9 When you have a response to say to the Human, or if you do not need to
  use a tool, you MUST use the format:
10
11 Thought: Do I need to use a tool? No
12

```

⁵https://api.python.langchain.com/en/latest/agents/langchain.agents.conversational.base.ConversationalAgent.html?highlight=prompt/format_instruction

13 AI: [your response here] """

Code Listing 4.2: Instruction Prompt

Se non c'è alcun tool recuperato dalla memoria procedurale, si passa al secondo agente, ovvero il Memory Agent.

Questo agente si occupa della Memory Chain: si utilizzano le informazioni recuperate dalla memoria episodica e dichiarativa per completare il prompt. Il prompt è formato da due parti: il prefisso, che specifica il ruolo del chatbot e in che modo dovrebbe rispondere; il suffisso, che include la memoria episodica e quella dichiarativa, aggiunge lo storico della conversazione e inserisce la domanda dell'utente.⁶

```

1 prefix = """You are the Cheshire Cat AI, an intelligent AI that passes
  the Turing test.
2 You are curious, funny and talk like the Cheshire Cat from Alice's
  adventures in wonderland.
3 You answer Human with a focus on the following context.
4 """
5 suffix = """
6 # Context
7
8 {episodic_memory}
9
10 {declarative_memory}
11
12 ## Conversation until now: {chat_history}
13 - Human: {input}
14 - AI: """

```

Code Listing 4.3: Prefisso del Main Prompt

L'output restituito dal LLM è mandato all'utente.

Anche in questo caso l'esecuzione degli agenti può essere personalizzata. Ad esempio, è possibile modificare il prefisso del prompt per modificare la "personalità" del Modello.

⁶https://github.com/cheshire-cat-ai/core/blob/main/core/cat/looking_glass/prompts.py

4.2 Messaggio

La funzione principale di un chatbot è rispondere alle domande degli utenti. Per svolgere questo compito si usa l'endpoint POST `/message`. La figura 4.2 rappresenta un flowchart di ciò che avviene per ogni messaggio inviato dall'utente.

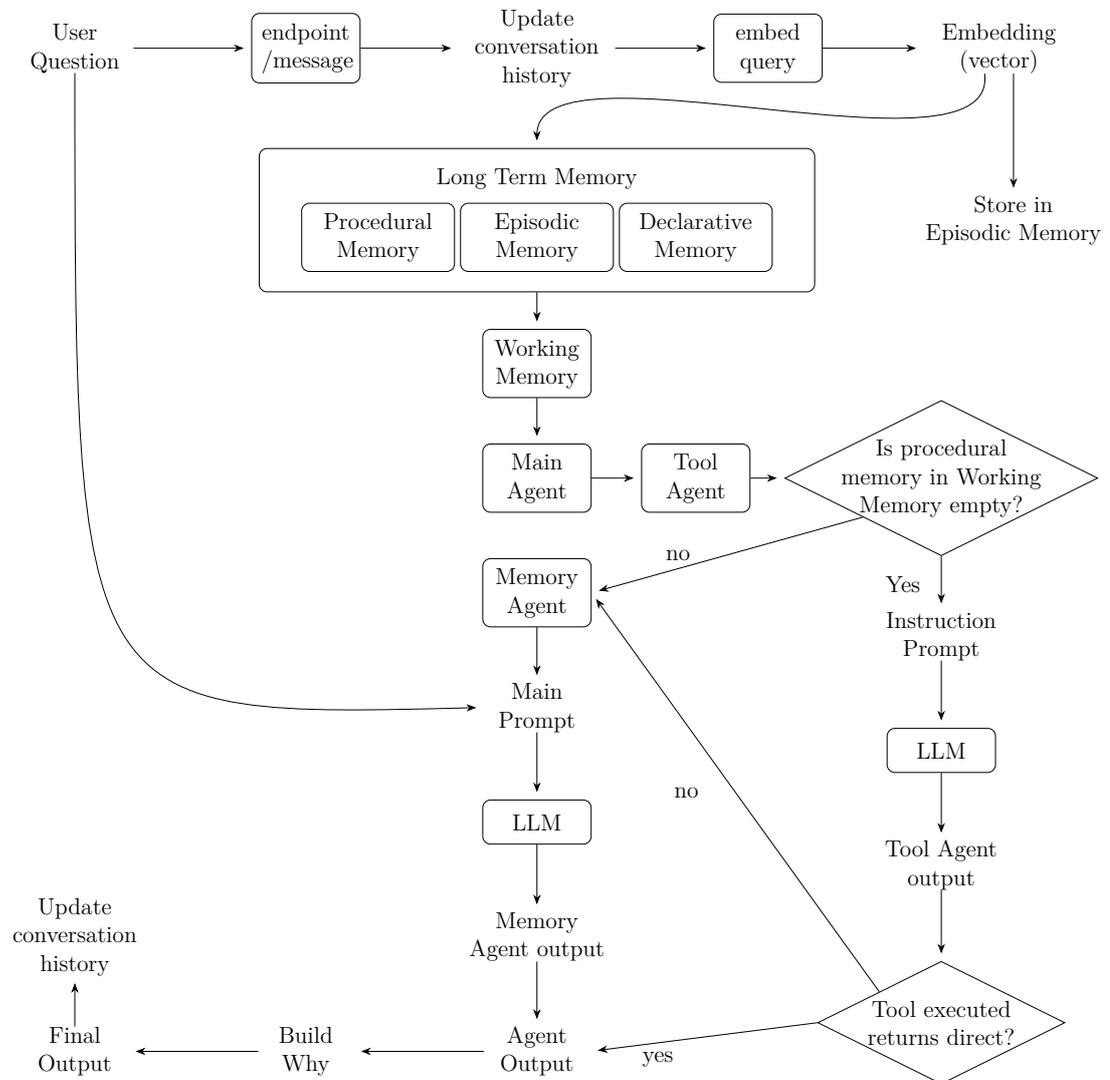


Figura 4.2: Flowchart dell'endpoint `/message`

Dopo aver salvato la domanda dell'utente nella Working Memory, l'Embedder converte la stringa di input in un vettore. Si provvede a salvare quest'ultimo nella Memoria Episodica e in seguito si recuperano tutti i dati affini. La selezione dipen-

de dal tipo di similarità utilizzata, e, in base al plugin attivo, sono presi soltanto i primi n documenti che superano un certo threshold. Una volta recuperate le informazioni dalla memoria Procedurale, Episodica e Dichiarativa, ne viene eseguita una copia nella Working Memory, che già contiene la history della conversazione ed altri metadati relativi al singolo utente.

Segue poi l'esecuzione degli Agenti. Il Tool Agent è eseguito soltanto se sono stati recuperati tools nella fase precedente (ovvero dalla memoria procedurale). In questo caso c'è la costruzione dell' *instruction prompt*, attraverso cui il Modello di Linguaggio sceglie la funzione da utilizzare. Se il tool è stato ideato per terminare in seguito all'esecuzione, l'output del Tool Agent è usato come output e dunque mandato all'utente. Se si prendesse, ad esempio, un plugin che integra la domotica, potrebbe contenere un tool per l'accensione o lo spegnimento di luci. Dato che il messaggio dell'utente ha come fine un'azione da svolgere, e non ottenere una risposta, il LLM non è invocato.

Se, in caso contrario, non sono state recuperate informazioni dalla memoria procedurale, oppure il tool non termina direttamente l'esecuzione, il secondo agente entra in esecuzione costruendo il *main prompt*. Il prompt svolge un ruolo primario, avendo la capacità di condizionare la risposta [9]. Ricevuto l'output del Modello, si costruisce il *why*, ovvero un dizionario contenente informazioni sui dati recuperati e utilizzati dal modello per rispondere.

La presenza di plugin, e quindi principalmente di hook e tool, modifica il comportamento standard. Ad esempio si può variare il prefisso o il suffisso del prompt; possono essere eseguite delle azioni personalizzate prima o dopo l'esecuzione degli agenti; si può restituire all'utente una risposta rapida prima di eseguire il primo agente; ecc.

4.3 Plugin

Il potenziale del Cheshire Cat AI risiede nella possibilità di scrittura di plugin. È possibile modificare ed estendere il framework per renderlo specializzato a svolgere un singolo ruolo, proprio come Wordpress è stato creato per la realizzazione di siti web. La presenza di un registro pubblico⁷ permette a qualsiasi admin di scaricare ed installare plugin scritti da altri utenti. Alcuni esempi includono: l’espansione dei file che possono essere ingeriti dal rabbit hole; l’espansione di Large Language Model supportati; l’aggiunta di servizi esterni come la compatibilità con la domotica; la possibilità di cercare online dei contenuti e non solo. Le funzioni principali nella scrittura di un plugin sono gli Hook, i Tool e i Form.

Hook La definizione di Hook visibile nella documentazione è: "Hooks are callback functions that are called from the Cat at runtime."^[1]

L’hook è quindi una funzione che verrà invocata, attraverso il mad hatter, durante il runtime. In particolare, si può influire sul flusso regolare eseguito dopo la ricezione della domanda (Tabella 4.1), sul comportamento degli agenti (Tabella 4.2) o sull’acquisizione di documenti, quindi sul rabbit hole (Tabella 4.3)

Tool Un tool è una funzione che è chiamata dal Large Language Model: gli input sono strutturati in modo da poter essere generati dal modello, e l’output è pensato per essere usato in seguito dal modello^[2].

In particolare, ogni tool deve contenere una stringa docstring che descriva quando dovrebbe essere invocata la funzione. Questa descrizione risulta fondamentale, in quanto è inserita nell’instruction prompt per chiedere al modello se eseguire o meno il tool. Nel Cheshire Cat AI, inoltre, la descrizione è utilizzata per generare l’embedding da salvare nella memoria procedurale.

Un tool, ad esempio, potrebbe essere la richiesta del meteo. Domandando "qual è il meteo oggi a Cercemaggiore?", il Modello chiamerebbe l’API del meteo con input Cercemaggiore, e l’output poi sarebbe rielaborato dal modello per rispondere all’utente. La forza risiede nella capacità di formattare il testo in una chiamata API ed eseguirla.

La possibilità di eseguire tool da parte di un Modello di Linguaggio è stata studiata nel paper *Toolformer: Language Models Can Teach Themselves to Use Tools*^[31]. Nell’articolo è stato esplorato l’utilizzo di tool per aiutare il Modello a risolvere funzionalità considerate di base, che a volte si rivelano problematiche per la generazione di allucinazioni. Prendendo in esame i problemi presenti nell’ambito dei LLM (impossibilità di accedere ad informazioni in tempo reale, tendenza ad inventare notizie, difficoltà di comprensione di linguaggi poco utilizzati nella fase di training, carenze

⁷<https://github.com/cheshire-cat-ai/plugins>

in ambito matematico, inconsapevolezza dello scorrimento del tempo), i ricercatori si sono focalizzati sulla risoluzione degli stessi, utilizzando tool esterni: un sistema di question answering, un motore di ricerca Wikipedia, una calcolatrice, un calendario e un sistema di traduzione.

Lo studio si basa sull'in-context learning [8]: sono sufficienti pochi esempi dell'utilizzo di un API per far comprendere al Modello come utilizzarle.

Form Un form è un particolare tool, attraverso il quale il Cheshire Cat AI può richiedere informazioni all'utente. Ad esempio, potrebbe essere usato da un parrucchiere per prendere gli appuntamenti. I dati utili in questo caso sarebbero orario, tipo di servizio, come ad esempio "taglio" o "colore", e nome del cliente. Se si realizzasse questo plugin, l'implementazione della classe che descrive un appuntamento sarebbe⁸:

```

1     class HairdresserAppointment (BaseModel):
2         client_name: str
3         service: str
4         date: str

```

Code Listing 4.4: Esempio Form: classe per il contenuto JSON da inviare all'API

Per attivare e stoppare il form si utilizzano delle frasi, seguendo lo stesso principio del docstring per i tool. Il chatbot guiderà la compilazione dei dati necessari attraverso l'utilizzo di una macchina a stati, anche questa personalizzabile modificando i metodi della classe base. Continuando l'esempio di prima, una semplice implementazione del form potrebbe essere:

```

1     class HairdresserForm (CatForm):
2         description = "Hairdresser Appointment"
3         model_class = HairdresserAppointment
4         start_examples = [
5             "I need to cut my hair",
6             "My hair color is old, I need to renew it"
7         ]
8         stop_examples = [
9             "I don't want to cut my hair",
10            "I like my hair color",
11        ]
12        ask_confirm = True
13
14        def submit(self, form_data):
15            # Fake API call to take appointment
16            response = requests.post(
17                "https://fakehairdresser/appointment",
18                json={

```

⁸implementazione a scopo illustrativo. Non è stata implementata realmente né testata.

Hook name	When is called
Before Cat bootstrap	Intervene before the Cat's instantiate its components
After Cat bootstrap	Intervene after the Cat's instantiated its components
Before Cat reads message	Intervene as soon as a WebSocket message is received
Cat recall query	Intervene before the recall query is embedded
Before Cat recalls memories	Intervene before the Cat searches into the specific memories
Before Cat recalls episodic memories	Intervene before the Cat searches in previous users' messages
Before Cat recalls declarative memories	Intervene before the Cat searches in the documents
Before Cat recalls procedural memories	Intervene before the Cat searches among the action it knows
After Cat recalls memories	Intervene after the Cat's recalled the content from the memories
Before Cat stores episodic memories	Intervene before the Cat stores episodic memories
Before Cat sends message	Intervene before the Cat sends its answer via WebSocket

Tabella 4.1: Hook disponibili per modificare il comportamento alla ricezione di un messaggio. La tabella è stata copiata dalla documentazione (Accessed: 16/08/2024. URL: <https://cheshire-cat-ai.github.io/docs/plugins/hooks/>)

```

19         # Build json from form_data
20     })
21     response.raise_for_status()
22     time = response.json()["appointment_time"]
23
24     # Return a message to the user with appointment details
25     return {
26         "output": f"Appointment time is {time}"
27     }

```

Code Listing 4.5: Esempio Form: classe base che gestisce il riempimento del JSON

Un volta quindi riempito il form (il messaggio mostrato all'utente sullo stato può essere modificato), si potrà inviare ed attendere la risposta. Anche la risposta può essere gestita opportunamente: supponendo che nella data inserita dal cliente non ci siano posti disponibili, si potrebbe comunicare e chiedere una modifica del form.

Hook name	When is called
Before agent starts	Prepare the agent input before it starts
Agent fast reply	Shorten the pipeline and returns an answer right after the agent execution
Agent prompt prefix	Intervene while the agent manager formats the Cat's personality
Agent prompt suffix	Intervene while the agent manager formats the prompt suffix with the memories and the conversation history
Agent allowed tools	Intervene before the recalled tools are provided to the agent
Agent prompt instructions	Intervent while the agent manager formats the reasoning prompt

Tabella 4.2: Hook disponibili per modificare il comportamento degli agenti, come la creazione del prompt. La tabella è stata copiata dalla documentazione Accessed: 16/08/2024. URL: <https://cheshire-cat-ai.github.io/docs/plugins/hooks/>

Hook name	When is called
Rabbit Hole instantiates parsers	Intervene before the files' parsers are instiated
Before Rabbit Hole insert memory	Intervene before the Rabbit Hole insert a document in the declarative memory
Before Rabbit Hole splits text	Intervene before the uploaded document is split into chunks
After Rabbit Hole splitted text	Intervene after the Rabbit Hole's split the document in chunks
Before Rabbit Hole stores documents	Intervene before the Rabbit Hole starts the ingestion pipeline
After Rabbit Hole stores documents	Intervene after the Rabbit Hole ended the ingestion pipeline
Rabbit Hole instantiates parsers	Hook the available parsers for ingesting files in the declarative memory
Rabbit Hole instantiates splitter	Hook the splitter used to split text in chunks

Tabella 4.3: Hook disponibili per modificare il comportamento del rabbit hole, ovvero l'ingestione dei documenti La tabella è stata copiata dalla documentazione Accessed: 16/08/2024. URL: <https://cheshire-cat-ai.github.io/docs/plugins/hooks/>

Capitolo 5

Implementazione di un plugin

Per dimostrare la semplicità e la potenza della scrittura di plugin, questo capitolo descriverà, in ogni singolo passaggio, il codice Python utilizzato e i risultati.

Per la scrittura del plugin è stato utilizzato il template fornito dagli sviluppatori del cheshire cat AI, in modo da seguire gli standard per la seguente pubblicazione sul registro pubblico.

L'idea del plugin è la segmentazione della memoria dichiarativa in gruppi, in modo che il caricamento e il recupero delle informazioni avvenga in base al gruppo attivo. Ogni utente ha la possibilità di creare un gruppo, senza avere alcun permesso specifico. Questa caratteristica consente al singolo utente di creare due o più contesti completamente separati.

In modo duale è disponibile l'eliminazione del gruppo, in questo caso delegata soltanto all'utente che lo aveva creato.

La feature principale del plugin è la possibilità di formare gruppi contenenti più utenti. Ad esempio, se un professore volesse creare un gruppo per il proprio corso, potrebbe caricarvi tutti i pdf e il materiale utilizzato. In seguito, potrebbe inserire tutti gli studenti al gruppo, con lo scopo di fornire un chatbot che possa rispondere alle domande, attingendo esclusivamente al materiale fornito.

L'aggiunta di account è disponibile soltanto per chi ha i permessi di lettura sugli altri utenti.

Il principale punto di forza del plugin è la configurazione in modo conversazionale, apprezzata anche dal creatore del Cheshire Cat AI. Attraverso l'uso dei Form, si può scatenare l'aggiunta o la rimozione di un utente semplicemente inviando, in chat, un messaggio del tipo "aggiungi un utente al gruppo / rimuovi un utente dal gruppo". La memorizzazione dei gruppi, e il mantenimento degli utenti all'interno dei gruppi, è gestita attraverso sqlite.

Il plugin è ora disponibile nel registro pubblico del Cheshire Cat AI, in seguito alla Pull Request accettata dai membri del team¹.

Form I form implementati riguardano l'aggiunta e la rimozione degli utenti in un gruppo.

Per implementarli, si parte dalla definizione del modello, che include i campi necessari per effettuare le operazioni di inserimento o rimozione degli utenti, quali "id utente" e "gruppo". Per entrambi i campi si possono inserire dei controlli, in modo che vengano selezionati soltanto utenti e gruppi esistenti. La gestione degli utenti è interna al Cheshire Cat, quindi si utilizza il db di default²; invece, per la gestione dei gruppi, si utilizzano le tabelle sql implementate. I controlli si basano sul decoratore *field_validator* reso disponibile da pydantic.

```

1
2 class UserInfo(BaseModel):
3     user_id: str
4     group_name: str
5
6     @field_validator('user_id')
7     @classmethod
8     def userExists(cls, value: str)-> str:
9         users = list(map(lambda x: x[1]['username'], crud.get_users().
10 items()))
11         if value not in users:
12             raise ValueError(f"user '{value}' does't exists")
13         else:
14             return value
15
16     @field_validator('group_name')
17     @classmethod
18     def userExists(cls, value: str)-> str:
19         groups = sqldb().getGroups()
20         log.info(groups)
21         if value not in groups:
22             raise ValueError(f"group '{value}' does't exists")
23         else:
24             return value

```

Code Listing 5.1: Implementazione del modello per il form di aggiunta/rimozione di utenti dal gruppo

Il modello è utilizzato nella classe che implementa effettivamente il form, contrassegnata dal decoratore *@form*.

¹Il codice sorgente del plugin è presente nella repository https://github.com/Tiestaa/ccat_groups.

²Per il momento, il Cheshire Cat AI usa TinyDB per memorizzare tutti i metadati, tra cui le informazioni sugli utenti.

La classe prevede l'utilizzo di due array di stringhe per scatenare o bloccare il Form, come succede anche nei Tool.

```

1 @form
2 class UserInsertGroupForm(CatForm):
3     description = "Add user in group"
4     model_class = UserInfo
5     start_examples = [
6         "insert user in group",
7         "add user in group",
8         "extend the group"
9     ]
10
11     stop_examples = [
12         "stop adding a user in group",
13         "stop inserting user",
14     ]
15
16     ask_confirm = True
17
18     def next(self):
19         user_permission = getUserPermission(self.cat.user_id)
20         if ("USERS" not in user_permission) or ("READ" not in
21 user_permission['USERS']):
22             return {"output": "Permissions not valid"}
23         else:
24             return super().next()
25
26     def submit(self, form_data):
27         db = sqldb()
28         user_id = form_data["user_id"]
29         group_name = form_data["group_name"]
30
31         ret = db.insert_user_in_group(group_name, user_id, self.cat.
32 user_id)
33
34         return {
35             "output": ret
36         }

```

Code Listing 5.2: Implementazione del form per l'aggiunta di un utente al gruppo.

Gli unici due metodi sovrascritti alla classe base (*CatForm*) sono:

- *next*: utilizzato per passare da uno stato all'altro durante l'esecuzione del Form. In base all'automa rappresentato da questi stati, si eseguiranno azioni diverse. Rispetto all'implementazione standard, si esegue un controllo sui permessi, che permette di stroncare, sul nascere, i tentativi di inserimento o rimozione non autorizzati. Se si supera il controllo, si esegue il comportamento standard (*super().next()*).

- *submit*: è necessario implementare questo metodo per decidere quale operazione effettuare, una volta raccolti i dati. Nell'esempio mostrato, l'operazione eseguita è l'inserimento delle informazioni all'interno dei database.

Rabbit Hole Hooks Gli hook sono utilizzati per aggiungere i metadati necessari per contrassegnare il gruppo, modificando il comportamento standard del Rabbit Hole. Nello specifico, il controllo del gruppo attivo avviene mediante l'istanza del Cat attiva in quel momento³.

Nel caso in cui, nel momento dell'upload dei file, non ci sia un gruppo attivo, il Rabbit Hole aggiunge i metadati, ovvero l'id dell'utente che ha effettuato la richiesta, necessari alla gestione della multiutenza.

Allo stato attuale del plugin, soltanto l'utente che ha creato un gruppo può caricare dei documenti.

Il controllo sulla proprietà del gruppo avviene nell'hook *before_rabbithole_splits_text*: nel caso in cui la verifica fallisse, l'utente emittente della richiesta riceverà un messaggio di errore in chat.

```

1 @hook
2 def before_rabbithole_insert_memory(doc, cat):
3     # insert metadata about user and group
4     if WORKINGMEMORYKEY in cat.working_memory and cat.working_memory[
5         WORKINGMEMORYKEY] != None:
6         doc.metadata['profile'] = cat.working_memory[WORKINGMEMORYKEY
7     ]
8     else :
9         doc.metadata["user_id"] = cat.user_id
10    return doc
11
12 @hook(priority = 5)
13 def before_rabbithole_splits_text(docs, cat):
14     if (WORKINGMEMORYKEY in cat.working_memory and cat.working_memory
15         [WORKINGMEMORYKEY] != None) and not sqldb().isGroupOwner(cat.
16         working_memory[WORKINGMEMORYKEY], cat.user_id):
17         cat.send_ws_message("You don't have the permissions to upload
18         document in this group.", msg_type='chat')
19     return []
20    return docs

```

Code Listing 5.3: Implementazione Rabbit Hole Hooks

Altri Hooks Se da un lato l'aggiunta degli utenti ad un gruppo avviene in via conversazionale attraverso l'uso di form, dall'altro, il cambio, la creazione o la rimozione di un gruppo, avvengono attraverso l'uso di comandi.

³In particolare si usa l'istanza dello *stray_cat*, che mantiene la sessione dell'utente.

I comandi disponibili sono:

```

1 """
2 [@p group_name]  - changes the current group to group_name
3 [@c group_name]  - creates a group named group_name
4 [@r group_name]  - deletes the group named group_name
5 [@p]             - prints active group
6 [@l]             - gets groups' list for current session
7 [@d]             - deactivates profiling. Active groups is set to None
8 [@h]             - help
9 """

```

Code Listing 5.4: Comandi disponibili per la gestione dei gruppi

Per implementare i comandi all'interno del chatbot, basta sfruttare l'hook *agent_fast_reply*. Sono considerati comandi tutti i messaggi che iniziano per '@'.

```

1 def agent_fast_reply(fast_reply , cat):
2     ...
3     if cat.working_memory.user_message_json.text.startswith("@"):
4         ...
5         command = cat.working_memory["user_message_json"]["text"]
6         match command[:2]:
7             # change profile
8             case "@p":
9                 ...
10            # create profile
11            case "@c":
12                ...
13            # remove profile
14            case "@r":
15                ...
16            # list all groups
17            case "@l":
18                ...
19            # deactivate profile
20            case "@d":
21                ...
22            # help
23            case "@h":
24                ...
25            case _:
26                ...
27
28     return fast_reply

```

Code Listing 5.5: Implementazione hook *agent_fast_reply*

Ogni comando, in seguito ai controlli dei permessi e alle esecuzioni delle operazioni, inserisce nel dizionario *fast_reply* la key *output*. In questo modo, quando l'esecuzione

dell'hook finisce, è già presente la stringa da restituire all'utente, evitando passaggi superflui.

5.1 Demo

Per la demo non è stato scritto soltanto il plugin illustrato precedentemente. Un secondo plugin implementa due aspetti:

- Uso del **Semantic Chunking**. Di default, il Cheshire Cat AI utilizza un chunking del testo basato su due parametri: la dimensione del chunk e l'overlap⁴. Diversamente, in questo plugin si è scelto di utilizzare uno splitter sperimentale che si basa sulla semantica di ogni frase. Il SemanticChunker⁵ confronta le varie frasi ed esegue delle operazioni di taglio e ricomposizione in base alla semantica calcolata dall'embedder.
- Aggiunta dei metadati riguardanti la posizione del chunk nel documento originale. In questo modo, il chatbot potrà aggiungere i riferimenti a pagine e nome dei documenti utilizzati.

Come LLM è stato utilizzato GPT-4o-mini, che ha una context window di 128k che consente di sostenere prompt abbastanza lunghi. Come Embedder è stato utilizzato il *text-embedding-3-large* di OpenAI.

L'appendice A mostra l'utilizzo dei due plugin. Nello specifico, sono eseguite le seguenti operazioni:

1. Controllo dei gruppi in cui l'utente⁶ può entrare.
2. Creazione del gruppo "ING_SW". Ora è presente nella lista dei gruppi.
3. Cambio del gruppo attivo utilizzando il comando *@p*.
4. Inserimento dell'utente AleRavv all'interno del gruppo. Come si evince dalla chat, è possibile farlo in modo conversazionale. Il bot estrarrà le informazioni dal contesto, e chiederà all'utente di fornire quelle mancanti.
5. Caricamento del file *F2023.IS.L02.DesignPatterns.pdf*, contenente la spiegazione dei design pattern.
6. Domanda riguardo l'Abstract Factory Design Pattern e risposta del bot, con inseriti i riferimenti alle fonti utilizzate.

⁴il numero di caratteri che si devono sovrapporre tra chunks adiacenti

⁵https://api.python.langchain.com/en/latest/text_splitter/langchain_experimental.text_splitter.SemanticChunker.html

⁶Si utilizza l'utente Francesco, che dispone di tutti i permessi disponibili

In seguito, l'utente AleRavv ha effettuato la stessa domanda, dopo aver impostato "ING_SW" come gruppo attivo. Recuperando gli stessi documenti, la generazione termina con una risposta simile.

Inoltre, se AleRavv decidesse di cambiare gruppo attivo o disattivare quello corrente ed eseguisse la stessa domanda, non troverebbe quel documento in fase di recupero. È disponibile una versione della demo, accessibile all'url: <https://tiestarasp.duckdns.org/admin>. Attraverso l'invio di una mail all'indirizzo francescotesta2002@hotmail.it potrete ottenere l'accesso.

Capitolo 6

Conclusioni

In questo elaborato sono stati analizzati i metodi per verticalizzare un Large Language Model di partenza. Allo stato dell'arte, i due metodi analizzati e impiegati nelle aziende sono il fine-tuning e RAG. Queste due metodologie rappresentano due lati della stessa medaglia: attraverso il primo si cambia la conoscenza interna del Modello; attraverso il secondo si fornisce conoscenza esterna al Modello.

Analizzando pro e contro, con il fine-tuning si riescono a raggiungere performance simili a quelle ottenute con la RAG, pur utilizzando modelli con meno parametri. Tuttavia, la modifica di parametri implica la perdita di conoscenza interna in favore di quella settoriale. Questo comporta la perdita di generalità. Inoltre, il fine-tuning deve essere effettuato da esperti del settore, dunque i costi sono maggiori.

Con la RAG, attraverso l'adozione di alcune metodologie, è possibile mantenere la generalità. L'implementazione risulta più semplice, ma non è esente da contro. Infatti, un limite riscontrabile è la grandezza della context window del Modello. Se i dati da recuperare e inviare al LLM, attraverso la maggiorazione del prompt, sono elevati, sono necessari Modelli che possano esaudire la richiesta di token in input. Inoltre, le performance della rete sono strettamente legate al Modello e all'embedder utilizzato.

Infine, entrambe le metodologie non riescono ad eliminare completamente la presenza di allucinazioni.

Soltanto attraverso le Ontologie e i Knowledge Graphs si possono minimizzare le allucinazioni. Architetture e soluzioni per l'implementazione di quest'ultima metodologia sono ancora in fase di sviluppo.

Nell'elaborato si è dato molto spazio all'analisi del Cheshire Cat AI, un framework che si ispira al modello Wordpress per la realizzazione di chatbot verticali in un settore. La conoscenza esterna è fornita attraverso l'upload di documenti, e, allo stesso tempo, è possibile personalizzarne il comportamento attraverso l'implementazione di semplici plugin in Python. Poiché il framework si basa su RAG, ne eredita tutti i pro e i contro.

Dopo l'analisi dell'architettura e del funzionamento, é mostrato un esempio di implementazione di un plugin. Con poco piú di 100 righe di codice, è possibile variare completamente il comportamento del framework.

In futuro, saranno implementati nel framework la multimodalità¹ e l'uso dei Knowledge Graph per la riduzione delle allucinazioni.

¹La multimodalità é la capacità di processare input diversi dal semplice testo, come immagini o video.

Bibliografia

- [1] Accessed: 16/08/2024. URL: <https://cheshire-cat-ai.github.io/docs/plugins/hooks/>.
- [2] Accessed: 20/08/2024. URL: <https://python.langchain.com/v0.2/docs/concepts/#tools>.
- [3] OpenAI (2023). *GPT-4 Technical Report*. Rapp. tecn. OpenAI, 2023.
- [4] Dean Allemang e Juan Sequeda. *Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue!* 2024. arXiv: 2405.11706 [cs.AI]. URL: <https://arxiv.org/abs/2405.11706>.
- [5] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil e Hannaneh Hajishirzi. *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. Accessed: 13/07/2024. 2023. arXiv: 2310.11511 [cs.CL]. URL: <https://arxiv.org/abs/2310.11511>.
- [6] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. Accessed:10/09/2024. 2016. arXiv: 1409.0473 [cs.CL]. URL: <https://arxiv.org/abs/1409.0473>.
- [7] Teodoro Baldazzi, Luigi Bellomarini, Stefano Ceri, Andrea Colombo, Andrea Gentili e Emanuel Sallinger. *Fine-tuning Large Enterprise Language Models via Ontological Reasoning*. 2023. arXiv: 2306.10723 [cs.CL]. URL: <https://arxiv.org/abs/2306.10723>.
- [8] Matt Bornstein e Rajko Radovanovic. *Emerging Architectures for LLM Applications*. Accessed: 09/06/2024. 2024. URL: <https://a16z.com/emerging-architectures-for-llm-applications/>.
- [9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish,

- Alec Radford, Ilya Sutskever e Dario Amodei. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].
- [10] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman e Luke Zettlemoyer. *QLoRA: Efficient Finetuning of Quantized LLMs*, 36. 2023. arXiv: 2305.14314 [cs.LG].
- [11] AD Dongare, RR Kharde, Amit D Kachare et al. «Introduction to artificial neural network». In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012). Accessed: 29/08/2024, pp. 189–194.
- [12] Josep Ferrer. *How Transformers Work: A Detailed Exploration of Transformer Architecture*. Accessed:11/09/2024. URL: https://www.datacamp.com/tutorial/how-transformers-work?dc_referrer=https%5C%3A%5C%2F%5C%2Fwww.google.com%5C%2F.
- [13] Dr. Harsuminder Kaur Gill. *From RNNs to LLMs: A Journey through Sequential Modeling in NLP*. Accessed:05/09/2024. URL: <https://medium.com/@harsuminder/from-rnns-to-llms-a-journey-through-sequential-modeling-in-nlp-d42de5eb2cb9>.
- [14] Alvise Giubelli. *La self-attention delle reti transformer*. Accessed:13/09/2024. URL: <https://www.humai.it/la-self-attention-delle-reti-transformer/>.
- [15] *How Many Websites Use WordPress in 2024? WordPress Statistics*. Accessed:18/10/2024. URL: <https://www.wpzoom.com/blog/wordpress-statistics/#:~:text=WordPress%5C%20powers%5C%2043.5%5C%25%5C%20of%5C%20all,over%5C%2030%5C%2C000%5C%20WordPress%5C%20themes%5C%20available..>
- [16] Linmei Hu, Zeyi Liu, Ziwang Zhao, Lei Hou, Liqiang Nie e Juanzi Li. *A Survey of Knowledge Enhanced Pre-trained Language Models*. 2023. arXiv: 2211.05994 [cs.CL]. URL: <https://arxiv.org/abs/2211.05994>.
- [17] IBM. Accessed: 29/08/2024. URL: <https://www.ibm.com/it-it/topics/deep-learning>.
- [18] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang e Jong C. Park. *Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity*. Accessed: 14/07/2024. 2024. arXiv: 2403.14403 [cs.CL]. URL: <https://arxiv.org/abs/2403.14403>.
- [19] kili-technology.com. Accessed:29/08/2024. URL: <https://kili-technology.com/data-labeling/nlp/nlp-deep-learning>.

-
- [20] Giuseppe La Rocca. *Serving the long tail*. Accessed: 12/06/2024. EGI.eu, 2016. URL: <https://www.digitalinfrastructures.eu/content/serving-long-tail>.
- [21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel e Douwe Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Accessed: 11/07/2024. 2021. arXiv: 2005.11401 [cs.CL]. URL: <https://arxiv.org/abs/2005.11401>.
- [22] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng e Jun Zhou. «A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects». In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022). Accessed:29/08/2024, pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.
- [23] Marc Moreno Lopez e Jugal Kalita. *Deep Learning applied to NLP*. Accessed:05/09/2024. 2017. arXiv: 1703.03091 [cs.CL]. URL: <https://arxiv.org/abs/1703.03091>.
- [24] Javier Canales Luna. *What is Reinforcement Learning from Human Feedback?* Accessed: 26/07/2024. 2023. URL: <https://www.datacamp.com/blog/what-is-reinforcement-learning-from-human-feedback>.
- [25] Stephen M. Walker II. *What is supervised fine-tuning?* Accessed: 12/06/2024. URL: <https://klu.ai/glossary/supervised-fine-tuning>.
- [26] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow e Yanai Elazar. *Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation*. 2023. arXiv: 2305.16938 [cs.CL].
- [27] Armine Papikyan. *Fine-tuning large language models (LLMs) in 2024*. Accessed: 26/07/2024. 2024. URL: <https://www.superannotate.com/blog/llm-fine-tuning#:~:text=LLM%20fine%2Dtuning%20is%20a,its%20ability%20for%20specific%20tasks..>
- [28] *Point-wise Feedforward Network*. Accessed:18/09/2024. URL: <https://www.evisioning.io/vocab/point-wise-feedforward-network>.
- [29] Simone Raponi. Accessed:05/09/2024. URL: <https://www.redhotcyber.com/post/lstm-e-gru-miglioramenti-delle-recurrent-neural-networks/>.
- [30] Politecnico di Milano Redazione Osservatori Digital Innovation. Accessed:29/08/2024. URL: https://blog.osservatori.net/it_it/natural-language-processing-nlp-come-funziona-lelaborazione-del-linguaggio-naturale.

-
- [31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda e Thomas Scialom. *Toolformer: Language Models Can Teach Themselves to Use Tools*. 2023. arXiv: 2302.04761 [cs.CL]. URL: <https://arxiv.org/abs/2302.04761>.
- [32] Juan Sequeda, Dean Allemang e Bryon Jacob. *A Benchmark to Understand the Role of Knowledge Graphs on Large Language Model's Accuracy for Question Answering on Enterprise SQL Databases*. 2023. arXiv: 2311.07509 [cs.AI]. URL: <https://arxiv.org/abs/2311.07509>.
- [33] Mohammad Mustafa Taye. *Understanding Semantic Web and Ontologies: Theory and Applications*. 2010. arXiv: 1006.4567 [cs.AI]. URL: <https://arxiv.org/abs/1006.4567>.
- [34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin. *Attention Is All You Need*. Accessed: 10/09/2024. 2023. arXiv: 1706.03762 [cs.CL]. URL: <https://arxiv.org/abs/1706.03762>.
- [35] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu e Zhen-Hua Ling. *Corrective Retrieval Augmented Generation*. Accessed: 13/07/2024. 2024. arXiv: 2401.15884 [cs.CL]. URL: <https://arxiv.org/abs/2401.15884>.
- [36] Yu Yu, Chao-Han Huck Yang, Jari Kolehmainen, Prashanth G. Shivakumar, Yile Gu, Sungho Ryu Roger Ren, Qi Luo, Aditya Gourav, I-Fan Chen, Yi-Chieh Liu, Tuan Dinh, Ankur Gandhe Denis Filimonov, Shalini Ghosh, Andreas Stolcke, Ariya Rastow e Ivan Bulyko. «Low-Rank Adaptation of Large Language Model Rescoring for Parameter-Efficient Speech Recognition». In: *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, dic. 2023. DOI: 10.1109/asru57964.2023.10389632. URL: <http://dx.doi.org/10.1109/ASRU57964.2023.10389632>.

Appendice A

Test group

```
FRANCESCO: @l
BOT: Profile list available:
FRANCESCO: @c ING_SW
BOT: ING_SW created
FRANCESCO: @l
BOT: Profile list available:
    - ING_SW
FRANCESCO: @p ING_SW
BOT: Switched to profile 'ING_SW'
FRANCESCO: @p
BOT: Active profile is: ING_SW
FRANCESCO: insert user AleRavv in group
BOT: Info until now:
    {
      "user_id": "AleRavv",
      "group_name": "ING_SW"
    }

    → Confirm? Yes or no?
FRANCESCO: yes
BOT: Bot: User AleRavv inserted in group ING_SW
```

In seguito, sono state caricate le slide riguardanti lo studio dei Design Pattern utilizzate nel corso di Ingegneria del Software, tenuto dal professore Succi.

FRANCESCO: Can you explain me the Abstract Factory Design Pattern?

BOT: Of course! The Abstract Factory Design Pattern is a creational design pattern that provides an interface for creating families of related or dependent objects without specifying their concrete classes. This pattern is useful when you need to create various types of products that share a common theme or interface.

Key Points:

1. **Purpose:** The Abstract Factory allows you to create products that belong to a specific family without needing to know the details of their implementation.
2. **Usage:** It is particularly useful when you want to ensure that products created together are compatible with each other.
3. **Components:**
 - **Abstract Factory:** An interface that declares methods for creating abstract products.
 - **Concrete Factory:** Implements the abstract factory methods to create specific products.
 - **Abstract Product:** An interface for a type of product.
 - **Concrete Product:** Implements the abstract product interface

Example:

Imagine a furniture factory that produces chairs and sofas. You may have different families of furniture, such as Victorian and Modern. The Abstract Factory would allow you to create a set of Victorian or Modern chairs and sofas without knowing their specific classes.

Class Diagram:

AbstractFactory

createChair ()

createSofa ()

ConcreteFactory1

createChair () -> VictorianChair

createSofa () -> VictorianSofa

ConcreteFactory2

createChair () -> ModernChair

createSofa () -> ModernSofa

AbstractChair
ConcreteChair
ConcreteSofa

Implementation:

Here's a simple implementation in pseudo-code:

```
// Abstract Product
interface Chair {
    void sitOn();
}

interface Sofa {
    void lieOn();
}

// Concrete Products
class VictorianChair implements Chair {
    public void sitOn() {
        System.out.println("Sitting on a Victorian chair");
    }
}

class ModernChair implements Chair {
    public void sitOn() {
        System.out.println("Sitting on a modern chair");
    }
}

// Abstract Factory
interface FurnitureFactory {
    Chair createChair();
    Sofa createSofa();
}

// Concrete Factories
class VictorianFurnitureFactory implements FurnitureFactory {
    public Chair createChair() {
        return new VictorianChair();
    }

    public Sofa createSofa() {
        return new VictorianSofa();
    }
}

class ModernFurnitureFactory implements FurnitureFactory {
    public Chair createChair() {
```

```
        return new ModernChair ();
    }

    public Sofa createSofa () {
        return new ModernSofa ();
    }
}

// Client code
FurnitureFactory factory = new VictorianFurnitureFactory ();
Chair chair = factory.createChair ();
Sofa sofa = factory.createSofa ();
```

Advantages:

- Promotes consistency among products.
- Makes exchanging product families easy.
- Isolates the client code from concrete classes.

Disadvantages:

- Difficult to support new kinds of products in each family.

This information is based on the "Abstract Factory" section from Giancarlo Succi's material on Design Patterns (pages 25-27 in F2023.IS.L02.DesignPatterns.pdf). If you have any more questions or need further clarification, feel free to ask!

L'utente AleRavv può cambiare il gruppo attivo e ottenere la stessa risposta, nonostante non abbia caricato alcun file.

Ringraziamenti

Ringrazio il professore Giancarlo Succi per avermi offerto l'opportunità di approfondire argomenti di mio interesse.

Ringrazio di cuore i miei Genitori, senza i vostri sacrifici non avrei potuto raggiungere questo traguardo. Con tutti i vostri sforzi state cercando di offrire a me e ai miei fratelli una vita migliore e ci state riuscendo. Non esprimo facilmente i miei sentimenti e la mia gratitudine nei vostri confronti, ma questa è un'occasione giusta per farlo. Vi ringrazio profondamente e vi voglio tanto bene.

Ringrazio mia sorella Valentina e mio fratello Pasquale. Mi avete guidato nel prendere le decisioni giuste e mi avete sempre fatto sentire protetto. So di sembrare spesso troppo razionale, e a volte quasi apatico, ma sappiate che siete stati fondamentali in questi anni. La distanza mi ha permesso di apprezzare momenti che prima consideravo banali, perchè con voi assumono un significato molto più profondo.

Un ringraziamento speciale va a Cesco, che è come un fratello per me. La tua empatia, bontà, simpatia e disponibilità ti rendono l'amico ideale, quello che ognuno vorrebbe. Non si può racchiudere la nostra amicizia in poche righe, e penso nemmeno sia necessario. I piccoli gesti e la condivisione del quotidiano rendono il nostro legame così profondo.

Ringrazio Alessio, Angelo e Marco. Le chiamate con voi e il tempo passato insieme mi hanno fatto sentire sempre a casa. Con voi ho condiviso gran parte delle mie esperienze negli anni, e anche durante il percorso della laurea siete sempre stati un punto di riferimento per alleggerire i pensieri e farmi sentire meglio.

Un profondo ringraziamento va a Martina. Considerando come era partito il nostro rapporto, difficilmente ci si poteva aspettare un'amicizia così sincera. In questi anni entrambi abbiamo avuto momenti difficili, e ogni volta potevamo contare l'uno sull'altro. Mi hai risollevato spesso e ho sempre potuto contare su di te, specialmente nei momenti di solitudine che mi hanno colpito. Sono grato di averti come amica, e spero che il nostro rapporto non cambi mai.

Ringrazio Alessandro, Alessandro, Camilla, Daniele, Elisa, Giulia, Leonardo, Lorenzo, Mattia, Pietro, Saverio e Simone. Coi voi ho condiviso tante ore di studio e lo sviluppo di progetti previsti dal corso. Mi avete fornito tranquillità e spensieratezza anche nei momenti antecedenti gli esami.

Ringrazio tutti quelli che sono stati pilastri fondamentali per raggiungere questo traguardo, da chi è ora uscito dalla mia vita a chi ora ci sta entrando.