

**UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA**

Dipartimento di Ingegneria “Enzo Ferrari”

---

Corso di Laurea Triennale:  
**INGEGNERIA INFORMATICA**

**IMPLEMENTAZIONE DELLA RETRIEVAL  
AUGMENTED GENERATION E  
CONFIGURAZIONE DEL SINGLE SIGN-ON  
IN OPEN WEBUI**

***Candidato:***

Mattia Massarenti

***Relatore:***

Lorenzo Baraldi

***Correlatori:***

Davide Bucciarelli

Davide Caffagni

---

Anno Accademico 2024 - 2025

# ABSTRACT

---

In questa tesi affronteremo i concetti teorici che sono stati per me oggetto di studio per poter realizzare un plugin che fosse in grado di implementare qualcosa di molto simile ad una tecnica che oggi prende il nome di “Retrieval Augmented Generation”. Tramite questo metodo sviluppato recentemente, per gli LLM risulta possibile ottenere maggiori informazioni riguardo agli input sottomessi dall’utente, raccogliendo dati locali o dalla rete. In questa trattazione l’attenzione sarà rivolta principalmente a come funziona il processo di RAG in Internet. Ai fini di riuscire a carpire quali sono i temi caratterizzanti gli input dell’utente per poter eseguire ricerche più specifiche, si è rivelato necessario per me condurre studi su una classe di LLM avanzati, basati su transformer [1]. Per questa ragione, ho deciso di rendere più fluente la trattazione inserendovi un capitolo dedicato all’introduzione ai concetti fondamentali del deep learning.

Dato che il mio lavoro si è incentrato sulla realizzazione di un plugin per un’applicazione web che ho adattato per UniMoRe ed, in particolare, AIImage Lab, ho deciso di mostrare, nel secondo capitolo di questa tesi, come funzionano i single sign-on, in quanto mi sono occupato di configurare quello dell’Università all’interno del sito che ho modificato.

# RINGRAZIAMENTI

---

Ringrazio per prima la mia fidanzata, Valentina, per i saggi consigli che sempre dispensa, per avermi fatto crescere, per l'energia e l'affetto che mi dà ogni giorno, per essere la mia compagna di vita.

Ringrazio i miei genitori, Sonia e Gabriele, e tutta la mia famiglia per avermi dato fiducia ed avermi dato la possibilità di coltivare le mie passioni. Grazie anche a mio nonno Cesare, perché, anche se non ha fatto in tempo partecipare a questa mia esperienza, i suoi insegnamenti e la sua saggezza sono stati e saranno per sempre per me fondamentali per affrontare ogni aspetto della vita.

Ringrazio i miei amici per avermi sempre sostenuto e per avermi permesso di staccare la spina tra un esame e l'altro. Cito in particolare i miei migliori amici, Fabio e Giacomo, per avermi accompagnato ogni giorno della mia vita, per avermi fatto sorridere anche nei momenti più duri, per aver condiviso con me le loro riflessioni e per avermi ammonito quando necessario.

Ringrazio i miei compagni di corso per avermi dato allegria nei momenti difficili ed aiutato quando necessario. Vorrei tra loro citare in particolare Manuel e Leonardo per i bei momenti trascorsi insieme ogni giorno e per la loro pazienza nel rispondere sempre a tutte le mie domande.

Ringrazio il mio relatore, professor Baraldi, per avermi dato l'opportunità di mettere a frutto gli studi di questi tre anni e per avermi dedicato il suo tempo, nonostante i suoi numerosi impegni.

Ringrazio i correlatori, Davide B. e Davide C., per tutto il supporto che mi hanno fornito. In particolare ci tengo a ringraziare Davide B. per l'aiuto che mi ha dato sia come correlatore sia come amico.

# INDICE

---

<b>1</b>	<b>Introduzione</b>	<b>9</b>
<b>2</b>	<b>Single sign-on</b>	<b>10</b>
2.1	Contesto . . . . .	10
2.2	Principi di funzionamento . . . . .	10
2.3	Vantaggi e svantaggi . . . . .	11
2.4	OIDC . . . . .	12
<b>3</b>	<b>Cenni di deep learning</b>	<b>14</b>
3.1	Il neurone . . . . .	14
3.2	Reti neurali . . . . .	16
3.3	Backpropagation . . . . .	17
3.4	Deep learning . . . . .	18
3.5	Word embedding . . . . .	19
3.6	Attenzione . . . . .	19
3.7	Transformer . . . . .	20
<b>4</b>	<b>Retrieval augmented generation</b>	<b>22</b>
4.1	Panoramica . . . . .	22
4.2	RAG . . . . .	22
4.3	RAG on-line . . . . .	23
4.4	Prestazioni . . . . .	24
4.5	Vantaggi e svantaggi . . . . .	26
<b>5</b>	<b>Keyword extraction</b>	<b>28</b>
5.1	Contesto . . . . .	28
5.2	BERT . . . . .	28
5.3	Pre-trained transformer . . . . .	30
5.4	Fine-tuning . . . . .	31
5.5	KeyBERT . . . . .	32
<b>6</b>	<b>Conclusione</b>	<b>33</b>
<b>A</b>	<b>Appendice</b>	<b>36</b>
A.1	Slurm . . . . .	36
A.2	Ollama . . . . .	37
A.3	Avvio di Open WebUI . . . . .	37
A.4	Configurazione del SSO . . . . .	38

A.5 RAG in Open WebUI . . . . .	42
---------------------------------	----

# ELENCO DELLE FIGURE

---

2.1	OIDC - SSO sequence diagram . . . . .	12
3.1	Parallelismo tra i neuroni biologico ed artificiale . . . . .	14
3.2	Schema visivo del calcolo dell'output del neurone artificiale . . . . .	15
3.3	Deep neural network: input, hidden e output layer . . . . .	16
3.4	Schema minimale del meccanismo di attenzione . . . . .	20
3.5	Architettura del transformer . . . . .	21
4.1	Storico e previsioni sul valore di mercato della RAG . . . . .	23
4.2	Accuratezza di diverse configurazioni RAG . . . . .	25
5.1	Architettura di BERT (schema) . . . . .	29
5.2	Tre tipi di embedding usati da BERT . . . . .	30
A.1	Schermata iniziale di Open WebUI . . . . .	38
A.2	OIDC - Flusso delle HTTP request . . . . .	39
A.3	Schermata di login con SSO attivo . . . . .	40
A.4	Schermata di login AImage Lab . . . . .	40
A.5	Open WebUI homepage . . . . .	41
A.6	Sezione delle impostazioni relativa a Web Search. Il selettore è attivo	42
A.7	Attivazione della Web Search . . . . .	43
A.8	Esempio di chat con Web Search attiva . . . . .	43
A.9	Esempio di informazioni raccolte . . . . .	44
A.10	Schermata dell'editor . . . . .	46
A.11	Schermata dell'editor . . . . .	47
A.12	Pulsanti di import ed export per le Functions . . . . .	47
A.13	Esempio di query senza RAG su Wikiepdia . . . . .	48
A.14	Output della stessa query con RAG su Wikiepdia . . . . .	48

# LISTINGS

---

5.1	Istanziazione di KeyBERT . . . . .	32
A.1	Shell script per avviare una shell interattiva tramite Slurm . . . . .	37
A.2	Struttura di una Filter Function . . . . .	45

# LISTA DEGLI ACRONIMI

---

- **ANN**: Artificial Neural Network.
- **IA**: Intelligenza Artificiale.
- **API**: Application Programming Interface.
- **BERT**: Bidirectional Encoder Representation from Trasformers.
- **GLUE**: General Language Understanding Evaluation.
- **HPC**: High Performance Computing.
- **IdP**: Identity Provider.
- **JSON**: Javascript Object Notation.
- **LLM**: Large Language Model.
- **LM**: Language Model.
- **MLM**: Masked Language Modeling.
- **MSRA**: Microsoft Research Asia.
- **NN**: Neural Network.
- **PLM**: Permuted Language Modeling.
- **NLP**: Natural Language Processing.
- **OIDC**: Open ID Connect.
- **RP**: Relying Party.
- **SAML**: Security Assertion Markup Language.
- **SSH**: Secure Shell.
- **SQuAD**: Stanford Question Answering Dataset.
- **SSO**: Single Sign-On.
- **SWAG**: Situations With Adversarial Generations.
- **UI**: User Interface.
- **URL**: Uniform Resource Locator.

# Capitolo 1

## INTRODUZIONE

---

Nel corso dei seguenti capitoli di questa tesi approfondiremo i concetti teorici che stanno alla base del mio lavoro, che è stato possibile grazie al supporto di due software open source, Open WebUI ed Ollama. Mostreremo quanto questi ultimi siano strumenti estremamente potenti e, per concludere, osserveremo con un esperimento come, se combinati insieme, tali software consentano di esprimere il massimo potenziale dell'IA.

Open WebUI è una piattaforma progettata con lo scopo di permettere agli utenti di interfacciarsi con i LLM in modo semplice, intuitivo e completamente offline. Essa è stata creata affinché fosse facilmente estensibile, perciò è possibile aggiungere molti componenti senza mai modificare il codice sorgente. Quest'ultima è la principale ragione per cui Open WebUI è stata così preziosa per ottenere i risultati prodotti da questa tesi.

Tale software supporta vari runner di LLM, come Ollama e API compatibili con OpenAI, con un motore di inferenza integrato per RAG. Di quest'ultima parleremo nei prossimi capitoli e ne analizzeremo le differenze con quello da me implementato.

Ollama è uno strumento open source che esegue LLM direttamente su una macchina locale. Ciò lo rende particolarmente interessante per chiunque, sviluppatori, ricercatori o aziende, desideri avere controllo sui dati scambiati tra l'utente ed il modello e prestare attenzione alla privacy della comunicazione.

Nel nostro caso, tale software risulta essere di particolare interesse siccome esso si trova installato sul cluster AImageLab-SRV, server dedicato ad HPC, di proprietà di UniMoRe, mantenuto da AImageLab, collocato al Dipartimento di Ingegneria “Enzo Ferrari” a Modena. In questo modo è possibile per ricercatori e studenti, come me, effettuare elaborazioni ad alte prestazioni, che non sarebbero eseguibili in tempi ragionevoli su sistemi general purpose.

# Capitolo 2

## SINGLE SIGN-ON

---

### 2.1 Contesto

Come già accennato nell'introduzione, Open WebUI è un software open source che ha come principale scopo quello di offrire all'utente la possibilità di interagire con i LLM in modo semplice ed intuitivo, senza essere degli esperti del settore. Tuttavia, questo è solo uno dei vantaggi di tale software.

Open WebUI è stato progettato seguendo la moderna concezione di ingegneria del software. L'adozione di tale approccio è ben chiara a partire dalla documentazione, la quale si propone come guida per lo sviluppatore per istruirlo su come configurare Open WebUI affinché abbia le caratteristiche di cui si necessita, ma senza mai modificare il codice sorgente. Inoltre, vengono fornite anche istruzioni su come programmare dei plugins da “agganciare” al software, senza aver bisogno di cambiare nulla del codice originale. La caratteristica che ho qui descritto è formalizzata nella disciplina di ingegneria del software come “Open/Closed principle”.

In questo capitolo vedremo come l'adozione del principio appena descritto sia stato determinante per l'implementazione del Single Sign-On di UniMoRe all'interno di Open WebUI.

### 2.2 Principi di funzionamento

Iniziamo anzitutto dando la definizione di SSO (in italiano “autenticazione unica” o “identificazione unica”) e spiegando in che modo esso si distingua dalla tradizionale iscrizione ad un sito web.

*“Il single sign-on è la proprietà di un sistema di controllo d'accesso che consente ad un utente di effettuare un'unica autenticazione valida per più sistemi software o risorse informatiche alle quali è abilitato.”*

Wikipedia, Single sign-on, 2025

Questa è la definizione formale, che potremmo riformulare in questo modo: se due endpoint, ad esempio un sito web ed UniMoRe, implementano single sign-on tra loro compatibili, allora sarà automaticamente possibile accedere a tale sito per tutti gli utenti che hanno un account UniMoRe.

Il metodo tradizionale consiste nell'obbligare l'utente ad iscriversi al servizio web fornendo dati personali e scegliendo un'email ed una password da utilizzare per

autenticarsi al prossimo accesso. Tramite l'autenticazione unica invece l'utente verrà rediretto alla pagina di login della propria organizzazione, dove inserirà le credenziali relative all'account che ha presso di essa.

Diamo ora alcune definizioni utili per una trattazione più chiara dell'argomento. D'ora in avanti chiameremo:

- **End User:** l'utente che desidera accedere ad un determinato servizio web;
- **Relying Party:** il servizio web richiesto dall'utente e che implementa il SSO tramite una certa organizzazione;
- **Identity Provider:** l'organizzazione che mette a disposizione le API e i sistemi necessari per permettere le autenticazioni a servizi esterni tramite SSO.

Come abbiamo appena detto, le RP e gli IdP devono adottare API tra loro compatibili: infatti, esistono numerose implementazioni che differiscono per architettura, che può seguire diversi approcci (centralizzato, federato o cooperativo), e protocollo utilizzato, come ad esempio SAML, OAuth [2] od OpenID [3]. Alcuni processi di autenticazione unica sono proprietari, quindi per le RP che desiderano adottarli sarà necessario attenersi all'opportuna documentazione fornita dall'IdP competente; Sono tuttavia diffusi anche molti schemi SSO open source ben documentati, che molte RP, pubbliche o private, impiegano.

### 2.3 Vantaggi e svantaggi

L'implementazione di SSO avvantaggia sia gli utenti sia la RP che lo implementa.

Anzitutto, grazie all'autenticazione unica è possibile semplificare la gestione delle password ed aumentare la sicurezza: infatti, maggiore è il numero delle password da gestire, maggiore è la possibilità che ne vengano utilizzate di simili tra loro e facili da memorizzare. Ciò abbasserebbe il livello di sicurezza. Implementare SSO che permettano gli accessi tramite IdP come Google, Microsoft od università permette di abbassare sensibilmente il numero di password da memorizzare per il sito.

Un altro grande vantaggio è che sono semplificate la definizione e la gestione delle politiche di sicurezza in quanto esse vengono stabilite e controllate dalle organizzazioni che mettono a disposizione la possibilità di accedere a determinati servizi tramite il proprio metodo di autenticazione.

Infine, per gli utenti risulta vantaggioso poter accedere ai servizi online tramite SSO perché ciò consente loro di dover ricordare una sola coppia di credenziali.

Purtroppo l'impiego degli SSO da parte di RP implica il rischio che, se le credenziali che l'utente usa per accedere all'organizzazione IdP vengono compromesse, un malintenzionato ha automaticamente l'accesso a tutte o alla maggior parte delle RP a cui la vittima accede tramite quel SSO. Per questa ragione è sempre buona pratica per gli utenti scegliere una password lunga e complessa e ad archiviarla in modo sicuro, mentre si consiglia alle RP che implementano SSO di attivare l'autenticazione a due o più fattori.

## 2.4 OIDC

Open WebUI supporta nativamente la possibilità di autenticarsi tramite SSO con Google, Microsoft, Github ed OIDC [3]. Ho utilizzato quest'ultimo per implementare l'autenticazione unica UniMoRe. Esso è open source ed è un software costruito su OAuth 2.0 [2] al quale aggiunge un ID Token, ovvero un JWT [4], [5] firmato dall'IdP, che contiene informazioni sull'utente. In Open WebUI è possibile utilizzare configurare questo tipo di SSO per un qualsiasi IdP che usa OIDC. Se la RP lo implementa, allora anche l'IdP deve per forza usare lo stesso protocollo perché la RP richiede l'ID Token.

Di seguito è possibile osservare il sequence diagram che mostra in sintesi lo svolgimento del processo di autenticazione unica tramite il protocollo OIDC.

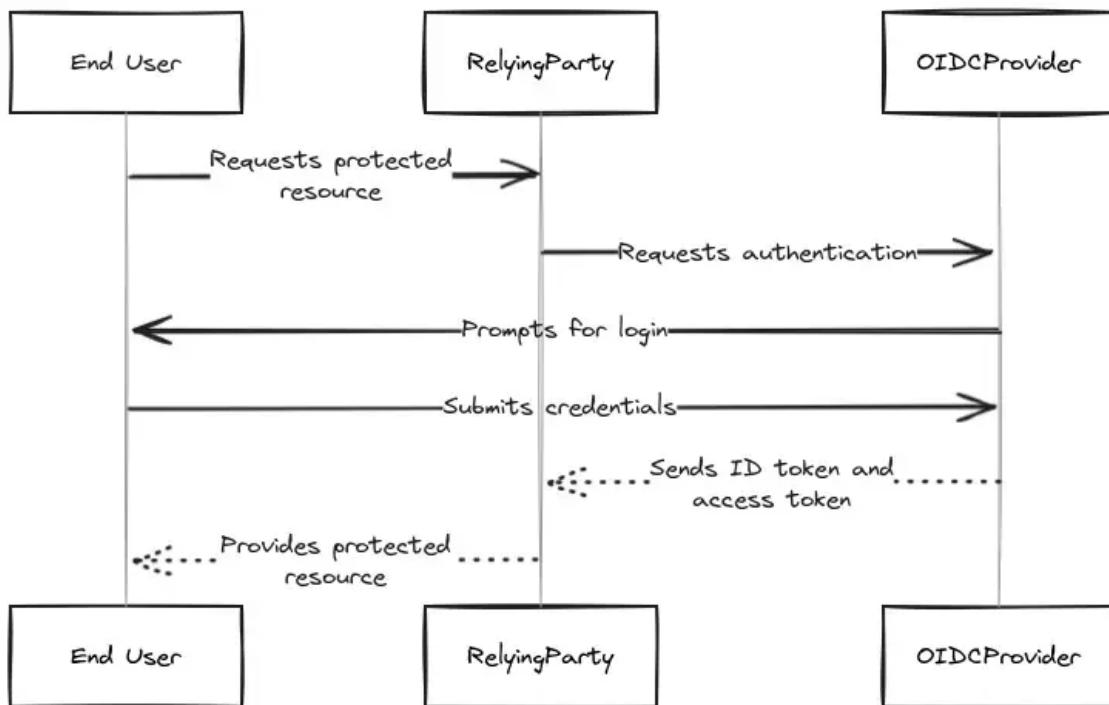


Figura 2.1: OIDC - SSO sequence diagram

OIDC Provider ricopre il ruolo di quello che finora abbiamo chiamato in modo più astratto IdP.

La seguente lista descrive i dettagli sulle azioni descritte dal diagramma:

1. Un utente accede all'end user UI della RP e seleziona il link per effettuare l'accesso tramite l'OIDC provider desiderato;
2. La RP prepara una richiesta di autorizzazione e la invia all'endpoint configurato per l'OIDC provider, con un ID client;
3. L'OIDC provider richiede l'autenticazione e il consenso dell'end user;
4. L'end user trasmette l'autenticazione e il consenso tramite le proprie credenziali;

5. L'endpoint del token dell'OIDC provider restituisce i token di accesso e ID.
6. La RP convalida il token e fornisce all'end user accesso alle risorse che l'utente aveva inizialmente richiesto.

Se il processo di login è terminato con successo, la RP può richiedere al OIDC provider informazioni sull'account dell'utente, usando lo stesso token che ha appena ricevuto.

# Capitolo 3

## CENNI DI DEEP LEARNING

---

In questo capitolo affronteremo alcuni dei concetti fondamentali del deep learning, che si riveleranno necessari per comprendere gli argomenti dei capitoli successivi.

### 3.1 Il neurone

Cominciamo dal definire che cosa sia un neurone artificiale:

*“Un neurone artificiale è una funzione matematica concepita come modello di un neurone biologico in una rete neurale. Il neurone artificiale è l’unità atomica di una rete neurale artificiale.”*

Wikipedia, Artificial neuron, 2025

Il design del neurone artificiale è stato ispirato dai circuiti neurali biologici. I suoi input sono analoghi ai potenziali postsinaptici eccitatori e inibitori a livello dei dendriti neurali. I suoi pesi sono i medesimi dei pesi sinaptici e il suo output è lo stesso del potenziale d’azione di un neurone, trasmesso lungo il suo assone. Quest’ultimo, detto anche “fibra nervosa”, è una lunga e sottile proiezione di una cellula nervosa nei vertebrati, che solitamente conduce impulsi elettrici, noti come potenziali d’azione, lontano dal corpo del neurone.

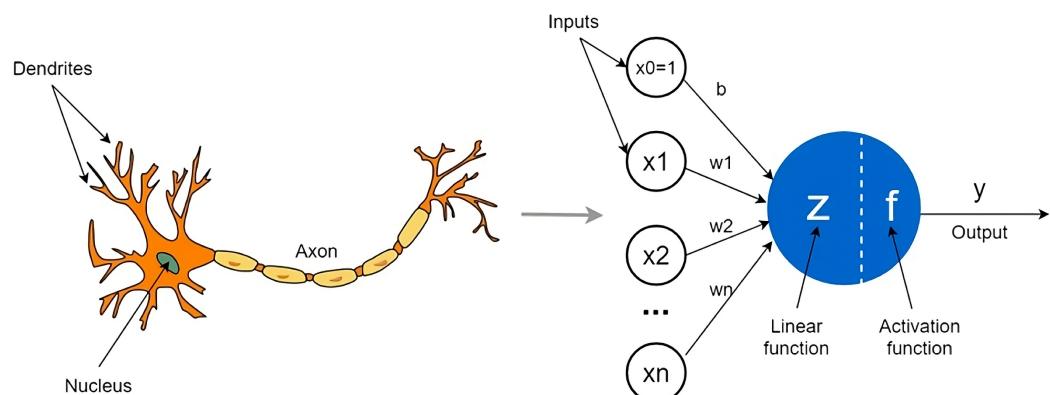


Figura 3.1: Parallelismo tra i neuroni biologico ed artificiale

Fonte: Ultralytics, 2025

Vediamo nel dettaglio come vengono processati gli input da parte di un neurone artificiale.

Ogni ingresso del neurone artificiale viene pesato separatamente e la somma degli input pesati, a cui spesso si aggiunge un termine, noto come “bias”, viene passata come parametro una funzione non lineare, chiamata “funzione di attivazione”. Come suggerito dal nome stesso, quest’ultima ha il compito di valutare se il neurone deve attivarsi o no. Per ottenere questo risultato è possibile sfruttare modelli matematici ideati appositamente per avere questo genere di comportamento.

Originariamente la funzione di attivazione era un gradino la cui soglia di attivazione era il bias: se quest’ultimo maggiorava la somma degli input pesati, allora il neurone rimaneva spento, viceversa si attivava. Oggi le funzioni non lineari adottate sono più complesse, dunque il bias serve per indurre uno spostamento, che consente di definire in modo più flessibile quando il neurone deve cominciare a reagire agli input.

Per un dato neurone artificiale  $k$  si hanno  $m + 1$  segnali d’ingresso  $x_0, \dots, x_m$ , ognuno pesato dal rispettivo valore  $w_0, \dots, w_m$ . Solitamente, all’input  $x_0$  viene sempre assegnato valore  $+1$ , il che lo rende il punto d’ingresso per il bias  $b_k = w_{k0}$ .

L’output del  $k$ -esimo neurone è dato da:

$$y_k = \varphi(v_k), \quad v_k = \sum_{j=0}^m w_{kj}x_j \quad (3.1)$$

Dove  $\varphi$  è la funzione di attivazione e  $v_k$  è la funzione lineare che esegue la somma pesata degli ingressi, mentre  $y_k$  è la funzione che associa il vettore di input  $\mathbf{x}$  al corrispondente output che si avrebbe dal  $k$ -esimo neurone.

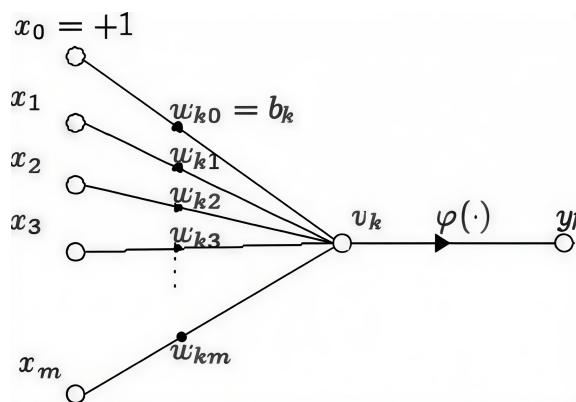


Figura 3.2: Schema visivo del calcolo dell’output del neurone artificiale

*Fonte: Wikipedia, Artificial neuron, 2025*

A seconda del loro compito, le funzioni di attivazione potrebbero avere forma sigmoide, utile ad esempio per la classificazione binaria, ma possono anche essere non lineari, lineari a tratti od a gradino. Sono inoltre spesso monotonamente crescenti, continue, differenziabili e limitate.

Capiremo nella seguente sezione come esprimere il grande potenziale dei neuroni artificiali, combinandoli in una struttura, che prende il nome di “rete neurale”.

### 3.2 Reti neurali

Adesso che sappiamo che cosa sia un neurone artificiale, prima di poter parlare di deep learning, ci serve ancora un ingrediente fondamentale: la rete neurale.

*“In machine learning, una rete neurale [6] (detta anche “artificial neural network”, ANN, o “neural net”, NN) è un modello computazionale ispirato alla struttura e alle funzioni delle reti neurali biologiche.”*

*“Una rete neurale è costituita da unità o nodi tra loro connessi, chiamati “neuroni artificiali”, che modellano lascivamente i neuroni nel cervello. [...] Questi sono collegati da connessioni che modellano le sinapsi presenti nel cervello umano.”*

Wikipedia, Neural network (machine learning), 2025

L’idea che sta alla base della rete neurale è quella di creare una struttura composta da neuroni artificiali interconnessi che comunicano tra loro. Ognuno di essi riceve segnali dai neuroni collegati in coda, li elabora ed invia un segnale a quelli connessi in testa. Gli “impulsi” inviati, così definiti per parallelismo con la controparte biologica, corrispondono ai numeri reali che vengono calcolati dalla funzione di attivazione. La potenza di ogni segnale su una certa connessione  $i$  viene determinata dal corrispondente peso  $w_i$ , che viene calibrato durante il processo di addestramento. L’output del neurone artificiale è analogo all’assone di quello biologico e il suo valore si propaga all’input dello strato successivo attraverso una sinapsi. Può anche uscire dal sistema, eventualmente come parte di un vettore di output.

Tipicamente, i neuroni vengono raggruppati in strati, “layer”. Ognuno di essi può effettuare trasformazioni diverse sui propri input. Il segnale viaggia a partire dal primo livello, detto “input layer”, passando per un numero indefinito di strati, chiamati “hidden layer”, ed infine uscendo dal cosiddetto “output layer”. Una rete neurale è detta “profonda”, “deep neural network”, se è composta da almeno due hidden layer. Un sistema di questo tipo è di “feedforward” se i neuroni sono organizzati in layer in cui le connessioni tra i nodi non formano mai cicli, altrimenti si dice “ricorrente”.

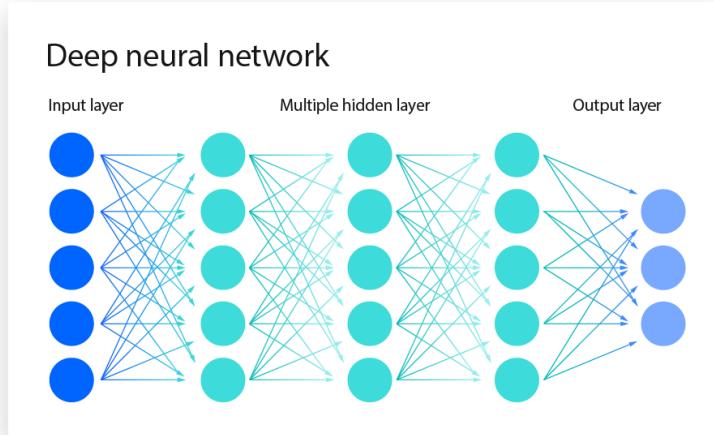


Figura 3.3: Deep neural network: input, hidden e output layer

Fonte: IBM, *Cos’è una rete neurale?*, 2025

Le reti neurali vengono solitamente addestrate attraverso una tecnica chiamata “minimizzazione del rischio empirico”. Questo metodo si basa sull’idea di ottimizzare i parametri della rete per minimizzare la differenza, detta “rischio empirico”, tra l’output previsto e i valori obiettivo effettivi in un dato dataset. Solitamente, per calibrare i parametri della rete vengono adottate tecniche basate sul gradiente, come la backpropagation [7]. Durante la fase di allenamento, le reti neurali artificiali apprendono dai dati di addestramento etichettati aggiornando iterativamente i loro parametri per minimizzare una funzione di perdita, “loss”, fissata. Questo metodo consente alla rete di generalizzare la propria formazione a dati non visti.

Le reti neurali artificiali oggi vengono utilizzate per eseguire compiti molto complessi, tra cui la modellazione predittiva, il controllo adattivo e la risoluzione di problemi di intelligenza artificiale. Possono imparare dall’esperienza e trarre conclusioni da un insieme di informazioni complesso e apparentemente non correlato.

### 3.3 Backpropagation

In machine learning, la backpropagation [7] è un metodo di calcolo del gradiente comunemente utilizzato per addestrare una rete neurale di tipo feedforward. Essa si può dire essere un’applicazione efficiente della regola della catena alle reti neurali.

Intuitivamente, la regola della catena afferma che, se si conosce il tasso di variazione istantaneo di un parametro  $z$  rispetto ad  $y$  e quello di  $y$  in funzione di  $x$ , allora è possibile calcolare il tasso di variazione istantaneo di  $z$  rispetto a  $x$  come prodotto dei due tassi di variazione:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \quad (3.2)$$

La backpropagation calcola il gradiente della funzione di perdita rispetto ai pesi della rete per un singolo esempio di input-output. Questa operazione viene eseguita in modo efficiente, valutando il gradiente uno strato alla volta, iterando all’indietro dall’ultimo strato per evitare calcoli ridondanti di termini intermedi nella regola della catena.

Siano dati:

- $\mathbf{x}$ : vettore di input;
- $\mathbf{y}$ : vettore di output obiettivo;
- $C$ : funzione di perdita, detta anche “funzione di costo”. Nei compiti di classificazione, essa è nota come “cross-entropy”;
- $L$ : numero di strati;
- $W^l$ : matrice dei pesi tra il layer  $l - 1$  e  $l$ , dove  $w_{jk}^l$  è il peso tra il  $k$ -esimo nodo dello strato  $l - 1$  e il  $j$ -esimo nodo del livello  $l$ ;
- $f^l$ : funzione di attivazione per il layer  $l$ ;
- $a_j^l$ : impulso che si ha all’attivazione (output) del  $j$ -esimo nodo del layer  $l$ .

La rete nel suo complesso è una combinazione di composizione di funzioni e moltiplicazione tra matrici:

$$g(x) := f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots)) \quad (3.3)$$

Per un set di addestramento ci sarà un set di coppie input-output,  $\{(x_i, y_i)\}$ . Per ognuna di queste paia, la loss del modello calcolata su quei due valori sarà pari alla differenza tra l'output predetto,  $g(x_i)$ , e quello obiettivo,  $y_i$ :

$$C(y_i, g(x_i)) \quad (3.4)$$

Si noti la distinzione: durante la valutazione del modello, i pesi sono fissi mentre gli input variano, l'output target potrebbe essere sconosciuto; la rete termina con lo strato di output, che non include il calcolo della funzione di perdita. Durante l'addestramento del modello, la coppia input-output è fissa, mentre i pesi variano; la rete termina stimando la funzione di perdita.

Il metodo della backpropagation prevede il calcolo del gradiente della loss, dopo averla valutata per una coppia di input fissata  $(x_i, y_i)$ , dove i pesi  $w_{jk}^l$  sono variabili. Ogni componente  $\frac{\partial C}{\partial w_{jk}^l}$  del gradiente può essere calcolato tramite la regola della catena, ma stimarli singolarmente è inefficiente. Per evitare di eseguire operazioni duplicate e di calcolare valori intermedi non necessari, conviene valutare il gradiente di ogni strato in ordine dall'ultimo al primo (per questo la tecnica è detta “retropropagazione”, in particolare quello dell’input ponderato di ogni livello, indicato con  $\delta^l$ .

Informalmente, possiamo spiegare più liberamente a parole che, poiché l’unico modo in cui un peso nella matrice  $W^l$  influenza linearmente la perdita è attraverso il suo effetto sullo strato successivo, i  $\delta^l$  sono gli unici dati necessari per calcolare i gradienti dei pesi allo strato  $l$ ; per questo motivo, i gradienti dei pesi dello strato precedente possono essere calcolati da  $\delta^{l-1}$  e ripetuti ricorsivamente.

Questa tecnica evita inefficienza in due modi: per prima cosa, evita la duplicazione perché, quando si calcola il gradiente allo strato  $l$ , non è necessario ricalcolare tutte le derivate sugli strati successivi  $l+1, l+2, \dots$  ogni volta; in secondo luogo, la backpropagation evita calcoli intermedi non necessari, perché in ogni fase stima direttamente il gradiente dei pesi rispetto all’output finale, che in addestramento ricordiamo corrispondere alla funzione di perdita, anziché calcolare inutilmente le derivate dei valori degli hidden layer rispetto alle variazioni dei pesi  $\frac{\partial a_{j'}^{l'}}{\partial w_{jk}^l}$ .

### 3.4 Deep learning

Ora abbiamo tutte le nozioni necessarie per definire formalmente che cosa sia il deep learning:

*“Nel machine learning, il deep learning si concentra sull’impiego di reti neurali multistrato per eseguire attività quali classificazione, regressione e apprendimento della rappresentazione. [...] L’aggettivo “deep”, “profondo”, si riferisce all’uso di più livelli (da tre a diverse centinaia o migliaia) nella rete.”*

Fondamentalmente, il deep learning si riferisce a una classe di algoritmi di machine learning, in cui si usa una gerarchia di livelli per trasformare i dati di input in una rappresentazione progressivamente più astratta e composita.

Ad esempio, in un modello di riconoscimento delle immagini, l'input grezzo può essere un'immagine, rappresentata come un tensore di pixel. Il primo livello rappresentativo può tentare di identificare forme di base come linee e cerchi, il secondo può comporre e codificare la disposizione dei bordi, il terzo può codificare un naso e degli occhi ed infine il quarto può riconoscere che l'immagine contiene un volto.

I metodi di apprendimento usati in questo campo sono di tipo supervisionato, semi-supervisionato o non supervisionato.

La maggior parte dei modelli moderni di deep learning si basa su reti neurali multistrato, come le NN convoluzionali ed i transformer [1]. Questi ultimi sono particolarmente interessanti perché sono il motore alla base dei sistemi che ho usato per lavorare ad Open WebUI. Prima di poterne dare una spiegazione adeguata, è necessario introdurre i concetti di word embedding ed attenzione, che vedremo nelle prossime sezioni.

### 3.5 Word embedding

Come di consueto, diamo una definizione formale del concetto:

*““word embedding” è un termine complessivo che indica, nell’elaborazione del linguaggio naturale, un insieme di tecniche di modellazione in cui parole o frasi di un vocabolario vengono mappate in vettori di numeri reali.”*

Wikipedia, Word embedding, 2025

L'embedding viene utilizzato nell'analisi del testo. In genere, ogni parola viene rappresentata tramite un vettore a valori reali, che codifica il significato del termine in modo tale che i vocaboli più vicini nello spazio vettoriale siano presumibilmente simili nel significato. I word embedding possono essere ottenuti utilizzando tecniche di modellazione del linguaggio e di apprendimento delle caratteristiche, in cui parole o frasi del vocabolario vengono mappate su vettori di numeri reali.

I metodi per generare questa mappatura sono numerosi; essi comprendono le reti neurali ed i modelli probabilistici. Molte delle nuove tecniche di word embedding sono realizzate con l'architettura delle reti neurali, piuttosto che con le più tradizionali tecniche di apprendimento supervisionato. I thought vector sono un'estensione delle word embedding per intere frasi o anche documenti.

### 3.6 Attenzione

*“In machine learning, l’attenzione è un metodo che determina l’importanza di ogni componente in una sequenza rispetto agli altri componenti nella stessa successione.”*

Wikipedia, Attention (deep learning), 2025

In passato era stato implementato il meccanismo di attenzione in un sistema di traduzione linguistica basato su di una rete neurale ricorrente seriale, ma un recente

paradigma, il transformer [1], ha superato le prestazioni della RNN sequenziale, basandosi maggiormente sullo schema di attenzione parallela.

Nel 2017 fu pubblicato un articolo scientifico, “Attention Is All You Need” [1], che divenne pietra miliare del machine learning, in cui fu introdotta una nuova architettura, detta “transformer”, basata sul meccanismo di attenzione introdotto in una pubblicazione del 2014 [8].

In “Attention Is All You Need” si parla di “dot-product attention” e “self-attention”. Si definisce in formule la dot-product attention come:

$$\text{Attention}(Q, K, V) := \text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) \times V \quad (3.5)$$

Dove  $Q, K, V$  sono rispettivamente le matrici query, key e value e  $d_k$  è la dimensione dei valori. La funzione softmax, o “funzione esponenziale normalizzata”, è una generalizzazione di una funzione logistica che mappa un vettore  $K$ -dimensionale  $\mathbf{z}$  di valori reali arbitrari in un altro vettore  $K$ -dimensionale  $\sigma(\mathbf{z})$  di valori compresi in un intervallo  $(0, 1)$  la cui somma è 1.

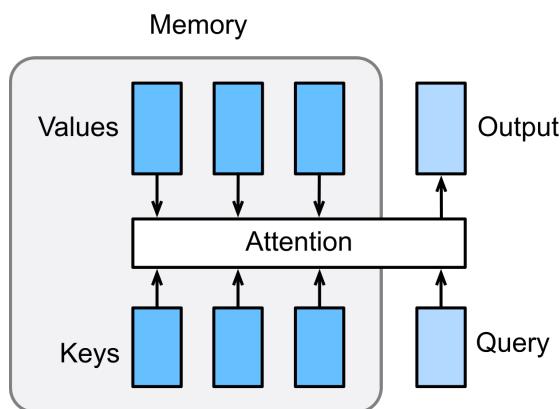


Figura 3.4: Schema minimale del meccanismo di attenzione

Fonte: Wikipedia, *Attention (deep learning)*, 2025

Poiché il modello si basa su matrici provenienti dalla stessa sorgente (ovvero la sequenza di input o la finestra di contesto), si elimina la necessità di reti neurali ricorrenti, garantendo la completa parallelizzabilità dell’architettura. Ciò differisce dalla forma originale del meccanismo di attenzione introdotto nell’articolo del 2014 [8].

### 3.7 Transformer

Leggiamo insieme infine la definizione di transformer [1]:

*“In deep learning, il transformer è un’architettura di rete neurale basata sul meccanismo di attenzione multi-head, in cui il testo viene convertito in rappresentazioni numeriche chiamate “token”, dove ognuno di essi viene convertito in un vettore tramite ricerca da una tabella di word embedding.”*

Wikipedia, Transformer (deep learning), 2025

In sintesi, con parole più semplici e senza addentrarci troppo nei dettagli, possiamo dire che un trasformer sia una deep NN, che è basata sul meccanismo di attenzione introdotto dai ricercatori di Google nel 2017 [1] e sul word embedding. A ogni livello della rete, ogni token viene contestualizzato nell'ambito della finestra di contesto con altri token (non mascherati) tramite un meccanismo di attenzione multi-head parallelo, consentendo di amplificare il segnale per i token chiave e di ridurlo per quelli meno importanti.

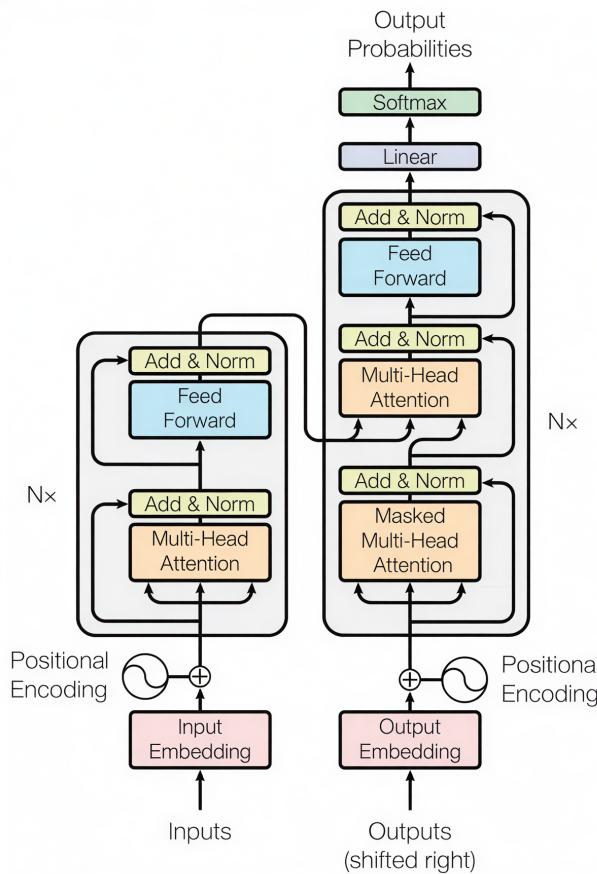


Figura 3.5: Architettura del transformer

*Fonte: Attention Is All You Need [1], 2017*

# Capitolo 4

## RETRIEVAL AUGMENTED GENERATION

---

### 4.1 Panoramica

In questo capitolo andremo ad osservare come le performance di un LLM [9] possano sensibilmente migliorare se gli viene fornita la possibilità di effettuare ricerche su Internet o, nel nostro caso specifico, su Wikipedia.

Così come per l'implementazione del SSO, anche per portare a termine questo esperimento l'oculata progettazione di Open WebUI è stata determinante. Infatti, tale software predispone nativamente di alcuni meccanismi, ognuno avendo un preciso utilizzo, per intercettare gli input dell'utente e consentire di rielaborarli prima che vengano inviati al LLM. Ciò mi ha permesso di focalizzare l'attenzione sullo sviluppo della RAG [10].

### 4.2 RAG

Come di consueto, prima di cominciare a parlare di che cosa sia la RAG e come l'ho implementata in questo progetto, diamone una definizione formale.

*“La Retrieval Augmented Generation (abbreviato RAG, in italiano: generazione potenziata da ricerca) è una tecnica che si propone di migliorare i risultati di uno strumento di intelligenza artificiale generativa tramite la ricerca in tempo reale di dati che si vanno ad aggiungere all'input utente. [...] Il concetto alla base della RAG è un elemento persistente nella storia dei sistemi di generazione automatica delle risposte, a partire quantomeno dai primi anni '70, attraverso diverse generazioni di modelli di generazione del testo. L'espressione Retrieval Augmented Generation è stata coniata dai ricercatori di Meta (al tempo chiamata Facebook) nel 2020, in un contesto di crescente interesse per i modelli linguistici di grandi dimensioni, decisamente superiori ai modelli precedenti.”*

Wikipedia, Retrieval augmented generation, 2025

Dal punto di vista implementativo, realizzare un meccanismo di RAG significa creare uno script che, a partire dall'input dell'utente, sia in grado di effettuare ricerche in tempo reale su documenti locali o da fonti Internet, raccoglierne i risultati e giustapporli, spesso rielaborandoli, all'input dell'utente, per poi spedire tutte queste informazioni al LLM come un unico messaggio.

I modelli di AI generativi testuali [11] sono capaci di sostenere una conversazione in base alle sole informazioni che sono state fornite loro durante l'addestramento, il che significa che qualsiasi cognizione posteriore alla loro formazione od estranea ai dati di allenamento è sconosciuta al modello. La RAG ha assunto grande valore perché permette ai LLM di ricevere un contesto contemporaneo all'input dell'utente, consentendo loro di elaborare una risposta incrociando le conoscenze provenienti dal proprio addestramento con le informazioni aggiornate appena ricevute.

Come chiarito dalla definizione, il termine RAG [10] è molto recente, risale a non più di cinque anni fa. Nonostante ciò, questo approccio sta assumendo crescente importanza sul mercato. Per darne un'idea, ho riportato qui uno dei numerosi grafici reperibili in rete che fornisce una panoramica sul valore economico attribuito all'approccio RAG a partire dall'anno in cui fu coniato il termine, facendo anche una predizione, condivisa da alcune fonti, sulla sua crescita nei prossimi cinque anni.

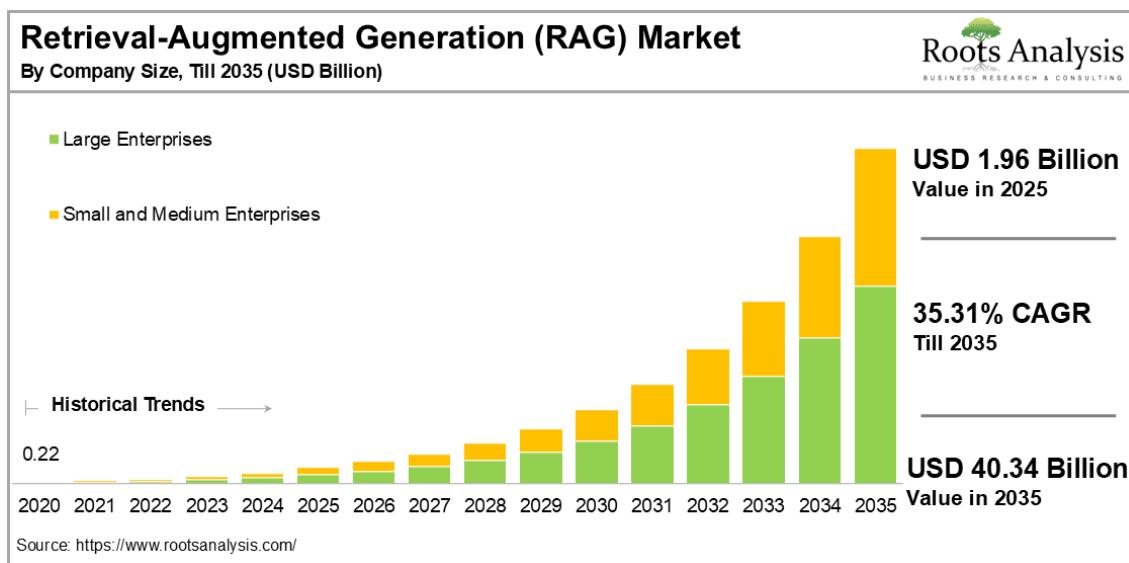


Figura 4.1: Storico e previsioni sul valore di mercato della RAG

Fonte: Roots Analysis, *Retrieval-Augmented Generation (RAG) Market*

### 4.3 RAG on-line

Si possono distinguere due macro categorie di RAG: off-line ed on-line. La prima consiste nel programmare un meccanismo per la raccolta di dati da documenti reperibili localmente, solitamente posti in una sezione dedicata. Open WebUI mette a disposizione uno spazio con questo scopo, in cui è possibile caricare ed organizzare informazioni che l'utente desidera rendere note al LLM [9] per la generazione di risposte più pertinenti alle proprie esigenze. La RAG on-line, invece, si tratta di un'operazione simile, ma i documenti raccolti provengono dalla rete. Siccome non sempre i risultati delle ricerche su Internet sono pertinenti alla reale domanda dell'utente, bisognerà adottare opportune strategie per filtrare i dati, trattenendo solo quelli più affini alla richiesta.

In questa trattazione ci soffermeremo sul processo completo di RAG on-line, essendo quest'ultimo quello che ho implementato in Open WebUI. Descriviamo in

modo più dettagliato la generazione potenziata da ricerca in rete scomponendola in una sequenza di cinque operazioni fondamentali:

1. **RAG statica**: si tratta della fase in cui lo script cerca su Internet informazioni pertinenti all'input (“query”) dell’utente su sua esplicita richiesta, espressa in genere tramite l’attivazione di un pulsante dedicato. Una pratica comune è quella di estrarre dalla query i concetti più significativi, come ad esempio parole chiave, per poter effettuare ricerche su Internet più accurate. Abbiamo definito questa tecnica “statica” perché ne esiste una variante dinamica più avanzata. In base alle interazioni dell’utente, la RAG dinamica è in grado di determinare autonomamente se sia necessario o meno attivare i processi successivi. Esistono sistemi particolarmente flessibili capaci di attivare il retrieval durante la generazione stessa della risposta; in tal caso la generazione viene messa in pausa per la durata dei processi di retrieval e aumento del prompt e, a seconda di dove porta il testo generato dal LLM, il retrieval può essere attivato più o meno volte;
2. **Retrieval**: in questa fase lo script va a ricercare nel corpus disponibile le informazioni di interesse alle questioni sollevate dall’utente. Oltre ai classici algoritmi di ricerca, sono state sviluppate ulteriori tecniche basate su IA per migliorare i risultati dell’indagine. Tra queste, citiamo tecniche avanzate come la “query rewriting”, famiglia di tecniche che affidano a un modello di linguaggio la stesura della query per la ricerca nel database, e il “re-ranking”, che utilizza un modello di linguaggio di piccole dimensioni per vagliare un grande numero di testi recuperati, scartare i meno rilevanti, e passare solo i restanti alla fase seguente;
3. **Aumento del prompt**: i dati ottenuti in fase di retrieval vengono aggiunti al contesto del modello di linguaggio prima o, nel caso di RAG dinamica, durante la generazione della risposta visibile all’utente. In ogni caso, le informazioni ottenute tramite retrieval, la domanda dell’utente, e gli eventuali messaggi precedenti nella conversazione, devono sempre essere aggregati tramite prompt engineering per ottenere il risultato finale più completo possibile;
4. **Generazione della risposta**: dopo aver ottenuto in input il contesto elaborato secondo le norme della terza fase, il LLM genera la risposta visibile all’utente;
5. **Rivelazione delle fonti**: idealmente, vengono mostrati all’utente i riferimenti ai documenti estratti tramite retrieval, in modo tale da rendergli più facile la verifica dell’attendibilità della risposta generata e favorirgli un approfondimento, se lo desidera.

#### 4.4 Prestazioni

In generale, tanto più la raccolta dati della RAG [10] è accurata, tanto più la risposta del LLM [9] sottostante sarà più probabilmente pertinente. Siccome la qualità dell’elaborazione effettuata dal modello di AI dipende molto dal suo addestramento e dai propri parametri, conviene incentrare l’attenzione sull’accuratezza del sistema RAG.

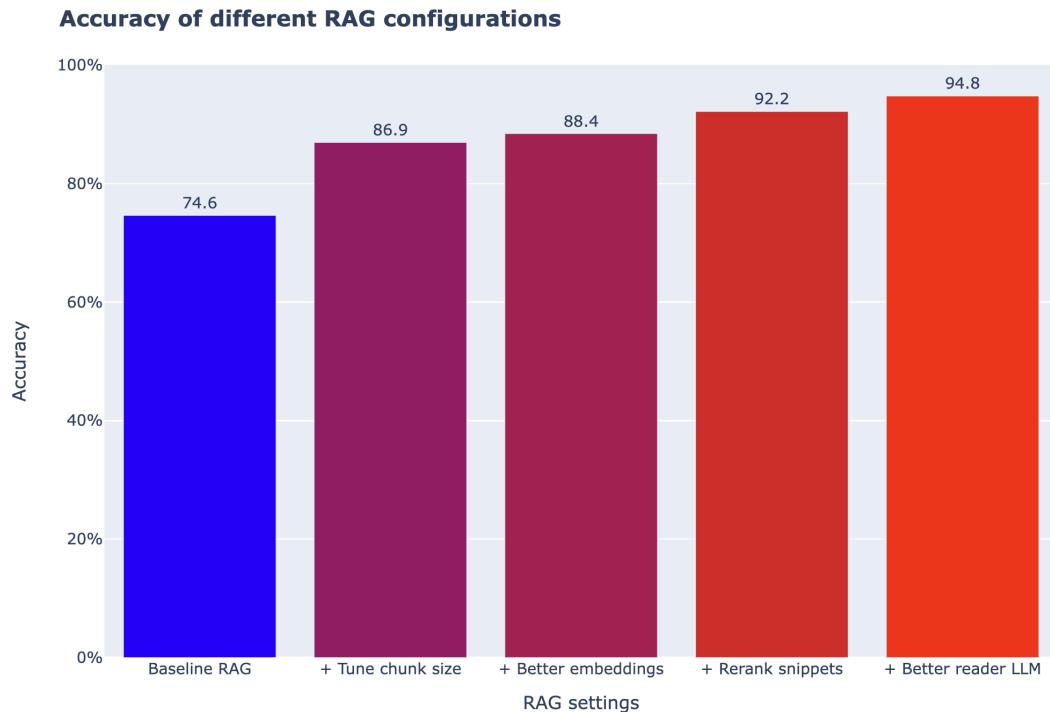


Figura 4.2: Accuratezza di diverse configurazioni RAG

*Fonte: HuggingFace, RAG Evaluation*

Questo grafico è significativo per noi non per il valore dell'accuracy, che dipende dall'implementazione della baseline RAG, ma per osservare le differenze di performance che si hanno tra le diverse configurazioni. Da tenere presente, per la corretta interpretazione del grafico, è che ogni colonna mostra il valore di accuratezza della configurazione precedente aggiungendole la caratteristica descritta dopo il segno +.

Chiariamo in breve il significato di ogni sistema di raccolta dati:

- **Baseline RAG:** implementazione minimale che si occupa unicamente di raccogliere i dati inerenti alla query dell'utente, organizzarli in sequenza e spedirli al LLM, mettendoli in coda all'input umano;
- **Tune chunk size:** imposta una dimensione minima e una massima per i “chunk”. Un chunk è una porzione di testo di dimensioni variabili che ha lo scopo di facilitare l’elaborazione del contenuto da parte del LLM. Ognuno di questi blocchi è composto da “token”, ovvero parole o caratteri, a scelta sulla base dello specifico caso. La pratica più comune è quella di impostare un limite inferiore di 128 token e uno superiore di 512, tuttavia l’argomento è controverso;
- **Better embeddings:** gli “embedder” sono modelli multimodali che convertono il testo o qualsiasi tipo di input in vettori. Gli “embeddings” sono gli output di tali sistemi e la loro qualità può essere migliorata in molti modi. Un esempio è il sistema a doppio livello, che consente di fare due volte la ricerca del contenuto: al primo passo si divide il documento in macro chunk, al secondo si decompone ognuna di queste parti in maniera semantica;

- **Rerank snippets:** il rerank consiste nella raccolta dei primi  $n$  vettori tra quelli creati in fase di embedding, assegnare ad ognuno di essi un punteggio di pertinenza tramite un modello più lento e accurato e, infine, selezionando i migliori  $k$  da inviare al LLM;
- **Better reader LLM:** questa caratteristica non è sotto il controllo di chi implementa la RAG, ma di chi addestra il LLM. Oggi la ricerca è molto attiva per realizzare modelli con elevate capacità di comprensione del testo per cooperare meglio con il supporto offerto da sistemi di raccolta dati, come quello che stiamo discutendo in questo capitolo.

Per concludere, possiamo dedurre facilmente osservando il grafico che i maggiori incrementi di prestazioni vengono percepiti con l'introduzione del dimensionamento dei chunk da consegnare al LLM e tramite il reranking degli embeddings. Siccome configurazioni sofisticate rallentano il processo di generazione della risposta visibile all'utente, è opportuno stabilire un rapporto tra costo computazionale e prestazioni soddisfacente per il proprio caso d'uso tenendo presente questo diagramma.

#### 4.5 Vantaggi e svantaggi

In parte abbiamo già detto quelli che sono i benefici fondamentali che indussero l'impiego della RAG [10], ma vediamone ora un riepilogo più dettagliato:

- **Fornire risposte aggiornate e accurate:** l'approccio RAG garantisce che la risposta di un LLM [9] non si basi esclusivamente su dati e parametri di addestramento statici e non aggiornati. Il modello utilizza invece sorgenti di dati esterne e aggiornate per fornire risposte.
- **Ridurre le risposte imprecise o le allucinazioni:** basando l'output del LLM su conoscenze esterne rilevanti, la RAG cerca di ridurre il rischio di risposte con informazioni errate o inventate (note anche come "allucinazioni"). I risultati possono includere citazioni delle fonti originali, consentendo una verifica umana.
- **Fornire risposte pertinenti:** utilizzando la RAG, il LLM sarà in grado di fornire risposte contestualmente pertinenti e adeguate ai dati proprietari o settoriali specifici di un'organizzazione.
- **Convenienza e longevità:** rispetto ad altri approcci utilizzati per personalizzare i LLM con dati specifici del settore, la RAG è semplice ed economica. Le organizzazioni possono implementare l'approccio RAG senza dover apportare modifiche al modello. Questo è particolarmente vantaggioso quando i modelli devono essere aggiornati frequentemente con nuovi dati, evitando di doverli addestrare nuovamente e rendendoli più longevi.

Invece, il rischio in cui si incorre è uno solo, ma determinante per un'implementazione di successo. Il pericolo di trasformare tutti i vantaggi della RAG in un solo grande svantaggio è quello di inviare al LLM informazioni potenzialmente fuorvianti, che potrebbero portarla parzialmente o completamente fuori strada rispetto alla query iniziale, ottenendo l'esatto opposto dei benefici per cui la RAG è stata ideata.

Oggi sono state sviluppate molte tecniche che, in una buona percentuale dei casi, permettono di evitare che la RAG si ritorca contro sé stessa tramite la manipolazione e il filtraggio dei dati raccolti tramite sofisticati modelli di machine learning, talvolta combinandone multipli contemporaneamente.

Vedremo nei prossimi due capitoli quali sono i supporti che ho adottato per fare in modo che il mio script per la RAG riducesse gli errori di questo genere, nei limiti delle mie possibilità e conoscenze.

# Capitolo 5

## KEYWORD EXTRACTION

---

### 5.1 Contesto

Come sappiamo, per effettuare ricerche su Internet o siti web, spesso l'approccio più efficace per ottenere buoni risultati è riuscire ad esprimere la richiesta tramite poche parole significative. Per questa ragione, serve trovare un metodo per estrarre dalla query dell'utente pochi vocaboli chiave che possano riassumere i concetti caratterizzanti del discorso, siccome l'input da utilizzare per effettuare RAG potrebbe essere composto da frasi multiple di complessità variabile.

Oggi esistono molti metodi più o meno complessi, coerentemente al grado di dettaglio desiderato, per consentire ai calcolatori di portare a termine compiti di questo genere. Per la mia implementazione di RAG ho deciso di adottare la libreria Python “KeyBERT”. Essa si basa sul modello di embedding di un famoso LM, sviluppato da Google AI, di nome “BERT” [12], che vedremo nella prossima sezione.

### 5.2 BERT

BERT [12] è un modello pubblicato dai ricercatori di Google per la prima volta nel 2018. Esso, tramite il self-supervised learning, è in grado di imparare a rappresentare il testo come una sequenza di vettori. Esso migliorò sensibilmente lo stato dell'arte degli LM e, a partire dal 2020, esso divenne la base di partenza preponderante per effettuare esperimenti nel campo del NLP.

BERT ha un'architettura transformer [1] di tipo “encoder-only”, che consiste di quattro moduli:

- **Tokenizer**: questo modulo converte un frammento di testo inglese in una sequenza di interi, detti “token”);
- **Embedding**: in questa fase BERT trasforma la sequenza di token in un array di vettori a valori reali. La struttura dati ottenuta rappresenta la conversione di token discreti in uno spazio euclideo a dimensione inferiore;
- **Encoder**: esso consiste in una pila di blocchi transformer con auto-attenzione, ma senza data masking;
- **Task head**: questo modulo converte nuovamente i vettori di rappresentazione finali in token codificati one-hot, producendo una distribuzione di probabilità prevista sui tipi di token. Può essere visto come un semplice decodificatore, che

decifra la rappresentazione latente in tipi di token, o come un “un-embedding layer”.

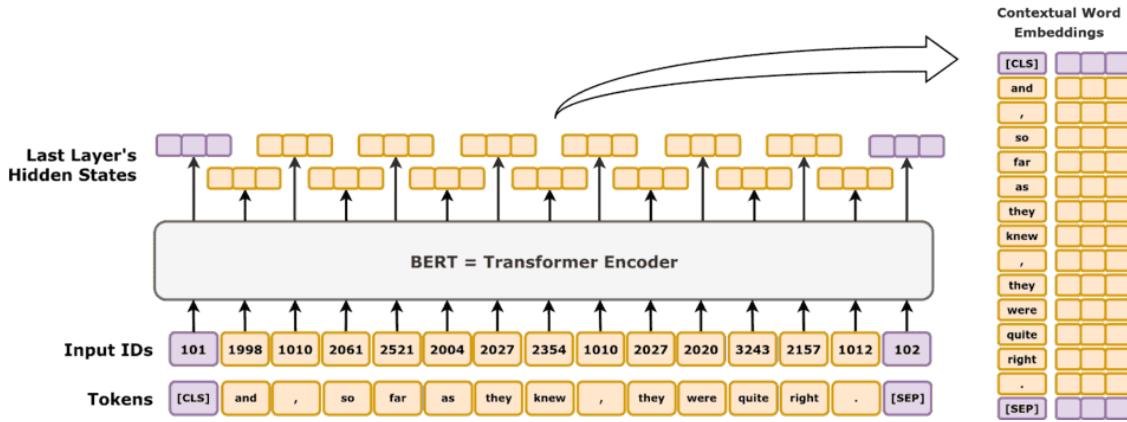


Figura 5.1: Architettura di BERT (schema)

*Fonte: Wikipedia, BERT (language model)*

Il tokenizer di BERT è WordPiece [13], un algoritmo basato su sotto-parole simile alla byte-pair encoding. Quest’ultimo approccio codifica le stringhe di testo in altre più corte, creando e usando una tabella di traduzione. WordPiece, invece, usa un vocabolario composto da 30.000 dimensioni, in cui ognuna di esse corrisponde ad un token. Quest’ultimo è un elemento che non sempre coincide con una parola intera: infatti, esso potrebbe essere un termine intero o anche solo un frammento, come ad esempio un suffisso, che appare frequentemente nei vocaboli. Qualsiasi token non presente nel vocabolario viene sostituito da [UNK] (“sconosciuto”).

Nel primo livello avvengono gli embedding. Essi sono tre: un primo di tipo token, un secondo di posizione ed un ultimo di tipo segmento.

- **Token embedding:** questo è un livello di embedding standard, che traduce un vettore one-hot in un vettore denso in base al suo tipo di token. Un array si dice “one-hot” se è binario, i suoi elementi rappresentano categorie ed essi sono tutti zero, ad eccezione fatta per uno solo di essi, che indica quale categoria è attiva. In NLP questo era il metodo più basilare per rappresentare le parole: ogni termine del vocabolario veniva codificato come un vettore one-hot. Oggi questa pratica è stata messa da parte perché gli array con tale rappresentazione sono sparsi e non molto efficaci;
- **Position embedding:** questo tipo di embedding si basa sulla posizione di un token nella sequenza. BERT utilizza embedding di posizione assoluti, in cui in una successione ogni locazione viene mappata a un vettore a valori reali. Ogni dimensione del vettore è costituita da una funzione sinusoidale che prende la posizione del token nella sequenza come input;
- **Segment embedding:** utilizzando un vocabolario di soli 0 o 1, questo livello di embedding produce un vettore denso in base all’appartenenza del token al primo o al secondo segmento di testo in quell’input. In altre parole, i token

di tipo 1 sono tutti quelli che compaiono dopo quello speciale [SEP]. Tutti i token precedenti sono di tipo 0.

I tre vettori di embedding vengono sommati tra loro per dare la rappresentazione iniziale di ogni token in funzione delle tre informazioni da essi fornite.

Dopo l'embedding, la rappresentazione vettoriale viene normalizzata, utilizzando un'operazione detta “LayerNorm”, che genera un vettore denso a 768 dimensioni per ogni token di input. Essa serve per comprimere le caratteristiche semantiche e sintattiche delle parole, usando un numero di token molto inferiore rispetto alla quantità contenuta dal vocabolario. Infine, i vettori di rappresentazione vengono passati attraverso 12 blocchi di codifica transformer, per poi essere decodificati nuovamente nello spazio del dizionario a 30.000 dimensioni.

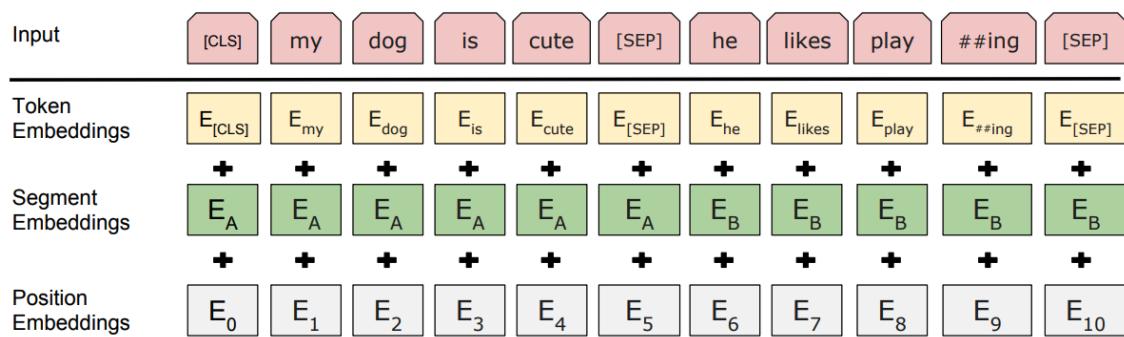


Figura 5.2: Tre tipi di embedding usati da BERT

Fonte: Wikipedia, BERT (language model)

### 5.3 Pre-trained transformer

Non esiste un'unica definizione del concetto di “pre-addestramento” in quanto dipende in parte dal contesto in cui viene attuato. Riporto qui una spiegazione reperita da una fonte, a mio avviso, autorevole:

*“In machine learning, si identifica come “pre-addestramento” la fase di allenamento che forma un modello general-purpose (talvolta chiamato modello di base) su dati pubblicamente disponibili. Il pre-addestramento è spesso seguito dal fine-tuning per dotare il modello di informazioni per compiere un’attività specifica.”*

NIST, pre-training, 2025

Di conseguenza, un modello transformer [1] si dice “pre-addestrato” quando il suo allenamento è avvenuto tramite l'apprendimento di schemi generali riguardanti un campo, avente di solito molte specializzazioni differenti. Tuttavia, ciò implica naturalmente che tale sistema, essendo general-purpose, sarà in grado di affrontare tutti i compiti specifici dell'ambito in cui è stato allenato, ma avrà prestazioni mediocri in ognuno di essi per la natura generalizzante.

Ad esempio, BERT [12] è concepito come modello pre-addestrato. Ciò significa che esso può essere utilizzato in numerose e diverse applicazioni nell'elaborazione del linguaggio naturale. In altre parole, dopo il pre-addestramento, BERT può essere perfezionato con meno risorse e su set di dati più piccolo per ottimizzare le

sue prestazioni in attività specifiche, come l’inferenza del linguaggio naturale e la classificazione del testo, o compiti di generazione basati su sequenze, come la risposta a domande e la creazione di risposte conversazionali.

L’unico modo di cui si dispone per far sì che le prestazioni dei modelli pre-addestrati migliorino è quello di specializzare questi ultimi in una certa attività in particolare, tramite un processo che si chiama “fine-tuning”.

#### 5.4 Fine-tuning

Il “fine-tuning” è un concetto fondamentale da affrontare prima di poter proseguire la trattazione sulla keyword extraction. Cominciamo dando una definizione sintetica:

*“In deep learning, il fine-tuning (in italiano “messa a punto”) è un approccio di trasferimento dell’apprendimento, in cui i parametri di un modello di rete neurale pre-addestrato vengono addestrati su nuovi dati.”*

Wikipedia, Fine-tuning (deep learning), 2025

La messa a punto può essere effettuata sull’intera rete neurale o solo su un sottoinsieme dei suoi livelli, a patto che quelli su cui non si effettua fine-tuning vengano “congelati” (ovvero, non vengano modificati durante la backpropagation).

Un modello può anche essere arricchito con “adapters”, cioè moduli leggeri inseriti nell’architettura del sistema, che modificano lo spazio di inclusione per l’adattamento al dominio. Questi contengono molti meno parametri rispetto al modello originale e possono essere messi a punto in modo efficiente dal punto di vista dei parametri, regolando solo i loro pesi e lasciando immutati i pesi rimanenti del modello.

Il fine-tuning viene in genere realizzata tramite apprendimento supervisionato. Tuttavia, esistono anche tecniche per mettere a punto un modello utilizzando supervisione debole [14]. La messa a punto può essere combinata con apprendimento da rinforzo da un obiettivo basato sul feedback umano per produrre grandi modelli linguistici, come ChatGPT [15] (una versione fine-tuned dei modelli GPT).

Il fine-tuning può degradare la robustezza di un modello rispetto ai cambiamenti di distribuzione [16], [17]. Una possibile mitigazione di questo genere di problemi consiste nell’interpolare linearmente i pesi di un modello fine-tuned con i pesi del modello originale, il che può aumentare notevolmente le prestazioni fuori distribuzione, mantenendo in gran parte le prestazioni in distribuzione del modello messo a punto [18].

Ad esempio, nella pubblicazione originale su BERT i risultati che dimostrano come una piccola quantità di messa a punto abbia consentito di raggiungere prestazioni all’avanguardia in una serie di attività di comprensione del linguaggio naturale: GLUE (composto da 9 attività), SQuAD [19] v1.1 e v2.0, SWAG [20]. Nella stessa pubblicazione, tutti i parametri di BERT sono stati fine-tuned.

## 5.5 KeyBERT

KeyBERT è una libreria che implementa una tecnica di keyword extraction minimale e di facile utilizzo, che sfrutta gli embedding BERT [12] per creare parole e frasi chiave e all'interno di un paragrafo o un documento.

Essa è stata molto preziosa perché consente di impiegare una vasta gamma di varianti fine-tuned del modello BERT, senza doverle addestrare per effettuare l'operazione di estrazione delle parole chiave dal testo, che nel mio caso consiste negli input dell'utente.

Per scegliere quale fine-tuned BERT KeyBERT debba usare come modello di embedding, non bisogna fare altro che specificarne il nome al costruttore in fase di istanziazione dell'oggetto KeyBERT.

Il modello su cui ho scelto di basare l'istanza di KeyBERT è paraphrase-multilingual-mpnet-base-v2, un fine-tuning del modello pre-addestrato MPNet [21]. Quest'ultimo, creato dal MSRA, è basato su un nuovo metodo di pre-addestramento che eredita i vantaggi e supera i limiti di quelli utilizzati per allenare BERT, il quale usò la strategia MLM, e XLNet [22], altro modello pre-addestrato, sviluppato dai ricercatori di Google AI, avente lo scopo di migliorare la gestione del contesto di BERT, che adottò la tecnica PLM. L'innovazione principale apportata dal metodo di addestramento di MPNet fu l'unione delle strategie MLM e PLM.

Ho selezionato paraphrase-multilingual-mpnet-base-v2 perché è stato progettato per operare su testi multilingue da parte di Sentence transformers, conosciuto anche come SBERT [23]. Quest'ultimo è un framework Python all'avanguardia per l'embedding di frasi, testo e immagini, creato da UKP Lab e oggi mantenuto da Hugging Face.

Di seguito possiamo osservare un piccolo stralcio di codice Python in cui mostro come creare correttamente un'istanza di KeyBERT basata su paraphrase-multilingual-mpnet-base-v2:

```
1  from keybert import KeyBERT
2  from sentence_transformers import SentenceTransformer
3
4  kw_model = KeyBERT(SentenceTransformer("sentence-
→ transformers/paraphrase-multilingual-mpnet-base-v2"))
```

Listing 5.1: Istanziazione di KeyBERT

Le opzioni son

## Capitolo 6

# CONCLUSIONE

---

Grazie al progetto che il mio relatore Lorenzo Baraldi mi ha proposto e agli strumenti software avanzati disponibili, come Open WebUI e KeyBERT, ho avuto la possibilità di fare appello a gran parte delle conoscenze e capacità che ho maturato nel corso di questi tre anni presso il Dipartimento di Ingegneria “Enzo Ferrari” di Modena.

Ciò che più mi ha accattivato durante lo sviluppo di questa tesi è stato unire e mettere in pratica in un unico progetto le nozioni teoriche dei corsi frequentati durante il percorso, riuscendone a percepire i risultati anche in un contesto più applicativo.

Lavorare ad Open WebUI sotto la supervisione del relatore e dei correlatori mi ha aiutato ad imparare a risolvere i problemi in modo più autonomo, ad avere più fiducia nelle mie capacità ed a sfruttare i miei studi con un approccio più critico e razionale, meno condizionato dall’incombenza degli esami, che a volte rischia di far perdere di vista quale sia il vero valore della conoscenza.

## BIBLIOGRAFIA

---

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser e I. Polosukhin, «Attention is all you need,» *NeurIPS*, 2017.
- [2] D. Hardt, *Rfc 6749: The oauth 2.0 authorization framework*, 2012.
- [3] D. Fett, R. Küsters e G. Schmitz, «The web sso standard openid connect: In-depth formal security analysis and security guidelines,» in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017.
- [4] M. M. Jones, P. I. J. Bradley e N. N. Sakimura, *Rfc 7519: Json web token (jwt)*, 2015.
- [5] K. Shingala, «Json web token (jwt) based client authentication in message queuing telemetry transport (mqtt),» *arXiv preprint arXiv:1903.02895*, 2019.
- [6] J. Schmidhuber, «Deep learning in neural networks: An overview,» *Neural networks*, 2015.
- [7] D. E. Rumelhart, G. E. Hinton e R. J. Williams, «Learning representations by back-propagating errors,» *nature*, 1986.
- [8] D. Bahdanau, K. Cho e Y. Bengio, «Neural machine translation by jointly learning to align and translate,» *arXiv preprint arXiv:1409.0473*, 2014.
- [9] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes e A. Mian, «A comprehensive overview of large language models,» *ACM Transactions on Intelligent Systems and Technology*, 2025.
- [10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel et al., «Retrieval-augmented generation for knowledge-intensive nlp tasks,» *NeurIPS*, 2020.
- [11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., «Language models are few-shot learners,» *NeurIPS*, 2020.
- [12] J. Devlin, M.-W. Chang, K. Lee e K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» in *NAAACL*, 2019.
- [13] X. Song, A. Salcianu, Y. Song, D. Dopson e D. Zhou, «Fast WordPiece Tokenization,» in *EMNLP*, 2021.
- [14] Y. Yu, S. Zuo e H. W. T. ChaoZhang, «Fine-Tuning Pre-trained Language Model with Weak Supervision: A Contrastive-Regularized Self-Training Approach,» *NAAACL*, 2021.

- [15] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat et al., «GPT-4 Technical Report,» *arXiv preprint arXiv:2303.08774*, 2023.
- [16] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark et al., «Learning transferable visual models from natural language supervision,» in *ICML*, 2021.
- [17] A. Kumar, A. Raghunathan, R. M. Jones, T. Ma e P. Liang, «Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution,» in *ICLR*, 2022.
- [18] M. Wortsman, G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs, R. G. Lopes, H. Hajishirzi, A. Farhadi, H. Namkoong et al., «Robust fine-tuning of zero-shot models,» in *CVPR*, 2022.
- [19] P. Rajpurkar, J. Zhang, K. Lopyrev e P. Liang, «SQuAD: 100,000+ Questions for Machine Comprehension of Text,» in *EMNLP*, 2016.
- [20] R. Zellers, Y. Bisk, R. Schwartz e Y. Choi, «Swag: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference,» *ACL*, 2018.
- [21] K. Song, X. Tan, T. Qin, J. Lu e T.-Y. Liu, «Mpnet: Masked and permuted pre-training for language understanding,» *NeurIPS*, 2020.
- [22] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov e Q. V. Le, «Xlnet: Generalized autoregressive pretraining for language understanding,» *NeurIPS*, 2019.
- [23] N. Reimers e I. Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,» in *EMNLP*, 2019.
- [24] G. I. T. Bray Ed., *RFC 8259: The JavaScript object notation (JSON) data interchange format*, 2017.

# Capitolo A

## APPENDICE

---

Per concludere la trattazione, spiegherò in questo capitolo come ho implementato un plugin per Open WebUI, che le consentisse di effettuare RAG di tipo baseline, e mostrerò infine i risultati di un esperimento qualitativo che dimostra quanto sia utile adottare approcci RAG, anche se molto semplici, per migliorare le performance di piccoli modelli con pochi parametri od addestrati su dati obsoleti.

### A.1 Slurm

Prima di poter parlare del mio lavoro, credo sia opportuno introdurre Slurm. Quest'ultimo è un sistema di gestione di cluster e pianificazione dei lavori open source, fault-tolerant e altamente scalabile per sistemi Linux di grandi e piccole dimensioni. Slurm non richiede modifiche al kernel per il suo funzionamento ed è relativamente autonomo. Come gestore del carico di lavoro del cluster, Slurm svolge tre funzioni chiave: in primo luogo, assegna agli utenti l'accesso esclusivo e/o non esclusivo alle risorse (nodi di elaborazione) per un certo periodo di tempo, in modo che possano svolgere il proprio lavoro; In secondo luogo, fornisce un framework per l'avvio, l'esecuzione e il monitoraggio del lavoro (normalmente un lavoro parallelo) sull'insieme di nodi allocati; infine, Slurm arbitra la contesa per le risorse gestendo una coda di lavoro in sospeso.

Per noi è fondamentale parlare di Slurm perché è lo strumento che permette a tutti gli utenti, me compreso, di lavorare contemporaneamente sul cluster, sfruttando le risorse in modo equo e controllato.

Invece di far eseguire direttamente al cluster qualsiasi compito, mi sono sempre occupato di chiedere a Slurm di fare da intermediario tra me e la macchina, così da sfruttare le risorse in modo efficiente e da non intralciare il lavoro altrui. Ciò è stato possibile grazie al sofisticato comando `srun`, che ho deciso di configurare all'interno di uno shell script, chiamato `s_shell`, “slurm shell”. Questo codice mi consentiva di chiedere a Slurm di avviare sul cluster una shell interattiva per me della durata di un ora con a disposizione le risorse che richiedevo.

Vediamo lo script:

```

1 #!/bin/bash
2
3 srun --immediate=10 \
4     --account=infra_mllm \
5     --partition=all_serial \
6     --gres=gpu:1 \
7     --nodes=1 \
8     --mem=20G \
9     --nodelist=ailb-login-03 \
10    --ntasks=1 \
11    --time=01:00:00 \
12    --pty bash

```

Listing A.1: Shell script per avviare una shell interattiva tramite Slurm

Senza addentrarci nel dettaglio di tutte le opzioni, poniamo attenzione al fatto che sia sempre necessario specificare il proprio account, altrimenti Slurm rifiuterà la richiesta. Ciò avviene perché i manutentori del cluster hanno installato in Slurm il plugin dedicato per l'accounting ed è stata impostata l'identificazione obbligatoria.

Ho deciso di inserire questo comando in uno script bash perché era molto frequente per me avere necessità di shell interattive per effettuare dei test.

## A.2 Ollama

Come abbiamo accennato nell'introduzione, Ollama è un software che serve per permettere agli LLM di essere eseguiti in locale.

Prima di poter eseguire azioni tramite questo potente software, è necessario avviarlo usando il comando `ollama serve` in una shell con risorse allocate da Slurm. Tramite quest'ultimo, in esecuzione sulla porta 11434, è possibile comunicare con Ollama o qualsiasi modello installato in esso. L'interazione con i LLM è interamente gestita da Open WebUI, che sfrutta il software protagonista di questo paragrafo come intermediario, siccome essa lo supporta nativamente.

Dopo aver avviato il server Ollama, per scaricare un modello pronto all'uso, serve lanciare il comando `ollama pull <modello>`. Dopodiché, conviene dichiarare la variabile d'ambiente `OLLAMA_MODELS=./ollama_models` per evitare di dover specificare la cartella in cui si trovano i modelli in Ollama. Per eseguire un LLM in generale si usa il comando `ollama run <modello>`.

Ora che sappiamo come eseguire i modelli in modo che siano raggiungibili da Open WebUI, vediamo come avviare quest'ultima per lo sviluppo e come configurare il SSO.

## A.3 Avvio di Open WebUI

Per poter testare eventuali modifiche apportate ad Open WebUI in fase di sviluppo, è necessario essere in grado di eseguirlo tramite Nodejs, noto framework per lo sviluppo web. Sui cluster HPC spesso i software come quest'ultimo sono disattivati, quindi per poter usare Nodejs serve lanciare il comando `modules load node-js`.

Successivamente, bisogna entrare nella cartella radice del codice sorgente di Open WebUI, che io ho chiamato `open-webui`, ed eseguire `npm install` per installare le dipendenze richieste da Nodejs per fare in modo che tutto il frontend funzioni correttamente. L'operazione richiederà un po' di tempo, quindi conviene nel mentre aprire una nuova shell e creare un ambiente virtuale con Python 3.11, spostarsi nella cartella `open-webui/backend/` ed installare le dipendenze necessarie per eseguire il backend con il comando `pip install -r requirements.txt`, dove `requirements.txt` è il file di testo in cui sono elencate le dipendenze scaricate da `pip`, nonché il package manager di Python. Si consiglia di prestare attenzione alla versione di PyTorch: in caso di problemi, come nel mio caso, consiglio di fare riferimento al sito ufficiale di PyTorch ed installarne una versione compatibile con il sistema in uso. Per la mia configurazione quella corretta era la 11.8.

Se l'installazione delle dipendenze è andata a buon fine, allora non resta che, in tre shell distinte, avviare il frontend ed il backend di Open WebUI e il server di Ollama. Per il primo basta posizionarsi nella cartella `open-webui` ed eseguire `npm run dev`. Il backend si avvia spostandosi in `open-webui/backend/`, attivando l'ambiente virtuale Python creato nella fase precedente ed eseguendo, infine, lo script `sh dev.sh`. Il server di Ollama si avvia semplicemente spostandosi nella sua cartella di installazione, allocando le risorse per il processo usando Slurm ed eseguendo `ollama serve`. Per avviare Ollama è possibile usare uno script, simile a quello che abbiamo visto nella sezione su Slurm per le shell interattive con allocazione di risorse, aggiungendo subito dopo lo shabang la linea `cd <percorso_alla_cartella_di_Ollama>` e sostituendo il parametro `--pty bash` con `ollama serve`.

Il processo di avvio potrebbe richiedere tempo. Al termine dovremmo vedere una schermata come la seguente:



Figura A.1: Schermata iniziale di Open WebUI

### A.4 Configurazione del SSO

Ora possiamo configurare il single sign-on. Seguendo le indicazioni fornite dalla documentazione di Open WebUI ho potuto configurare un SSO perfettamente funzionante.

Tutto quello che bisogna fare è, a partire dalla cartella radice del progetto, modificare il file `/backend/dev.sh`. Qui, con i dati che il mio relatore mi ha gentilmente fornito, ho configurato le seguenti variabili d'ambiente:

- `OAUTH_CLIENT_ID`: deve contenere il client ID per OIDC;
- `OPENID_PROVIDER_URL`: a questa variabile bisogna assegnare l'OIDC well known URL;
- `OPENID_REDIRECT_URI`: questo campo è opzionale, ma si è rivelato necessario nel mio caso specifico per gestire problemi relativi al port forwarding. Non avendo il cluster un browser installato al suo interno, si è presentata la necessità di instaurare una connessione SSH verso un sistema, in questo caso il mio computer, che ne possiede uno. In questo modo ho ottenuto una UI accessibile da parte degli utenti che vogliono accedere a Open WebUI;
- `WEBUI_URL`: deve contenere l'URL base del frontend del sito web. Questo passaggio è fondamentale perché se questa variabile non viene configurata, allora, dopo che il processo di autenticazione è terminato con successo, il backend di Open WebUI ritornerà un JSON [24] `{"Details": "Not Found"}` e non sarà in grado di far accedere l'utente alla propria homepage.

Tutte queste variabili d'ambiente devono essere impostate come globali, tramite l'apposita direttiva Unix `export`. Questa operazione è necessaria perché il web server Python Uvicorn crea dei processi figli, che non sarebbero in grado di ricevere la configurazione se le variabili d'ambiente non fossero globali.

Facendo riferimento al sequence diagram visto nella sezione precedente, nel caso di Open WebUI, l'end user è il frontend del sito web, la RP è il backend, mentre l'OIDC provider riassume gli endpoint esposti da UniMoRe.

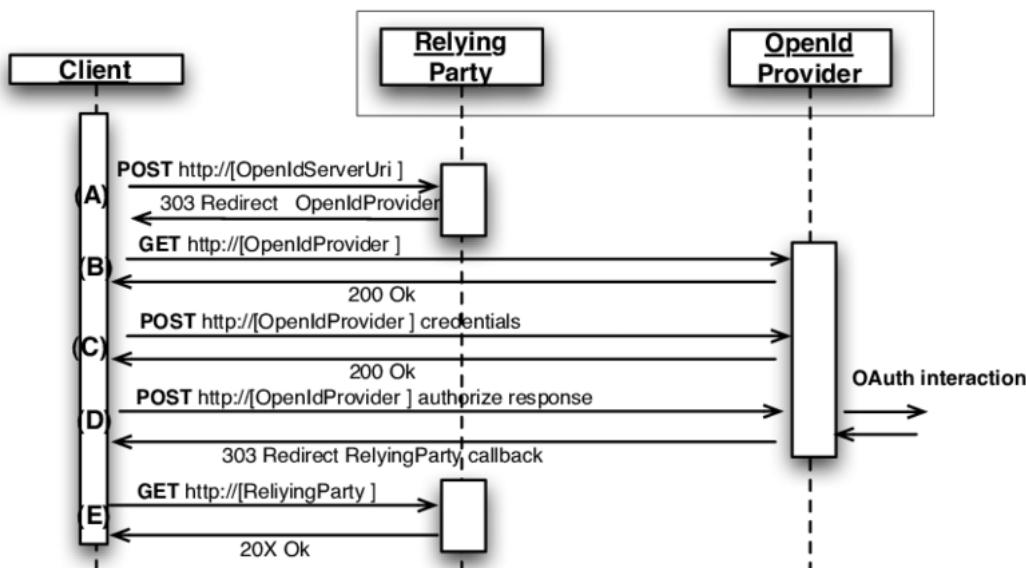


Figura A.2: OIDC - Flusso delle HTTP request

## APPENDICE A. APPENDICE

---

Per verificare che il SSO sia configurato correttamente e funzionante, avviamo Open WebUI come illustrato nella sezione precedente e, una volta raggiunta la pagina iniziale, premiamo sulla freccia al centro della schermata in basso e dovremmo vedere l'interfaccia di accesso con i campi per autenticarsi come amministratore e, sotto di essi, un pulsante che ridirige alla pagina di login AIImage Lab.

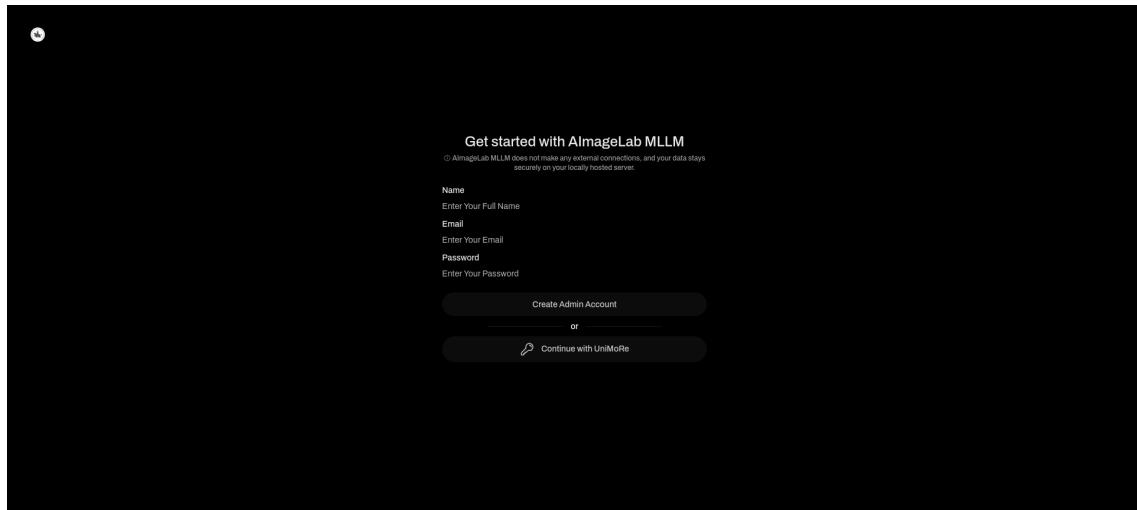


Figura A.3: Schermata di login con SSO attivo

Premendo sul tasto, dovremmo raggiungere questa pagina:

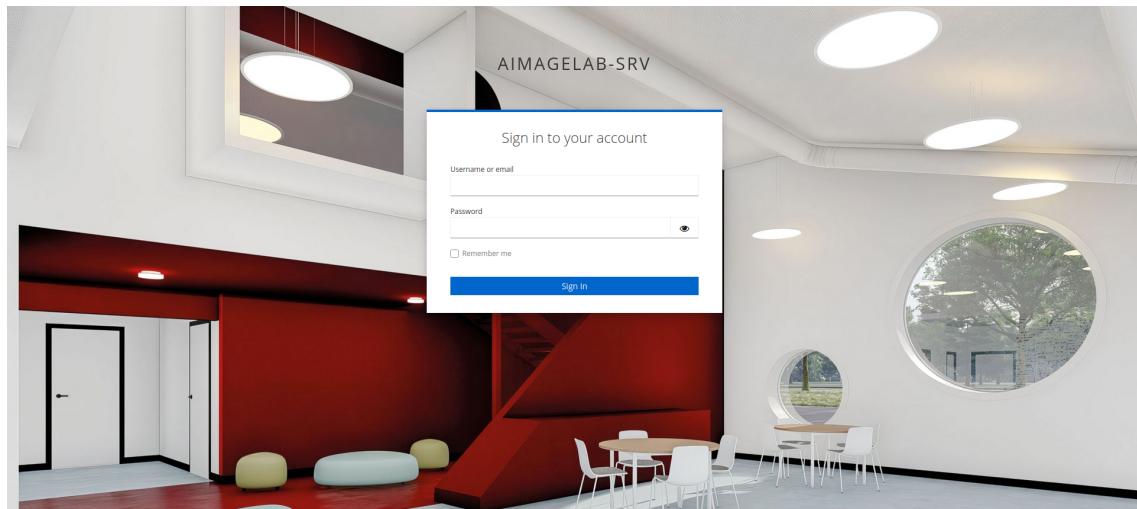


Figura A.4: Schermata di login AIImage Lab

Infine, dovremmo essere rediretti alla schermata principale di Open WebUI correttamente autenticati con il nostro account UniMoRe.

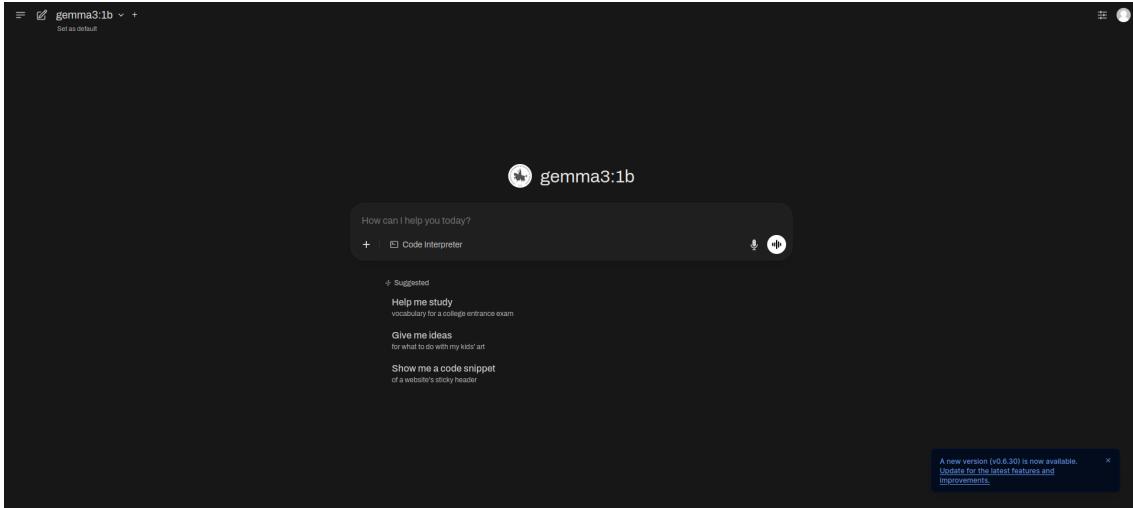


Figura A.5: Open WebUI homepage

Come si vede in quest'ultima immagine, Open WebUI ha rilevato la presenza di Ollama in esecuzione sulla porta 11434, su cui avevo già installato il modello “gemma3:1b”. Si può selezionare con quale LLM si desidera interagire tramite il menu a tendina che si vede nella nell’angolo in alto a sinistra della schermata.

Come abbiamo potuto osservare dalle recenti immagini, nel corso dello sviluppo ho provveduto a sostituire le icone del sito web con il logo di UniMoRe. Per farlo ho utilizzato uno strumento online che, data un’immagine di input, è in grado di generare un pacchetto di icone in formato standard con il codice necessario per includerle all’interno del progetto, del quale va specificata la tecnologia usata per il frontend, nel mio caso SvelteKit.

Vediamo nella seguente sezione come ho attivato in Open WebUI il plugin per effettuare RAG che ho implementato.

## A.5 RAG in Open WebUI

In Open WebUI è implementata nativamente la possibilità di richiedere la raccolta di informazioni sul web. Ciò è possibile recandosi nelle impostazioni, tramite il pulsante che raffigura la foto profilo dell’utente sull’interfaccia in alto a destra, entrando in “Admin panel”, poi nella nuova schermata selezionando “Settings”. Da qui, nella sezione denominata “Web Search”, è possibile attivare la funzionalità di ricerca su web attivando il selettore relativo all’impostazione chiamata “Web Search”.

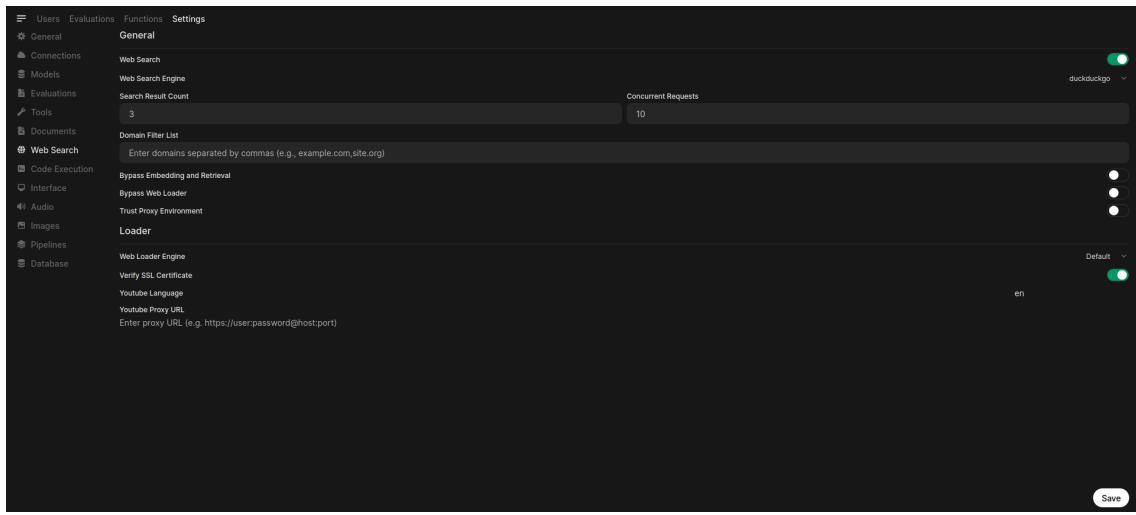


Figura A.6: Sezione delle impostazioni relativa a Web Search. Il selettore è attivo

Tuttavia, attivare il pulsante non basta per avere un sistema di ricerca fonti funzionante. Essendo Open WebUI un software pensato per essere estensibile e personalizzabile, bisogna selezionare quale sia il motore a cui appoggiarsi per la web search tra quelli disponibili. Per farlo, non bisogna fare altro che aprire il menu a tendina dell’impostazione “Web Search Engine”, come si può vedere dall’ultima immagine appena sotto il selettore che abbiamo appena attivato. Per fare una breve prova, io avevo usato DuckDuckGo perché può essere adoperato senza specificare una API key.

Nell’ultima schermata che abbiamo visto notiamo anche che è possibile specificare il numero di risultati che si desidera venga trattenuto dalla ricerca online. Ai fini di test, ho scelto di richiederne tre.

Per rendere qualsiasi modifica delle impostazioni di Open WebUI permanente, compresa quella che abbiamo visto per la Web Search, basta premere il pulsante “Save” nell’angolo in basso a destra.

Dopo aver eseguito queste operazioni, entrando in una qualsiasi chat con un LLM, sarà possibile richiedere esplicitamente la ricerca di fonti online attivando il pulsante “Web Search”.

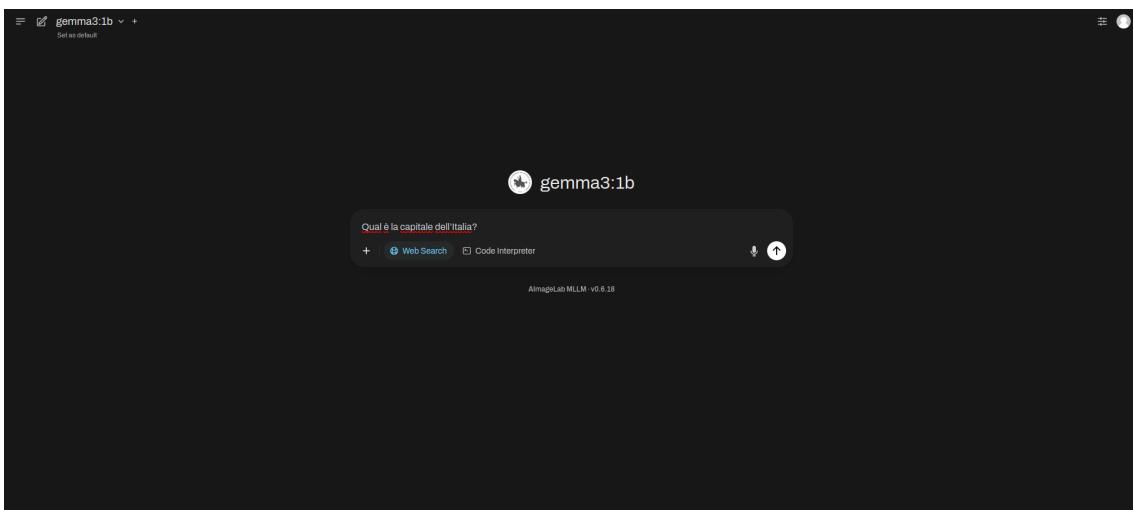


Figura A.7: Attivazione della Web Search

Tuttavia, durante i test non ero soddisfatto dai risultati ottenuti tramite la Web Search perché i motori disponibili gratuitamente davano in output fonti di dubbia affidabilità. Ho deciso di documentare qui con due schermate i risultati di una query molto semplice per vederne insieme un esempio. Nella prima si vedono le fonti elencate in fondo alla risposta del modello, mentre nella seconda i dettagli relativi ad uno dei risultati della ricerca.

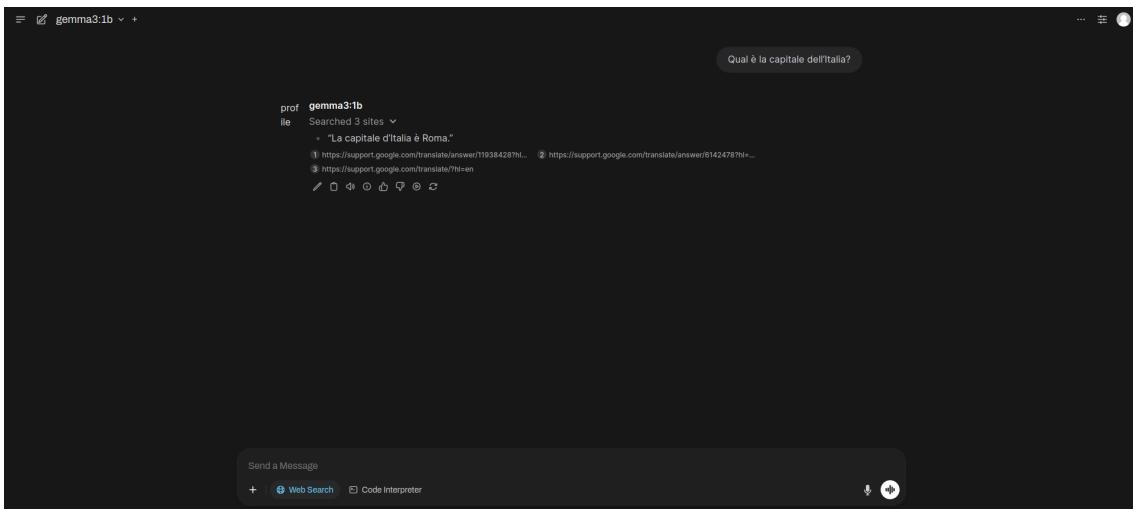


Figura A.8: Esempio di chat con Web Search attiva

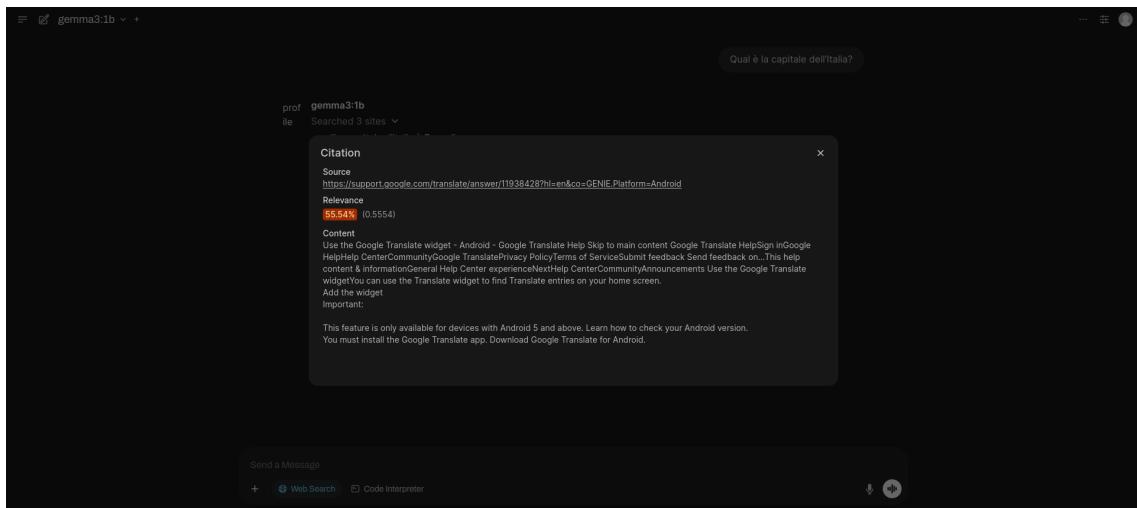


Figura A.9: Esempio di informazioni raccolte

Inoltre, l'unico metodo per effettuare RAG nativo di Open WebUI è quello offline. Ho deciso dunque di scrivere uno script che fosse in grado di intercettare la query dell'utente, di inferirne la lingua e di estrarne le parole chiave, per poi sfruttare queste ultime per raccogliere informazioni da Wikipedia quanto al contesto dell'input dell'utente.

Lo script che ho realizzato si trova nella gerarchia della versione del progetto di Open WebUI da me modificata, disponibile sul mio profilo Github come repository derivata da quella ufficiale, in una cartella denominata “functions”. Il nome di quest’ultima non è casuale, ma si pone in continuità con la scelta degli sviluppatori del software originale di aggiungere il supporto nativo per l’integrazione di plugin Python, definiti “Functions” che possono essere di tre tipi: “Pipes”, “Filters” e “Actions”.

Cito testualmente la documentazione di Open WebUI:

*“A differenza degli strumenti esterni che potrebbero richiedere integrazioni complesse, le Functions sono integrate e vengono eseguite all’interno dell’ambiente Open WebUI. Ciò significa che sono veloci, modulari e non dipendono da dipendenze esterne. Considerate le funzioni come blocchi da costruzione modulari, che consentono di migliorare il funzionamento di Open WebUI, adattandoli esattamente alle proprie esigenze.”*

Open WebUI, Functions, 2025

Senza entrare nei dettagli di ognuno di queste categorie, i Filters sono quelli che si adattavano al mio caso d’uso. Tramite questi script si possono effettuare elaborazioni di qualsiasi genere alle query, attraverso funzioni descritte nella documentazione, intercettando sia l’input dell’utente per il LLM sia l’output prodotto dal modello.

Lo scheletro di un filtro avente un pulsante per permetterne l'attivazione dalla UI si presenta così:

```

1  from pydantic import BaseModel, Field
2  from typing import Optional
3
4  class Filter:
5      class Valves(BaseModel):
6          pass
7
8      def __init__(self):
9          # Initialize values (optional configuration for the
10             ↪ Filter)
11          self.valves = self.Valves()
12
13          # Next line creates a switch UI in Open WebUI
14          self.toggle = True
15
16          # Use SVG Data URI!
17          self.icon = """data:image/svg+xml;base64,PHN2ZyB4 ...
18             ↪ here continues SVG URI"""
19          pass
20
21      async def inlet(
22          self, body: dict, __event_emitter__, __user__:
23              ↪ Optional[dict] = None
24      ) -> dict:
25          await __event_emitter__(
26              {
27                  "type": "status",
28                  "data": {
29                      "description": "Toggled!",
30                      "done": True,
31                      "hidden": False,
32                  },
33              }
34          )
35          return body
36
37      def stream(self, event: dict) -> dict:
38          # This is where you modify streamed chunks of model
39             ↪ output.
40          print(f"stream event: {event}")
41          return event
42
43      def outlet(self, body: dict) -> None:
44          # This is where you manipulate model outputs.
45          print(f"outlet called: {body}")

```

Listing A.2: Struttura di una Filter Function

## APPENDICE A. APPENDICE

I Filter devono essere scritti dall'editor di Open WebUI. Per farlo, bisogna tornare di nuovo nell'Admin panel, ma questa volta entrare nella sezione "Functions". Dopodiché, premendo il pulsante + in alto a destra e selezionando la voce "New Function", ci troveremo all'interno dell'editor.

Nel caso siano necessari moduli Python che non sono già installati di default nell’ambiente virtuale nativo di Open WebUI, occorre scaricarli manualmente tramite pip, interrompendo l’esecuzione dell’applicazione.

≡ Users Evaluations Functions Settings

< Function Name

Function ID

Function Description

```
1 #!/usr/bin/python
2 # title: Example Filter
3 # author: open-webui
4 # author url: https://github.com/open-webui
5 # funding url: https://github.com/open-webui
6 # version: 0.1
7 #"""
8 #
9 # from pydantic import BaseModel, Field
10 # from typing import Optional
11 #
12 #
13 # class Filter:
14 #     class Valves(BaseModel):
15 #         priority: int = Field(
16 #             default=0, description="Priority level for the filter operations."
17 #         )
18 #         max_turns: int = Field(
19 #             default=0, description="Maximum allowable conversation turns for a user."
20 #         )
21 #     pass
22 #
23 # class UserValves(BaseModel):
24 #     max_turns: int = Field(
25 #         default=4, description="Maximum allowable conversation turns for a user."
26 #     )
27 #     pass
28 #
29 # def __init__(self):
30 #     # Indicates custom file handling logic. This flag helps disengage default routines in favor of custom
31 #     # implementations, informing the WebUI to defer file-related operations to designated methods within this class.
32 #     # Alternatively, you can remove the files directly from the body in from the inlet hook
33 #     # self.file_handler = True
34 #
35 #     # Initialize 'Valves' with specific configurations. Using 'Valves' instance helps encapsulate settings,
36 #     # which ensures certain are managed cohesively and not confused with external file-like file handler.
37 #
38 #     # don't install random functions from sources you don't trust.
39 #     # Save
```

Figura A.10: Schermata dell'editor

Come si può vedere, l'applicazione mette già a disposizione una struttura base un po' diversa da quella che ho mostrato nella pagina precedente. Consiglio vivamente di mantenere tale schema iniziale ed estenderlo siccome è sempre aggiornato, mentre quello disponibile sulla documentazione potrebbe non esserlo in caso di un recente aggiornamento.

Una volta terminata la stesura del proprio script, bisogna salvarlo, premendo su “Save” in angolo in basso a destra, ed è possibile visualizzarlo nella lista delle Functions in Open WebUI. Se si desidera attivarlo, bisogna premere sul selettori che giace sulla sua stessa riga, poi aprire il menu contrassegnato dai tre puntini a sinistra del bottone appena descritto ed abilitare l’opzione “Global”, per rendere il plugin disponibile per tutti gli utenti.

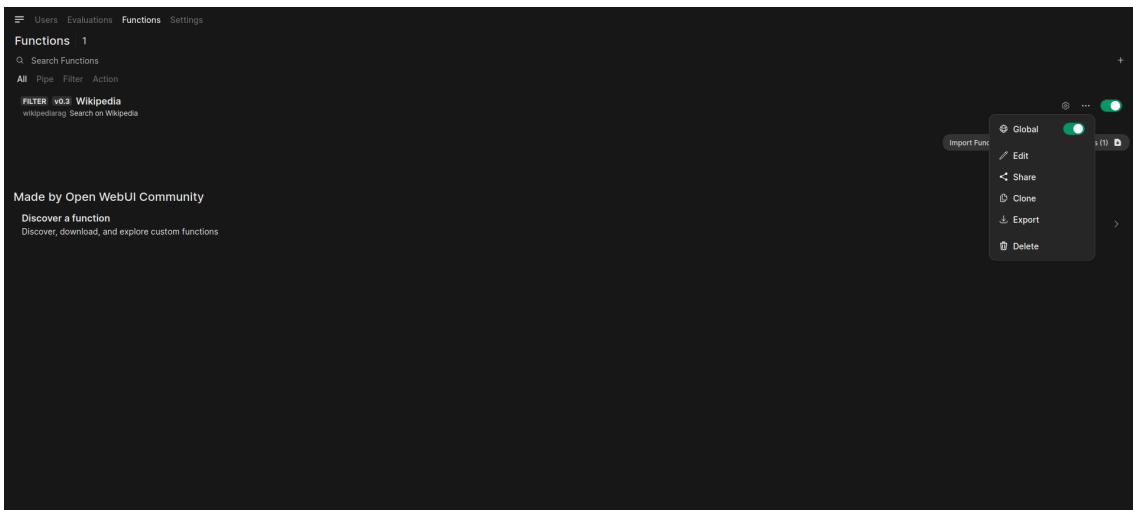


Figura A.11: Schermata dell'editor

Le Functions possono essere scaricate localmente tramite il pulsante “Export Functions”, in basso a destra sotto la lista delle Functions disponibili nell’applicazione. Esse verranno salvate in formato JSON [24]. Se si dispone già di un file JSON contenente una Function, allora essa può essere caricata su Open WebUI usando il pulsante “Import Functions”.

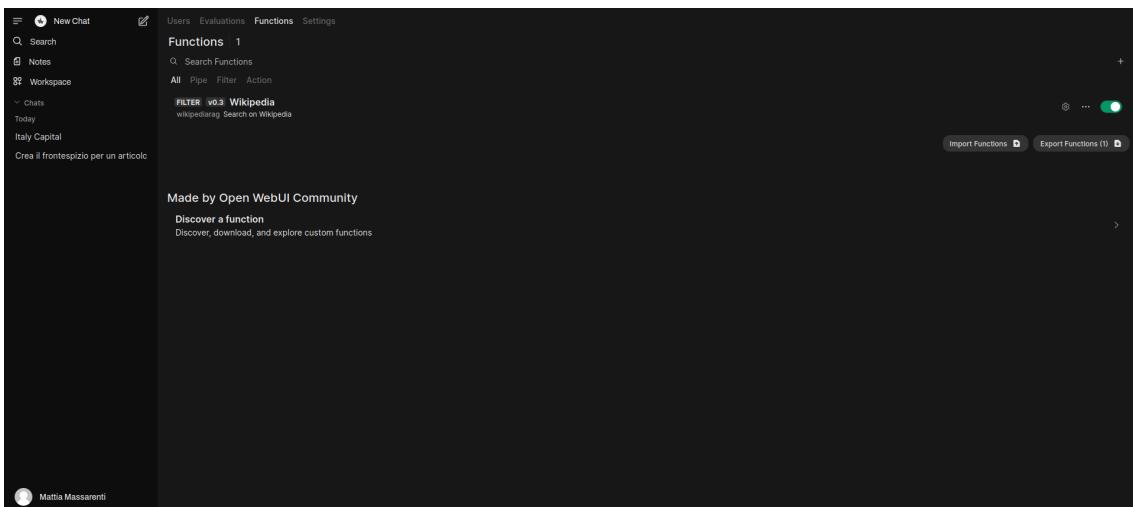


Figura A.12: Pulsanti di import ed export per le Functions

## APPENDICE A. APPENDICE

---

Per concludere, vediamo con un piccolo esperimento come l'output, usando il modello leggero gemma3:1b, migliori nel caso in cui si attivi il plugin per effettuare la RAG che ho implementato, rispetto a quello che si otterrebbe dal LLM consci solo dei dati visti in addestramento. Chiederò al modello di dirmi dove si trova Correggio, un paese italiano della provincia di Reggio Emilia, che sicuramente gemma3:1b non conosce in quanto LLM di piccole dimensioni.

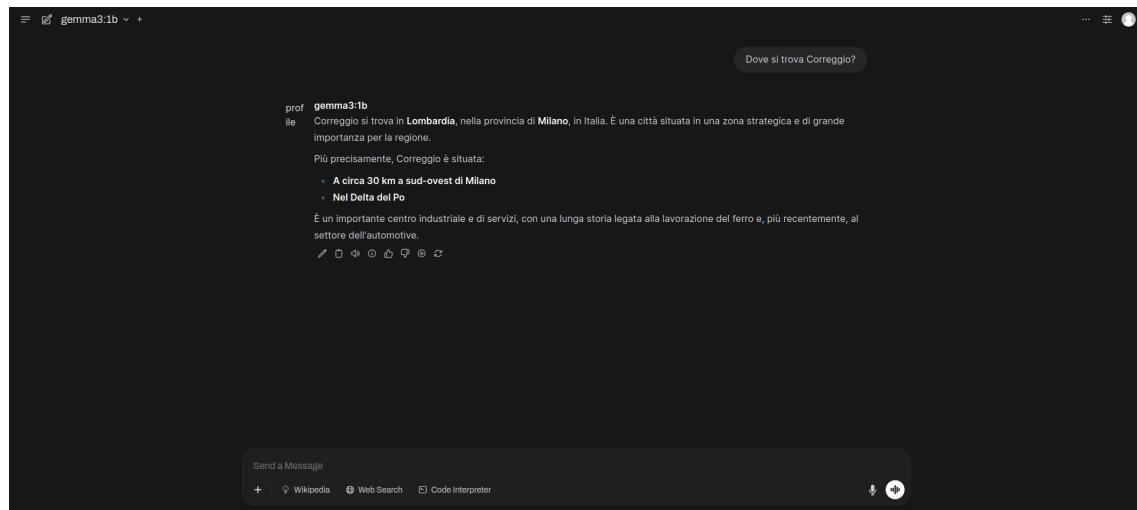


Figura A.13: Esempio di query senza RAG su Wikiepdia

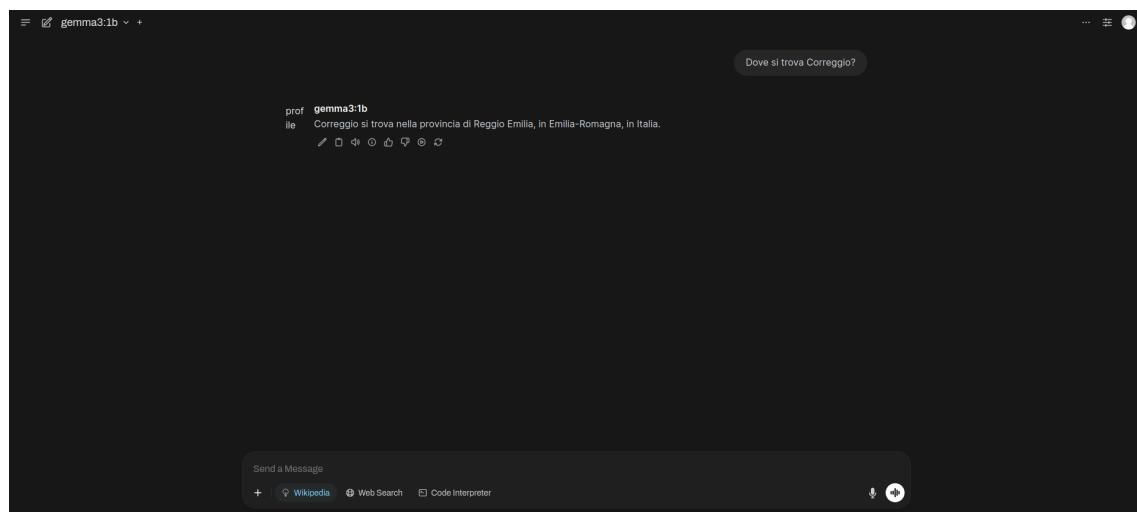


Figura A.14: Output della stessa query con RAG su Wikiepdia

Da questo tipo di RAG si trae il vantaggio di contestualizzare meglio le domande o le richieste dell'utente e di garantire l'affidabilità della fonte, che è sempre Wikipedia.