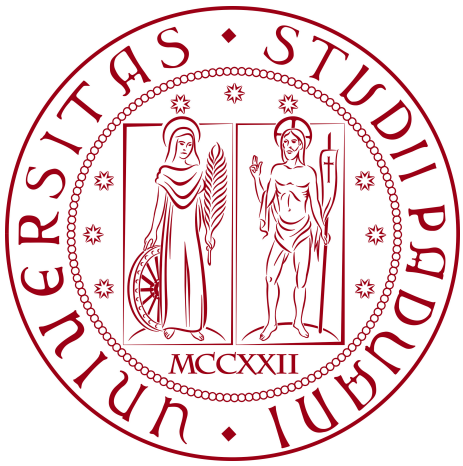
Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



Innovazioni nell'automatizzazione dei Test di Codice tramite Large Language Models (LLM)

Tesi di Laurea Triennale

Relatore

Prof. Ballan Lamberto

Laureanda

De Laurentis Arianna Pia

Matricola 2008077

ANNO ACCADEMICO 2023-2024

"I think you should always bear in mind that entropy is not on your side."

Ringraziamenti

Ringrazio il Professore Lamberto Ballan, relatore di questa tesi di laurea, per la disponibilità che mi ha dato durante la stesura dell'elaborato.

Desidero esprimere la mia più sincera gratitudine ai miei genitori, il cui sostegno incondizionato è stato fondamentale per il raggiungimento di questo traguardo.

Ringrazio Davide per essermi sempre stato vicino e avermi motivata fortemente durante tutto questo percorso.

Ho infine il desiderio di ringraziare il mio tutor aziendale Gregorio Piccoli e l'azienda Zucchetti per avermi permesso di crescere professionalmente.

Padova, Luglio 2024 *Arianna Pia De Laurentis*

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi ore, della laureanda De Laurentis Arianna Pia presso l'Università degli Studi di Padova. Il tirocinio è stato svolto sotto la supervisione del CTO dell'azienda ospitante Zucchetti SPA, Gregorio Piccoli. Tutor interno è stato il Prof. Ballan Lamberto.

Questa tesi esplora l'uso dei Large Language Models (LLM) come supporto fondamentale agli sviluppatori, concentrandosi principalmente sulla generazione di test di unità e dati sintetici.
Si è anche approfondita l'applicazione del Federated Learning (FL) come metodologia avanzata per l'addestramento dei modelli di Machine Learning, focalizzandosi sulla protezione della privacy dei dati di addestramento.

Il progetto è stato suddiviso in due parti: la prima dedicata allo studio dell'applicazione dei Large Language Models (LLM) nella generazione dei test e nella generazione dei dati sintetici.
La seconda parte è dedicata allo studio del Federated Learning, per garantire privacy e sicurezza nel processo di addestramento dei modelli.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

All'interno del capitolo verranno presentati in ordine: una introduzione al progetto di stage, l'azienda proponente, l'organizzazione del testo e le convenzione tipografiche

1.1 Introduzione al Progetto

L'avanzamento dei Large Language Model (LLM)$_G$ ha rivoluzionato il modo in cui gli sviluppatori si approcciano allo sviluppo software. Questi modelli sono estremamente utili per una varietà di compiti, tra cui suggerire correzioni, produrre porzioni di codice e generare documentazione. Inoltre, facilitano la comprensione del codice esistente, migliorano la qualità del software e accelerano il processo di debugging. Utilizzando gli LLM, gli sviluppatori possono anche automatizzare compiti ripetitivi, esplorare nuove soluzioni e migliorare la collaborazione all'interno dei team di sviluppo. In questo progetto ci si focalizza sull'esplorare metodi per utilizzare gli LLM per automatizzare la generazione di test di unità e per generare dati sintetici.

Nella fase iniziale del progetto è stata svolta un'analisi su come gli LLM possano assistere lo sviluppatore nella generazione di test di unità. Successivamente, si è condotta una ricerca sull'utilità degli LLM nella produzione di dati sintetici. Infine, per analizzare la questione della privacy e sicurezza dei dati utilizzati nel processo di addestramento dei modelli di Machine Learning (ML)$_G$, è stata prodotta un'implementazione del Federated Learning (FL)$_G$.

Abbiamo dimostrato la fattibilità di queste tecniche attraverso una serie di esperimenti e di Proof of Concept (PoC)$_G$.

1.2 L'azienda

Zucchetti SPA è un'azienda leader nel settore tecnologico, con una lunga storia di innovazione e sviluppo di soluzioni avanzate per il business.
Fondata nel 1978, Zucchetti è cresciuta fino a diventare uno dei principali fornitori di software, soluzioni ICT e servizi per imprese di diverse dimensioni e settori. L'azienda offre una vasta gamma di prodotti, tra cui software gestionale, soluzioni per la gestione delle risorse umane, sistemi di sicurezza, fatturazione elettronica e gestione documentale, oltre a soluzioni verticali per settori specifici come retail, hospitality, healthcare, automotive e manufacturing.

Figura 1.1: Logo Zucchetti SPA

1.3 Organizzazione del testo

Il secondo capitolo espone l'idea, gli obiettivi e la pianificazione del progetto;

Il terzo capitolo approfondisce le soluzioni già esistenti ai problemi analizzati;

Il quarto capitolo illustra le decisioni architetturali e le risorse impiegate nella realizzazione dell'applicazione Coding Assistant;

Il quinto capitolo illustra come garantire privacy e sicurezza nel processo di addestramento di un LLM proponendo una versione semplificata di Federated Learning;

Nel sesto capitolo riassume i risultati ottenuti.

Riguardo la stesura del testo, sono state adottate le seguenti convenzioni tipografiche:

- I termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*$_{\boldsymbol{G}}$;
- i termini con maggiore rilevanza sono evidenziati con il carattere **grassetto**.

Capitolo 2

Descrizione del Progetto

In questo capitolo verranno discussi in modo esteso gli argomenti chiave del progetto, l'idea di base su cui si fonda, gli obiettivi da raggiungere e la pianificazione del lavoro.

2.1 Presentazione dei Contenuti

In questa sezione verranno presentati gli argomenti fulcro del progetto che verranno approfonditi in dettaglio nei capitoli successivi.

2.1.1 Large Language Model (LLM)

Il linguaggio svolge un ruolo fondamentale nel facilitare la comunicazione e l'espressione tra esseri umani, oltre che alla loro interazione con le macchine [1]. La sua importanza è radicata nella capacità di trasmettere idee, emozioni e istruzioni in modo chiaro ed efficace, creando un ponte tra individui e tecnologie. Negli ultimi anni nell'ambito del Machine Learning (ML), i Large Language Models (LLM) hanno guadagnato crescente attenzione grazie alle loro sorprendenti capacità. Questi modelli, addestrati su enormi quantità di dati, hanno dimostrato di poter comprendere e generare linguaggio naturale con un alto grado di accuratezza e fluidità.
Un LLM è un algoritmo di Deep Learning (DL)$_G$ in grado di eseguire una varietà di compiti legati al Natural Language Processing (NLP)$_G$, quali: interpretare, tradurre, prevedere o generare testo e altri contenuti. Questi modelli utilizzano l'architettura

definita Transformer$_G$[2], caratterizzata da un Encoder per il processamento dell'input e da un Decoder per la generazione dell'output. Vengono addestrati su vasti dataset, permettendo loro di acquisire una maggiore accuratezza nelle risposte generate [3]. Gli LLM sono anche noti come Neural Networks (NN)$_G$, sistemi di calcolo ispirati al funzionamento del cervello umano. Le reti neurali operano tramite una rete di nodi stratificati, simili ai neuroni, che elaborano e trasmettono informazioni attraverso connessioni ponderate, permettendo al modello di apprendere e migliorare le sue prestazioni su compiti complessi di Natural Language Processing (NLP).

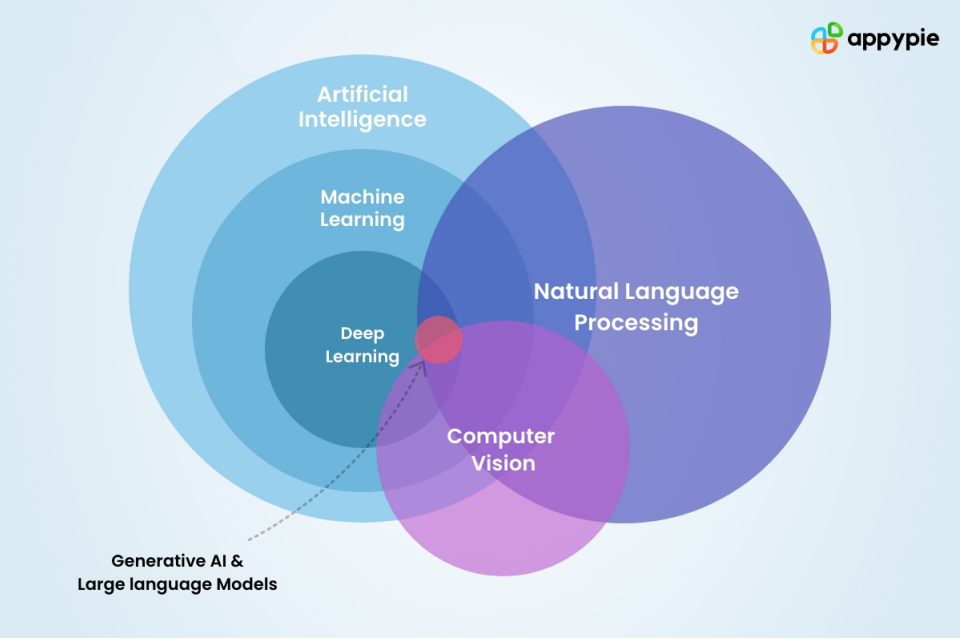


Figura 2.1: Immagine branche AI

La fama crescente dei Large Language Models è dovuta alla vastità degli ambiti applicativi che essi hanno. A livello educativo educativo fungono da assistenti all'apprendimento, in grado di creare sistemi di tutoring automatico che aiutano a comprendere concetti complessi attraverso spiegazioni dettagliate, oppure fungendo da supporto alla scrittura per correggere errori grammaticali e suggerire alternative o anche traduzioni al testo [4]. In ambito lavorativo gli LLM possono automatizzare una serie di compiti che richiedono comprensione e produzione del linguaggio naturale. Ad esempio, possono essere utilizzati per rispondere alle domande nei servizi clienti, per generare report automatici o per analizzare grandi quantità di testo per estrarre informazioni utili [5]. Particolare attenzione viene data al loro utilizzo nello

sviluppo software. Automatizzando compiti ripetitivi, gli LLM possono contribuire a migliorare il processo di produzione del software riducendo il carico di lavoro degli sviluppatori e aumentando l'efficienza complessiva [6]. Ad esempio, possono scrivere documentazione per il codice sorgente, suggerire correzioni e miglioramenti al codice esistente o generare codice a partire da specifiche di alto livello. Nei capitoli successivi, verrà approfondito in dettaglio come gli LLM possano generare test a partire dalla documentazione e dal codice sorgente [7] [8]. Sarà inoltre trattata la loro capacità di generare dati sintetici per vari scopi, contribuendo a migliorare l'efficacia e l'efficienza dei processi di sviluppo software. Tuttavia, è importante notare che gli LLM presentano dei limiti e quindi non sostituiscono completamente gli sviluppatori, ma piuttosto li supportano nel loro lavoro quotidiano, consentendo loro di concentrarsi su compiti più creativi e ad alto valore aggiunto [9] [10].

Alcuni dei modelli LLM più utilizzati sono [11]:

- **GPT-3 (Generative Pre-trained Transformer 3)** sviluppato da OpenAI;
- **BERT (Bidirectional Encoder Representations from Transformers)** sviluppato da Google;
- **T5 (Text-To-Text Transfer Transformer)** sviluppato da Google;
- **LLaMA (Language Learning through Model Adaptation)** sviluppato da Meta AI;
- **Mistral** sviluppato da mistral.ai;

Figura 2.2: Logo di Chat GPT

Figura 2.3: Logo di LLamA

2.1.2 Retrieval-Augmented Generation (RAG)

Analizzando le capacità dei Large Language Models (LLM), emergono alcuni limiti che è importante evidenziare:

- **Allucinazioni e Output Incorretti**: i LLM possono generare informazioni false, il che può indurre gli utenti a credere erroneamente che il contenuto generato sia affidabile [12];
- **Lunghezza dell'Input e dell'Output**: i LLM presentano alcune limitazioni tecniche riguardanti la lunghezza del prompt di input e del testo di output. Sebbene molti LLM possano accettare prompt fino a qualche migliaio di parole, testi più lunghi possono superare questo limite. In tali situazioni, potrebbe essere necessario suddividere l'input in parti più piccole e processarle separatamente [13];
- **Dipendenza dal training dataset**: una significativa limitazione dei LLM è la loro dipendenza dai dati con cui sono stati addestrati. Questi modelli apprendono dai vasti insiemi di dati utilizzati durante la fase di addestramento, ma non possono accedere a nuove informazioni o aggiornamenti in tempo reale successivamente. A causa di questo possono generare risposte obsolete o incomplete.

Per superare quest'ultima limitazione e integrare informazioni aggiornate, è possibile utilizzare tecniche come la Retrieval-Augmented Generation (RAG)$_G$.
RAG combina le capacità generative degli LLM con l'accesso a database esterni o motori di ricerca, permettendo di recuperare e includere informazioni più recenti e pertinenti nelle risposte generate [14]. Questo approccio migliora l'accuratezza e la rilevanza delle risposte, mantenendo l'efficienza e la coerenza nella generazione del linguaggio naturale.
Il funzionamento di un RAG prevede tre fasi:

- **Indexing**: prevede la raccolta di dati che verranno indicizzati. Essi verranno prelevati da fonti come documenti interni, codice, database aziendali, pagine web o articoli di ricerca. Una volta raccolti, i dati verranno ripuliti in modo tale

da rimuovere possibili inconsistenze e errori. Questo assicura che i dati siano in un formato coerente e facilmente indicizzabile. Successivamente verranno realizzate delle rappresentazioni vettoriali (Embedding$_G$) memorizzate all'interno di un Vector Database$_G$e utilizzate per costruire un indice che consente un recupero rapido ed efficiente delle informazioni.

- **Retrieval**: questa fase è fondamentale per ottenere informazioni provenirnti da fonti esterne e integrarle nel processo di generazione del testo. Inizia con la formulazione della Query$_G$, in cui l'input dell'utente viene analizzato e strutturato in un embedding, in modo tale da identificare le parole chiave e costruire una richiesta di informazioni.

  In seguito, tramite algoritmi come il K-Nearest Neighbour (KNN)$_G$, avverrà una ricerca di similarità tra l'embedding della query dell'utente e gli embedding dei dati presenti nel vector database. Questo processo permette di ottenere le informazioni che verranno utilizzate per generare la risposta finale.

- **Generation**: in questa fase le informazioni prelevate al momento del retrieval, insieme alla richiesta iniziale dell'utente, vengono integrate nel processo di generazione di una risposta coerente e significativa.

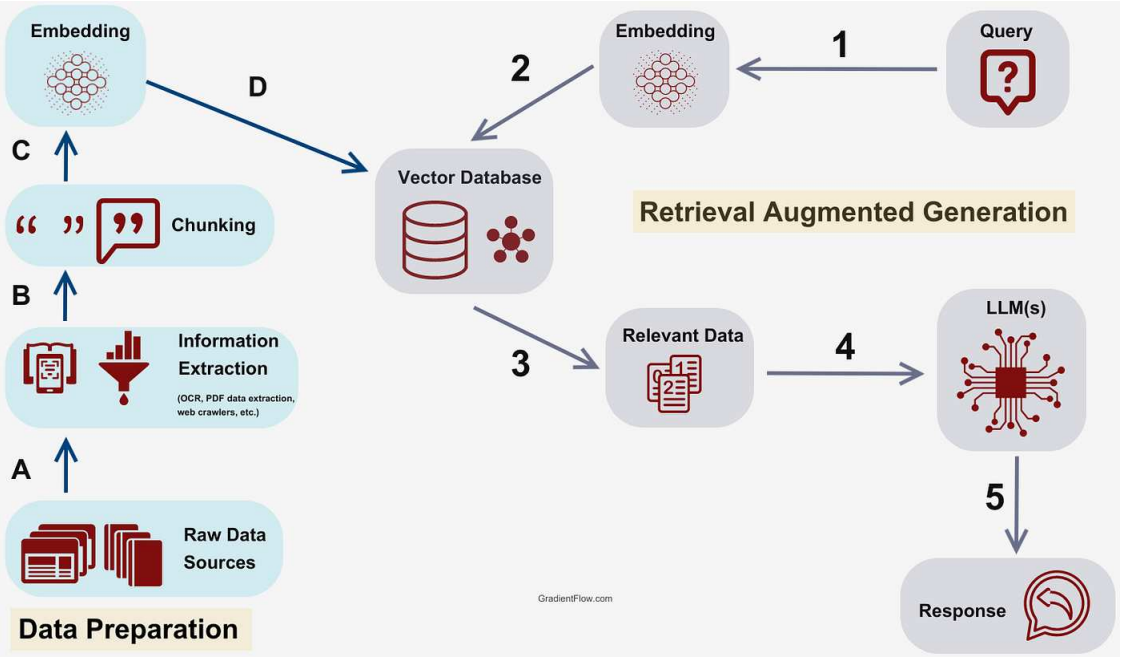


Figura 2.4: Struttura di un RAG

2.1.3 Dati Sintetici

I dati sintetici sono dati generati artificialmente che imitano le proprietà e le caratteristiche dei dati reali. Essi vengono generati a partire da algoritmi e modelli matematici e hanno lo scopo di simulare dati reali. Un esempio sono i dati numerici, testuali o immagini. L'utilizzo dei dati sintetici è molto vasto:

- In ambito di **sicurezza** permettono di preservare la privacy degli individui in quanto i dati reali spesso contengono infgormazioni sensibili che devono essere protette;
- Possono essere usati per addestrare modelli di **machine learning** quando i dati reali sono scarsi o difficili da ottenere;
- Sono **scalabili**, ovvero possono essere generati in grandi quantità e possono essere adattati per coprire casi specifici che potrebbero non essere presenti nei dati reali.

Si è riscontrato che una significativa quantità di dati sintetici può essere generata a partire dai Large Language Models [15].

2.1.4 Federated Learning

Il Federated Learning (FL)$_G$, è una metodologia di Machine Learning che si occupa di mantenere la privacy e la sicurezza dei dati usati nel processo di addestramento di un modello [16] . Anzichè centralizzare i dati in un unico server, ogni dispositivo addestra localmente una copia del modello utilizzando i propri dati. Successivamente, solo i parametri aggiornati del modello (come i pesi del modello) vengono inviati a un server centrale, che li aggrega per aggiornare il modello globale. Questo processo viene ripetuto iterativamente, con il modello aggiornato redistribuito ai dispositivi per ulteriori cicli di addestramento. I vari dispositivi dunque non condividono tra di loro dati che potrebbero essere sensibili [17].
Rispetto all'apprendimento centralizzato, questo approccio comporta anche nuove sfide, quali l'impostazione dell'infrastruttura di comunicazione tra i dispositivi, il coordinamento del processo di apprendimento, l'integrazione dei risultati delle parti

e la gestione dell'eterogeneità dei dati.

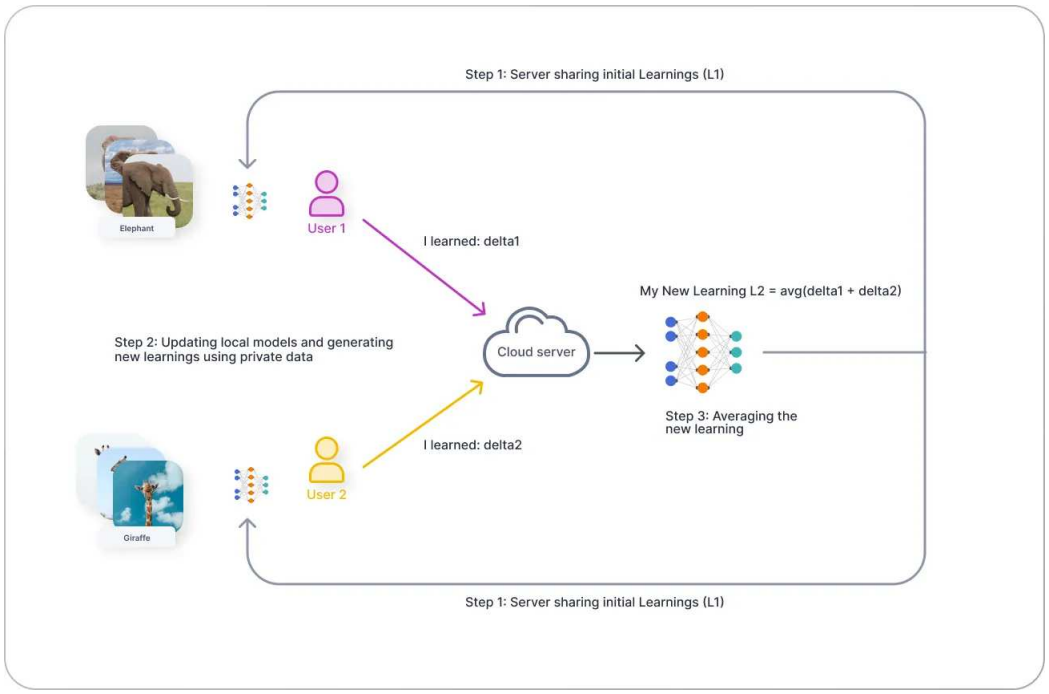


Figura 2.5: Schema sul Federated Learning

2.2 L'idea e gli obiettivi

Gli obiettivi dello stage sono stati: l'implementazione di una strategia per permettere agli LLM di generare test e test cases al fine di verificare codice sorgente, generazione di dati sintetici e studio di tecniche come il Federated Learning per garantire privacy e sicurezza relativa all'uso dei dati nel processo di addestramento di un modello.
Si farà riferimento alle tipologe di requisiti come segue:

- **OB**: indica un requisito obbligatorio.
- **DE**: indica un requisito desiderabile.
- **OP**: indica un requisito opzionale.

Le sigle precedentemente indicate saranno seguite da un numero, identificativo del requisito.

ID	Descrizione
ROB1	Configurare e implementare strategie di generazione di test tramite LLM utilizzando il codice sorgente.
ROB2	Integrare l'LLM con la documentazione del progetto per la generazione dei test.
ROB3	Implementare il Federated Learning per la creazione di dati sintetici.
ROB4	Valutare e confrontare i risultati ottenuti con dati sintetici rispetto a quelli reali.
ROB5	Implementare misure di sicurezza e protezione dei dati sintetici per garantire la privacy e la sicurezza.
RDE1	Estendere l'utilizzo del LLM per la generazione di test anche ad altri linguaggi di programmazione oltre a Java e JavaScript.
RDE2	Ottimizzare le strategie di generazione di test tramite LLM per migliorare l'efficienza e l'accuratezza.
RDE3	Esaminare approcci alternativi al Federated Learning per la creazione di dati sintetici e valutarne l'efficacia e l'efficienza.
RDE4	Condurre analisi approfondite sull'impatto delle misure di sicurezza implementate sui dati sintetici e sulla loro utilità per il processo di sviluppo del software.
ROP1	Estendere l'utilizzo del LLM per la generazione di test anche per testare la correttezza di query in SQL.

Tabella 2.1: Tabella dei requisiti

2.3 Pianificazione del Lavoro

Il periodo di svolgimento del progetto è stato dal 6 Maggio 2024 al 28 Giugno 2024, per un totale di 320 ore suddivise in 8 settimane lavorative full-time. La tabella seguente mostra le attività che sono state svolte ogni settimana e la loro durata:

Settimana	Durata	Descrizione attività
1	40 ore	ricerca di letteratura relativa all'impianto e lo studio delle tecnologie.
2	40 ore	elaborazione e l'implementazione concreta delle strategie per la generazione dei test tramite Large Language Models (LLM) al fine di verificare il codice sorgente.
3	40 ore	sviluppo di un'approccio integrato che permetta all'LLM di sfruttare anche la documentazione descrittiva del progetto per generare test.
4	40 ore	scrittura di documentazione e test del codice prodotto.
5	40 ore	studio relativo alle tecnologie e sistemi esistenti per la generazione di dati sintetici.
6	40 ore	implementazione del Federated Learning per la creazione di dati sintetici.
7	40 ore	analisi sulla privacy e sicurezza applicate all'uso dei dati sintetici generatis.
8	40 ore	scrittura di documentazione e test del codice prodotto.

Tabella 2.2: Tabella delle attività

2.4 Variazione degli Obiettivi Finali e della Pianifcazione

Alcuni obiettivi sono stati modificati, il che ha comportato una conseguente revisione della pianificazione delle attività lavorative. Vengono riportate rispettivamente le tabelle dei requisiti e della pianificazione del lavoro aggiornate.

I requisiti che sono stati modificati sono:

- **ROB3 e ROB5**: In considerazione della specificità delle richieste del progetto, si è deciso di suddividere l'implementazione del sistema di generazione di dati sintetici e l'implementazione del Federated Learning. Questa scelta è stata dettata dall'esigenza di garantire un approccio mirato e approfondito per ciascuna risoluzione.

- **RDE2, RDE3, ROP1**: sono stati rimossi perchè ritenuti non necessari.

ID	Descrizione
ROB1	Configurare e implementare strategie di generazione di test tramite LLM utilizzando il codice sorgente.
ROB2	Integrare l'LLM con la documentazione del progetto per la generazione dei test.
ROB3	Implementare una soluzione per la creazione di dati sintetici attraverso LLM.
ROB4	Valutare e confrontare i risultati ottenuti con dati sintetici rispetto a quelli reali.
ROB5	Implementare una versione semplificata di Federated Learning.
RDE1	Estendere l'utilizzo dell'LLM per la generazione di test anche ad altri linguaggi di programmazione oltre a Java e JavaScript.
RDE2	Condurre analisi approfondite sul sistema di sicurezza e privacy dei dati che vengono utilizzati nel processo di addestramento di un modello nel Federated Learning.

Tabella 2.3: Tabella aggiornata dei requisiti

Settimana	Durata	Descrizione attività
1	40 ore	ricerca di letteratura relativa all'impianto e lo studio delle tecnologie.
2	40 ore	elaborazione e implementazione concreta delle strategie per la generazione dei test di unità tramite Large Language Models (LLM).
3	40 ore	sviluppo di un'approccio integrato che permetta all'LLM di sfruttare anche la documentazione descrittiva del progetto per generare test.
4	40 ore	scrittura di documentazione e test del codice prodotto.
5	40 ore	studio e implementazione relativa alle tecnologie e sistemi esistenti per la generazione di dati sintetici.
6	40 ore	implementazione di una versione semplificata di Federated Learning.
7	40 ore	analisi sulla privacy e sicurezza applicate all'uso del Federated Learning
8	40 ore	scrittura di documentazione e test del codice prodotto.

Tabella 2.4: Tabella aggiornata delle attività

2.5 Strumenti Utilizzati

Di seguito verranno elencati i vari strumenti utilizzati durante lo svolgimento del progetto.

2.5.1 Google Docs

Google Docs è una piattaforma web fornita da Google, che offre agli utenti la possibilità di creare, modificare e condividere una vasta gamma di documenti, inclusi documenti di testo, fogli di calcolo e presentazioni. Questo strumento è particolarmente apprezzato per la sua facilità d'uso e per le numerose funzionalità che offre.

Figura 2.6: Logo Google Docs

2.5.2 Google Scholar

Google Scholar è un servizio gratuito fornito da Google che offre un'ampia collezione di articoli accademici, tesi, libri, white papers e altre risorse accademiche che trattano discipline. È progettato per aiutare gli utenti a trovare letteratura accademica in modo rapido e semplice, facilitando la ricerca di informazioni su argomenti specifici.

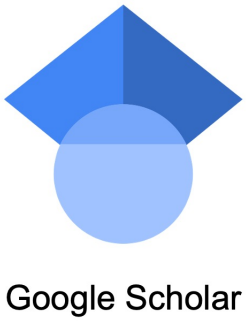


Figura 2.7: Logo Google Scholar

2.5.3 Keynote

Keynote è sviluppato da Apple Inc., disponibile per il sistema operativo macOS e per iOS. È progettato per consentire agli utenti di creare presentazioni di alta qualità, utilizzando una varietà di strumenti e funzionalità.

Figura 2.8: Logo Keynote

2.5.4 Overleaf

Overleaf è una piattaforma di scrittura collaborativa basata su cloud per la creazione di documenti in LATEX, è ampiamente utilizzato nella comunità accademica

per la produzione collaborativa di articoli, tesi e altri documenti scientifici. Fornisce strumenti per la gestione di progetti, revisione del testo e integrazione di formule matematiche complesse. Overleaf semplifica il processo di scrittura e formattazione, eliminando la necessità di installare e gestire un ambiente LATEX localmente.

Figura 2.9: Logo Overleaf

2.5.5 PyCharm

PyCharm è un ambiente di sviluppo integrato (IDE) specificamente progettato per sviluppatori che lavorano con il linguaggio di programmazione Python. È sviluppato da JetBrains e offre una vasta gamma di funzionalità e strumenti per semplificare il processo di sviluppo del software in Python.
Durante il tirocinio è stato utilizzato per l'implementazione del PoC.

Figura 2.10: Logo PyCharm

2.5.6 WebStorm

WebStorm è un ambiente di sviluppo integrato (IDE) sviluppato da JetBrains, progettato per lo sviluppo di applicazioni web. Supporta linguaggi come JavaScript, TypeScript, HTML e CSS.
Durante il tirocinio è stato utilizzato per l'implementazione del PoC.

Figura 2.11: Logo WebStorm

2.5.7 GitHub

GitHub è una piattaforma di hosting basata su Git, un sistema di controllo delle versioni distribuito. Fondata nel 2008, GitHub offre agli sviluppatori un ambente collaborativo per la gestione dei progetti software, consentendo loro di condividere, collaborare e contribuire al codice sorgente in modo efficiente.

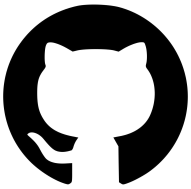


Figura 2.12: Logo GitHub

2.5.8 LangChain

LangChain è un framework open-source per lo sviluppo di applicazioni che utilizzano LLM. Offre una serie di componenti e strumenti che facilitano l'integrazione, l'interazione e la gestione dei modelli all'interno di applicazioni complesse. LangChain permette la creazione di pipeline di elaborazione, consentendo di combinare diversi modelli e tecniche di pre e post-processing per ottenere risultati ottimali. Inoltre supporta l'integrazione con vari servizi di NLP e API di terze parti, permettendo di sfruttare al massimo le capacità degli LLM.

Figura 2.13: Logo di LangChain

2.5.9 Ollama

Ollama è un'applicazione desktop progettata per eseguire localmente Large Language Models (LLM) in un ambiente sicuro e controllato. Utilizzando il proprio motore di hosting, Ollama permette l'esecuzione dei LLM direttamente sul dispositivo dell'utente, eliminando la necessità di connessione a server remoti.
Ollama supporta una varietà di modelli pre-addestrati e consente l'importazione di modelli personalizzati, offrendo flessibilità nell'uso dei LLM.

Figura 2.14: Logo di Ollama

2.5.10 ChromaDB

ChromaDB è un database specializzato nella gestione di dati rappresentati attraverso embeddings, una tecnica utilizzata per trasformare dati complessi (come testi, immagini, audio) in vettori numerici. ChromaDB può eseguire ricerche semantiche efficaci. Questo significa che, invece di cercare corrispondenze esatte di parole chiave, il database può trovare contenuti rilevanti basandosi su similarità concettuali tra i dati.

Figura 2.15: Logo di ChromaDB

Capitolo 3

Soluzioni Esistenti

Il progetto in esame include attività distinte che richiedono approcci di risoluzione separati: la generazione di test a partire dal codice sorgente utilizzando Large Language Model (LLM), la creazione di dati sintetici, l'implementazione di una versione semplificata del Federated Learning. Data la natura eterogenea di queste attività, verranno presentate, nei paragrafi seguenti, soluzioni esistenti che affrontano ciascuno di questi problemi in modo specifico.

3.1 Generazione di test tramite LLM

La letteratura riguardante l'utilizzo dei Large Language Models (LLM) per la generazione di test e test cases a partire dal codice sorgente è ricca di studi che analizzano diverse modalità di implementazione.
Modelli come GPT-3, PaLM, Codex e LLaMA, che superano le centinaia di miliardi di parametri, sono stati scelti per la loro efficienza e capacità di apprendimento contestuale, dove il modello impara a partire da esempi e contesti rilevanti al task. Modelli più piccoli come BERT invece non hanno ottenuto gli stessi risultati. Studi hanno notato che la principale sfida non risiede nella generazione del codice di test, ma nella determinazione degli input da fornire all'LLM per ottenere una maggiore copertura del sistema sotto test. Una soluzione al problema si è rivelata essere la progettazione di prompt efficaci per la generazione di test cases, Xie et al [7] si concentrano sulla creazione di prompt che permettano all'LLM di comprendere al meglio il contesto del test da generare. Vikram et al. [7] esplorano invece l'uso degli LLM per generare

test utilizzando la documentazione del codice. Interessante è l'idea TestGen-LLM di Meta [18], che utilizza i Large Language Models (LLM) per migliorare il codice dei test scritti dagli sviluppatori. TestGen-LLM assicura che i test superino una serie di requisiti per garantire un miglioramento effettivo rispetto alla suite di test originale, eliminando problemi dovuti alle allucinazioni degli LLM.
Sono state anche valutate le capacità di ChatGPT nella generazione di test di unità [19]. Si è riscontrato che i test che genera presentano ancora problemi di correttezza, inclusi errori di compilazione e fallimenti di esecuzione, principalmente causati da asserzioni errate. Tuttavia, i test che passano, somigliano ai test scritti manualmente, ottenendo copertura e a volte la preferenza degli sviluppatori. La generazione di test basata su ChatGPT risulta essere molto promettente se si migliora la correttezza dei test generati.
Sempre considerando l'utilizzo di ChatGPT si menziona il tool ChatUniTest [20]. Basandosi su di un framework di generazione, validazione e riparazione si propone di generare suite di test, e nel caso in cui queste risultassero incorrette, attraversa un porcesso di correzione.

3.2 Creazione di dati sintetici con LLM

La generazione di dati sintetici mediante l'uso di Large Language Models (LLM) è un campo di ricerca emergente e promettente. La letteratura attuale evidenzia un crescente interesse per l'uso degli LLM per creare dati sintetici che possono simulare le proprietà dei dati reali [21]. Questa tecnica è particolarmente utile in contesti in cui i dati reali sono scarsi, costosi da ottenere, o contengono informazioni sensibili che necessitano di protezione.

Un esempio significativo in questo ambito è rappresentato da DataDreamer [22], una libreria scritta in Python, open source, usata oltre che per attività di prompting e training anche per la generazione di dati sintetici. Si è dimostrato che i dati sintetici generati dagli LLM possono essere particolarmente utili per l'addestramento dei modelli di classificazione.
Interessante è la soluzione propasta da Gretel.ai [23] , un'azienda che offre una piattaforma per la creazione di dati sintetici. La piattaforma utilizza tecniche avanzate di

Machine Learning (ML) per generare dati sintetici che imitano i dati reali, permettendo di addestrare modelli senza compromettere la privacy o la sicurezza dei dati. Gretel.ai può essere utilizzata per creare dati tabellari, immagini o persino interi database.
Un esempio che non utilizza Large Language Models (LLM) per generare i dati ma algoritmi di randomizzazione e una vasta libreria di valori predefiniti è Mockaroo, uno strumento per generare dati sintetici realistici e personalizzabili, esportabili in vari formati (CSV, JSON, SQL, Excel).
Una novità è rappresentata da Nemotron-4 340B sviluppato da NVIDIA [24], una famiglia di modelli che gli sviluppatori possono usare per generare dati sintetici allo scopo di addestrare Large Language Models (LLM). Nemotron-4 trova applicazione in vari ambiti industriali, quali quello finanziario, manifatturiero e della sanità. Nello specifico viene utilizzato Nemotron-4 Instruct Model per generare i dati sintetici a partire da una query specifica data in input dall'utente, successivamente Nemotron-4 Reward Model agisce allo scopo di filtrare i dati generati e selezionare solo quelli che vengono ritenuti rilevanti e che rispettano specifici requisiti.

3.3 Implementazione del Federated Learning

Il Federated Learning (FL) segna un notevole progresso nel campo dell'intelligenza artificiale e nella protezione della privacy dei dati. Come introdotto precedentemente nel capitolo di descrizione del progetto, il FL consente l'addestramento dei modelli di Machine Learning (ML) direttamente sui dispositivi degli utenti, senza la necessità di trasferire i datasets ai server centrali. Questo metodo risponde alle crescenti preoccupazioni riguardo alla privacy e alla sicurezza dei dati, assicurando che le informazioni personali restino localizzate sui dispositivi degli utenti. Un esempio per la sperimentazione con il Federated Learning (FL) è TensorFlow Federated (TFF), sviluppato da TensorFlow, un framework open-source progettato per elaborazioni su dati decentralizzati. TFF permette l'addestramento di un modello globale condiviso attraverso numerosi client partecipanti che mantengono localmente i loro training dataset.
Altro esempio è NVIDIA FLARE™ (NVIDIA Federated Learning Application Run-

time Environment) [25], che rappresenta un SDK open-source per il Federated Learning il cui obiettivo è rendere più facile agli sviluppatori l'uso di FL nella loro ricerca e nelle applicazioni reali. Uno strumento open-source utilizzato per l'addestramento di modelli tramite Federated Learning (FL) è rappresentato dal framework Open Federated Learning (OpenFL)[26]. Basato su Python, OpenFL è caratterizzato da pipeline di addestramento costruite sia tramite TensorFlow che con PyTorch, e può essere facilmente esteso ad altri framework di Machine Learning (ML) e Deep Learning (DL).

Capitolo 4

Coding Assistant

Il capitolo seguente illustra la soluzione progettuale relativa all'utilizzo dei Large Language Models (LLM) per la generazione dei test di unità e dei dati sintetici.

4.1 Proposta di Soluzione

L'idea alla base della soluzione sviluppata consiste nella realizzazione di un'applicazione web minimale, progettata per fungere da supporto e assistenza agli sviluppatori in modo efficiente e intuitivo. L'applicazione offre due principali funzionalità: la generazione di test di unità per la verifica degli script e la generazione di dati sintetici per vari usi.

Per la generazione dei test di unità, l'utente deve inserire sia la documentazione pertinente che gli script da verificare. Questo processo permette all'applicazione di comprendere il contesto e le specifiche del codice, generando test di unità accurati e rilevanti che aiutano a validare il comportamento degli script.

Per quanto riguarda la generazione di dati sintetici, l'utente deve fornire specifiche dettagliate che descrivano le caratteristiche desiderate dei dati. Questo permette di creare dataset personalizzati che rispondano esattamente alle esigenze dell'utente, facilitando attività come test, simulazioni e analisi senza dover ricorrere a dati reali.

Questa soluzione, attraverso la combinazione di test di unità automatizzati e la generazione di dati sintetici, mira a migliorare significativamente l'efficienza e la qualità del processo di sviluppo software, offrendo un prezioso strumento di supporto agli sviluppatori.

4.2 Progettazione Architetturale

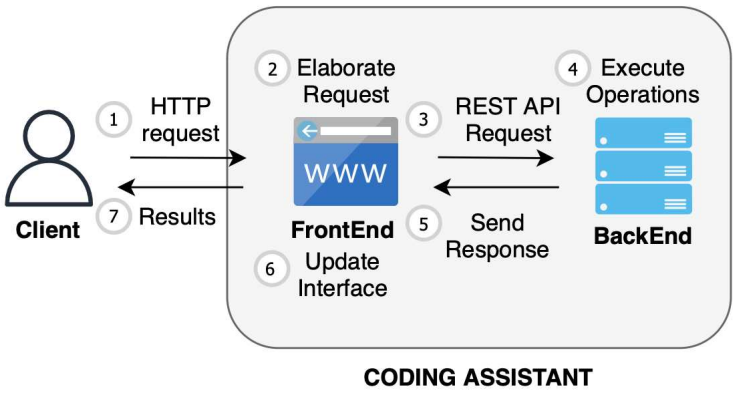


Figura 4.1: Struttura di Coding Assistant

Per la progettazione architetturale dell'applicazione, è stato applicato il principio architetturale della Separation of Concerns (SoC)$_G$. Questo principio consiste nella suddivisione di un sistema software in parti distinte, ognuna delle quali gestisce una specifica funzionalità.
In particolare, l'architettura dell'applicazione è stata suddivisa in due principali componenti: il backend e il frontend.

- **Front-end**: il frontend si occupa dell'interfaccia utente e delle interazioni client-side. È responsabile della presentazione dei dati, dell'acquisizione degli input degli utenti e della comunicazione con il backend.
- **Back-end**: il backend è responsabile della logica di business. Esso gestisce le richieste degli utenti, elabora i dati e permette la comunicazione con il frontend.

Nelle sezioni successive verranno spiegate in modo più dettagliato e completo le scelte implementative adottate per ciascuna di queste componenti. Verranno illustrati i framework e le tecnologie utilizzate e le specifiche tecniche che hanno guidato lo sviluppo dell'applicazione.

4.2.1 Front-end

Per quanto riguarda la parte frontend, gli strumenti utilizzati sono:

- **Tailwind CSS**: un framework CSS utility-first che fornisce una vasta gamma di classi predefinite, consentendo uno stile rapido ed efficiente senza la necessità di scrivere CSS personalizzato. Questo approccio facilita la creazione di interfacce utente coerenti e reattive, migliorando la produttività degli sviluppatori e mantenendo il codice pulito e manutenibile;
- **TSX**: un'estensione di TypeScript specifica per l'uso con React con cui sono state realizzate le pagine dell'applicazione. L'uso di TSX permette di combinare markup HTML e logica TypeScript, fornendo una potente tipizzazione statica e migliorando l'affidabilità del codice. Questo approccio consente di costruire componenti riutilizzabili e modulari, rendendo lo sviluppo del frontend più strutturato e scalabile;
- **Next.js**: è un framework open-source basato su React, sviluppato e mantenuto da Vercel. È progettato per facilitare lo sviluppo di applicazioni web con funzionalità avanzate;
- **Zustand**: libreria di React utilizzata per gestire lo stato dell'applicazione, esso è centralizzato, quindi non viene disperso nei vari componenti;
- **Uppy**: libreria di JavaScript per la gestione del caricamento dei file.

In questa sezione, mostrando degli snippet di codice, verranno illustrate in dettaglio le scelte di design e implementazione adottate.

Organizzazione dei file

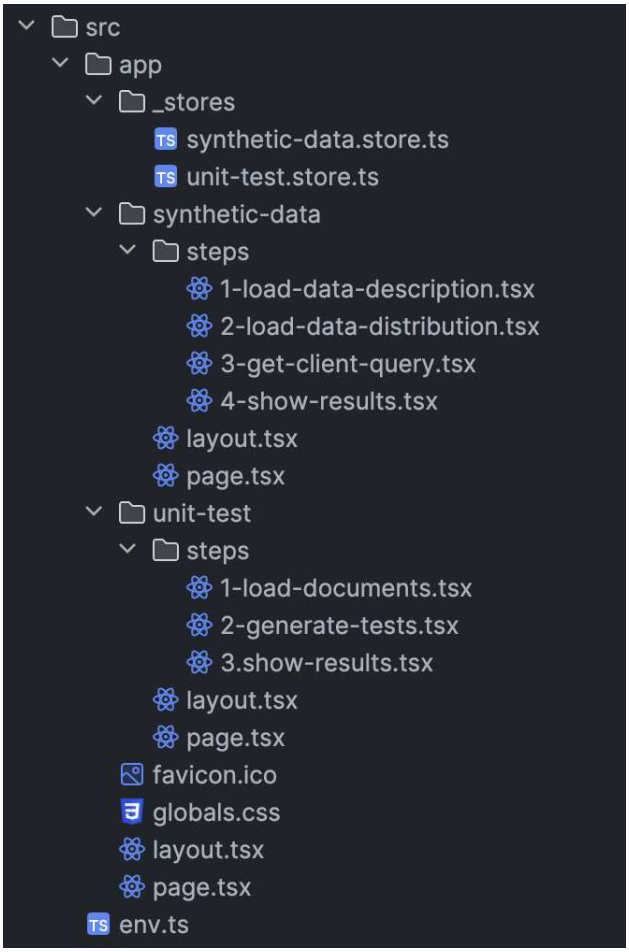


Figura 4.2: Organizzazione dei file

Nell'organizzazione del progetto all'interno della cartella 'src', è stata adottata una suddivisione chiara per le funzionalità di generazione di test di unità e di dati sintetici. Si è deciso di gestire il caricamento di file da parte dell'utente tramite l'implementazione di diversi step per renderlo più ordinato e gestibile. La divisione delle pagine all'interno di queste cartelle facilita la navigazione e la manutenzione del codice, permettendo una chiara separazione delle responsabilità e migliorando la leggibilità complessiva del progetto.

Layout delle pagine

```tsx
import type { Metadata } from "next";

export const metadata: Metadata = {  no usages
  title: "LLM Project",
  description: "LLM Project University",
};

export default function SyntheticDataLayout({  no usages
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <div
      className={"flex w-full h-full justify-center items-center"}
      style={{ height: "100vh" }}
    >
      {children}
    </div>
  );
}
```

Figura 4.3: Codice di Layout delle pagine

Questa porzione di codice mostra la creazione del componente **SyntheticDataLayout**, fondamentale per la presentazione uniforme delle pagine all'interno dell'applicazione, in questo caso della pagine relative alla funzionalità di generazione dei dati sintetici. Questo facilita la gestione e il controllo del layout senza dover ripetere il codice CSS e gli stili di posizionamento in ogni componente specifico della pagina.

Gestione dello state

```ts
import { create } from "zustand";

export enum SyntheticDataStep {  Show usages
  LoadDataDescription = 0,
  LoadDataDistribution = 1,
  GetClientQuery = 2,
  ShowResults = 3,
}

type SyntheticDataStore = {
  step: SyntheticDataStep;
  setStep: (step: SyntheticDataStep) => void;
  response?: string;
  setResponse: (response?: string) => void;
  clientQuery?: string;
  setClientQuery: (clientQuery?: string) => void;
  isLoading: boolean;
  setIsLoading: (isLoading: boolean) => void;
};

export const useSyntheticDataStore : UseBoundStore<StoreApi<Synthet... = create<SyntheticDataStore>((set) : {...} => ({
  step: SyntheticDataStep.LoadDataDescription,
  response: undefined,
  clientQuery: "",
  isLoading: false,
  setStep: (step : SyntheticDataStep ) => set({ step }),
  setResponse: (response : string | undefined ) => set({ response }),
  setClientQuery: (clientQuery : string | undefined ) => set({ clientQuery }),
  setIsLoading: (isLoading : boolean ) => set({ isLoading }),
}));
```

Figura 4.4: State per il generatore di dati sintetici

Questa porzione del codice definisce un enum **SyntheticDataStep**, che rappresenta i vari passaggi del processo di gestione dei dati sintetici. Questo enum viene utilizzato per tenere traccia del progresso attraverso diverse fasi. Successivamente viene definito il tipo **SyntheticDataStore** che descrive la struttura dello stato dell'applicazione, e sarà gestito tramite Zustand. Infine viene defino il processo di creazione dello Store

Codice della pagina di uno step (1)

```
type Step2LoadDataDistributionProps = {};

export const Step2LoadDataDistribution = (  Show usages
  props: Step2LoadDataDistributionProps,
) => {
  const { setStep } = useSyntheticDataStore();

  const [uppy : Uppy<Record<string, ?>, Record... ] = useState( initialState: () =>
    new Uppy( opts: {
      locale: Italian,
      // debug: true,
      id: "Step2LoadDataDistribution",
      allowMultipleUploadBatches: true,
      restrictions: {
        allowedFileTypes: ["text/plain"],
      },
    }).use(XHRUpload,  opts: {
      bundle: true,
      endpoint: `${env.NEXT_PUBLIC_BACKEND_API_URL}/synthetic-data/load-data-distribution`,
      fieldName: "files",
      formData: true,
    }),
  );

  uppy.on( event: "complete",  callback: () : void  => {
    setStep(SyntheticDataStep.GetClientQuery);
  });
```

Figura 4.5: Codice della pagina di uno step (1)

Questa prima porzione di codice definisce il componente **Step2LoadDataDistribution**, che rappresenta la seconda fase di un processo per caricare i file che contengono le distribuzioni dei dati. Si nota che oltre a definire i vari parametri di configurazione si utilizza anche il plugin XHRUpload, che specifica l'endpoint dell'API di backend dove i file caricati verranno inviati.

Codice della pagina di uno step (2)

```
return (
  <>
    <h1 className="text-white font-bold text-2xl">STEP 2</h1>
    <h2 className="text-xl mb-12 font-base text-zinc-300">
      Carica i file che riportano le distribuzioni dei dati qui sotto
    </h2>
    <div className="flex flex-wrap p-10">
      <Dashboard
        uppy={uppy}
        height={450}
        theme="dark"
        proudlyDisplayPoweredByUppy={false}
      ></Dashboard>
    </div>
    <div className="flex flex-col justify-center items-center flex-wrap p-10 gap-6">
      <h1 className="font-semibold">
        Hai già caricato i file indicanti le distribuzioni?
      </h1>
      <button
        className="bg-zinc-50 text-black px-4 py-2 rounded-md font-semibold mt-4"
        onClick={() => setStep(SyntheticDataStep.GetClientQuery)}
      >
        Vai al prossimo step
      </button>
    </div>
  </>
);
};
```

Figura 4.6: Codice della pagina di uno step (2)

Questo blocco di codice rappresenta il render del componente **Step2LoadDataDistribution**, che è responsabile di guidare l'utente attraverso il secondo passo del processo di caricamento dei file di distribuzione dei dati.

Codice della pagina di generazione dei dati sintetici

```
import {
  SyntheticDataStep,
  useSyntheticDataStore,
} from "@/app/_stores/synthetic-data.store";

export default function SyntheticDataGeneratorPage() : React.JSX.Element  {  no usages
  const { step : SyntheticDataStep  } = useSyntheticDataStore();

  const renderStep = () : Element  => {  Show usages
    switch (step) {
      case SyntheticDataStep.LoadDataDescription:
        return <Step1LoadDataDescription />;
      case SyntheticDataStep.LoadDataDistribution:
        return <Step2LoadDataDistribution />;
      case SyntheticDataStep.GetClientQuery:
        return <Step3GetClientQuery />;
      case SyntheticDataStep.ShowResults:
        return <Step4ShowResults />;
    }
  };
```

Figura 4.7: Codice del componente SyntheticDataGeneratorPage

Questo codice definisce il componente **SyntheticDataGeneratorPage**, che gestisce un flusso sequenziale di passi all'interno di una pagina per generare dati sintetici. Nello specifico la funzione **renderStep()** del componente utilizza uno statement switch basato sullo stato step per determinare quale componente renderizzare in base al passo corrente del flusso di lavoro.

4.2.2 Back-end

Per la parte backend gli strumenti utilizzati sono:

- **Python**: linguaggio di programmazione principale, grazie alla sua versatilità e all'ampia disponibilità di librerie per l'elaborazione del linguaggio naturale e il Machine Learning;
- **Llama3**: è stato scelto come modello di linguaggio, sviluppato da Meta. Questo modello è stato scaricato localmente tramite Ollama, optando per la versione con 8 miliardi di parametri, ritenuta adeguata per le esigenze dell'implementazione prevista;
- **ChromaDB**: database vettoriale, scelto per la sua capacità di gestire efficientemente i dati vettoriali e supportare le funzionalità avanzate del sistema;
- **Uvicorn**: un server ASGI (Asynchronous Server Gateway Interface) veloce ed efficiente che viene utilizzato per eseguire l'applicazione, garantendo alte prestazioni e supporto per richieste asincrone;
- **FastAPI**: framework web scelto per costruire l'interfaccia del backend grazie alla sua semplicità d'uso e alla capacità di gestire richieste HTTP in modo asincrono.

In seguito, verranno descritte in dettaglio le soluzioni implementate per la generazione dei test di unità e dei dati sintetici, evidenziando le caratteristiche specifiche di ciascuna.

4.2.2.1 Generazione di test tramite Large Language Model (LLM)

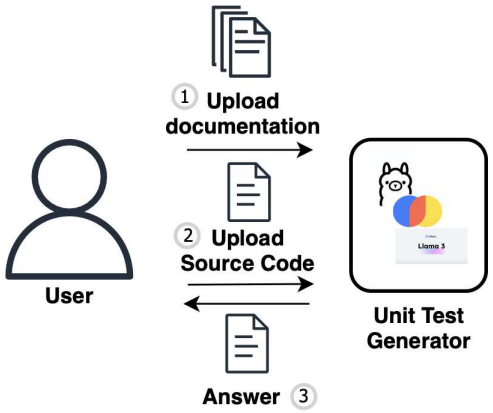


Figura 4.8: Struttura del Generatore di Test di Unità

Lo scenario principale riguarda un team di sviluppo impegnato nell'elaborazione di una suite di test con l'obiettivo di valutare la correttezza del codice prodotto. Questo processo è essenziale per garantire che il software funzioni come previsto, sia privo di errori e rispetti i requisiti specificati. L'attore coinvolto nel sistema in esame è quindi il team di sviluppatori, il cui obiettivo è minimizzare il tempo necessario per l'elaborazione della suite di test in questione.
L'interazione dell'attore con il sistema si sviluppa nel modo seguente:

1. Il team seleziona lo script di cui vuole generare i test;
2. Il team seleziona anche la documentazione relativa a tale codice;
3. Il sistema produce un file in formato markdown, che descrive sia come procedere nella stesura del codice dei test sia i test veri e propri;
4. Il team può decidere se implementare i test appena generati oppure scrivere i propri, seguendo le linee guida generate dal sistema.

In seguito viene mostrata più in dettaglio l'implementazione pensata:

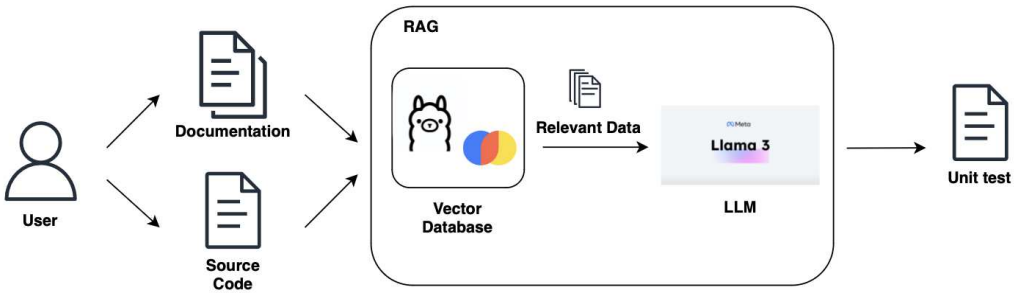


Figura 4.9: Struttura dettagliata del Generatore di Test di Unità

Come soluzione si è deciso di creare un sistema che sfrutta una verisone semplificata di un Retrieval-Augmented Generation (RAG).
Questo approccio richiede un database vettoriale in grado di memorizzare e gestire efficientemente i documenti, generare embeddings e permettere ricerche di similarità per la generazione della risposta da restituire all'utente. Come anticipato precedentemente, ChromaDB è stato scelto come database vettoriale, e nello specifico è stato utilizzato per le seguenti attività:

1. **Memorizzazione della documentazione del codice sorgente**: La documentazione relativa al codice sorgente per il quale si devono generare i test viene caricato e vengono generati i relativi embeddings;

2. **Recupero Semantico delle Informazioni**: Gli embeddings vengono utilizzati nel processo di ricerca semantica, per trovare documenti simili o rilevanti. Questo è essenziale al momento della generazione dei test.

Le informazioni recuperate al termine della fase di information retrieval sono relative alla documentazione utile del codice sorgente. Queste informazioni vengono processate da Llama 3 per generare dei riassunti da inserire successivamente all'interno del prompt.

In seguito viene mostrata in dettaglio la struttura del prompt:

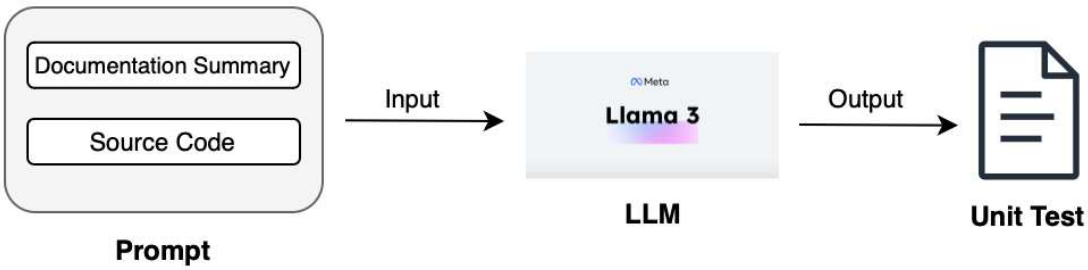


Figura 4.10: Struttura dettagliata del Prompt

Dopo aver implementato il vector database con ChromaDB, il passo successivo è stato garantire che la generazione automatica dei test avvenisse in modo completo e corretto. Per raggiungere questo obiettivo, è stato necessario studiare e costruire con attenzione il prompt da fornire al Large Language Model (LLM) utilizzato nel progetto.
Il prompt deve includere informazioni dettagliate e specifiche. Le componenti chiave identificate sono state:

1. **Codice Sorgente**: Tutte le porzioni di codice di cui è richiesto generare un test;
2. **Informazioni dalla Documentazione**: Ulteriori informazioni derivanti dalla documentazione che possono aiutare a generare test più corretti e completi.

4.2.2.2 Generazione di dati sintetici tramite Large Language Model (LLM)

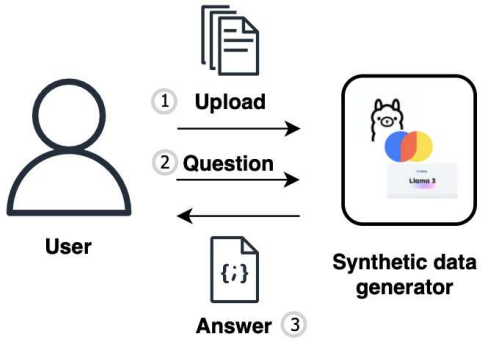


Figura 4.11: Struttura del Generatore di Dati Sintetici

Lo scenario coinvolge un team di sviluppatori impegnato in varie attività, come il testing e la validazione di software, il popolamento di database e l'addestramento di modelli di machine learning. Per eseguire queste attività in modo efficiente, è essenziale che il team disponga di un grande volume di dati.
L'interazione dell'attore principale, il team di sviluppatori, con il sistema avviene nel seguente modo:

1. Il team seleziona i file di testo che descrivono i soggetti o gli elementi di interesse per generare i dati;

2. Il team specifica le distribuzioni o le caratteristiche desiderate per gli elementi che saranno parte dei dati da generare;

3. Il team inserisce le specifiche per la richiesta, come il tipo di dati che si desiderano;

4. Il sistema produce un file in formato JSON che elenca i dati richiesti dall'utente.

Questo processo aiuta il team a ottimizzare l'esecuzione delle proprie attività, fornendo rapidamente dati sintetici conformi alle specifiche richieste, utili per il testing, il popolamento dei database o l'addestramento dei modelli di machine learning.

In seguito viene mostrata più in dettaglio l'implementazione pensata:

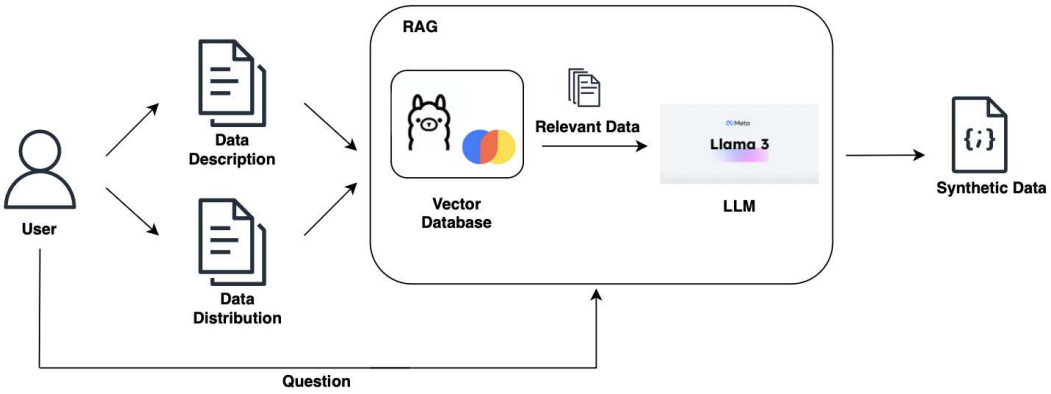


Figura 4.12: Struttura dettagliata del Generatore di Dati Sintetici

Come nella soluzione precedente, anche in questo caso si è deciso di implementare una verisone semplificata di un Retrieval-Augmented Generation (RAG).
E' stato utilizzato un database vettoriale in grado di memorizzare e gestire i documenti, generare embeddings e permettere ricerche di similarità per la generazione della risposta da restituire all'utente. Si ricade nuovamente nell'utilizzo di ChromaDB, che nello specifico è stato utilizzato per le seguenti attività:

1. **Memorizzazione dei Documenti Descrittivi**: La documentazione che descrive caratteristiche degli elementi dei dtai da generare;

2. **Memorizzazione dei Documenti relativi alle Distribuzioni**: La documentazione che descrive le distribuzioni specifiche che dovranno avere alcuni elementi dei dati da generare;

3. **Recupero delle Informazioni**: La richiesta dell'utente viene utilizzata come query per il database vettoriale, in questo modo verranno prelevati i documenti che semanticamnete hanno maggiore affinità con i dati che devono essere generati.

In seguito viene mostrata in dettaglio la struttura del prompt:

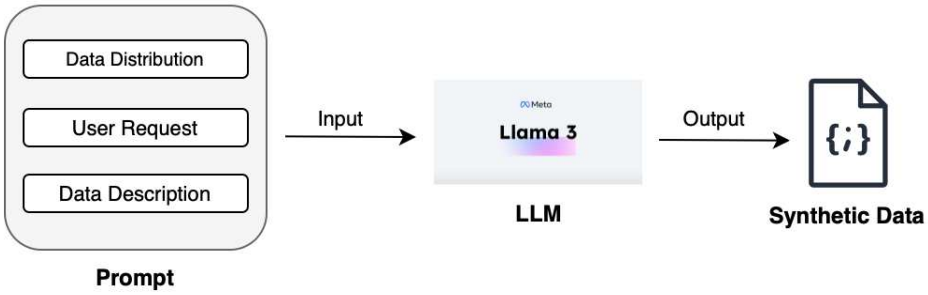


Figura 4.13: Struttura del Prompt

Per garantire una generazione soddisfacente dei dati, è essenziale che il prompt fornito come input al Large Language Model (LLM) sia strutturato correttamente. In particolare, il prompt deve includere:

1. **Informazioni sulla Distribuzione**: tutte le informazioni relative alla distribuzione dei dati che devono essere generati;
2. **Informazioni dalla Documentazione**: tutte le informazioni relative ai dati che devono essere generati, queste sono ricavate come descritto in fase di information retrival;
3. **User Request**: la richiesta fornita dall'utente.

Di seguito vengono mostrati ed analizzati alcuni code snippets relativi alla gestione del codice del backend. Il codice è relativo alla funzionalità di generazione dei dati sintetici, la generazione di test è similare.

Caricamento dei documenti nel RAG

```python
async def load_documents_synthetic_data_to_vectordb(files: list[UploadFile]):
    delete_collection_chromadb(name="docs_synthetic_data")

    # Initialize the ChromaDB collection
    docs_collection = init_collection_chromadb(name="docs_synthetic_data")

    # Load the documents
    documents = await load_files_from_uploadfile(files)
    print("Loaded documents: " + str(len(documents)))

    # Create an Ollama embeddings object
    embeddings = OllamaEmbeddings(model="llama3")

    # Store the documents in the collection
    docs_collection = store_documents(documents=documents, collection=docs_collection, embeddings=embeddings)
```

Figura 4.14: Caricamento dei documenti nel RAG

Nello snippet di codice viene illustrata la funzione per il caricamento dei documenti all'interno di un vector database. In primo luogo, viene inizializzata un'istanza del vector database utilizzando ChromaDB. Successivamente, una funzione apposita viene invocata per caricare la lista dei documenti di interesse. Infine, i documenti vengono memorizzati nel vector database sfruttando gli embeddings di Ollama.

Generazione dei dati

```python
async def generate_data_synthetic_data_to_vectordb(query_string: str):

    docs_collection = get_collection_chromadb(name="docs_synthetic_data")
    embeddings = OllamaEmbeddings(model="llama3")
    prompt = f"Retrieve all the information about the structure and the distribution of the data found in this request: {query_string}"
    embedding_query = embeddings.embed_query(prompt)
    results = docs_collection.query(
        query_embeddings=[embedding_query],
        n_results=5
    )
    data = results['documents']
    distribution_elements = ollama.generate(model="llama3",
                                            prompt=f"From {data} get all the distribution of the data related to: {query_string}")
    distribution = distribution_elements["response"]
    description_elements = ollama.generate(model="llama3",
                                           prompt=f"From {data} get all the descriptions of the data related to: {query_string}")
    description = description_elements["response"]

    output = ollama.generate(model="llama3",
                             options={"temperature": 0.0, "max_tokens": 12000},
                             prompt=f"You are an experienced developer. Generate result in JSON format."
                                    f"Generate at least ten JSON objects that represent this request: {query_string}"
                                    f"only use the following data descriptions: {description},"
                                    f"and the distributions {distribution}. Don't write something like:"
                                    f"Here are ten JSON objects that represent a customer profile with its purchase "
                                    f"history in JSON format Write only valid JSON format. "
                                    f"Don't truncate JSON objects. Write all objexts, don't skip any. ")
    return output
```

Figura 4.15: Codice di generazione dei dati sintetici

Il codice presentato definisce una funzione asincrona che genera dati sintetici basati su una query dell'utente. In dettaglio, la funzione esegue le seguenti operazioni: recupera i primi cinque documenti semanticamente affini alla query dal vector database, utilizzando un modello di embedding avanzato. Successivamente, questi documenti vengono ulteriormente filtrati da un LLM per escludere eventuali testi non pertinenti. Infine, il modello Llama 3 viene utilizzato per generare i dati sintetici in formato JSON, garantendo che siano completi e corretti.

Gestione degli endpoint

```python
Arianna
@app.post("/synthetic-data/load-data-description")
async def synthetic_data_load_documents(files: list[UploadFile]):
    res = await load_documents_synthetic_data_to_vectordb(files)
    return {
        "status": "success",
        "response": res
    }


Arianna
@app.post("/synthetic-data/load-data-distribution")
async def synthetic_data_load_documents(files: list[UploadFile]):
    res = await load_documents_synthetic_data_to_vectordb(files)
    return {
        "status": "success",
        "response": res
    }


Arianna
@app.post("/synthetic-data/get-client-query")
async def synthetic_data_generate_data(client_query: ClientQueryRequest):
    query_str = client_query.query
    content_response = await generate_data_synthetic_data_to_vectordb(query_str)
    print(content_response)
    return {
        "content": content_response
    }
```

Figura 4.16: Endpoint per la generazione dei dati sintetici

Ogni endpoint è progettato per gestire un tipo specifico di richiesta e utilizza funzioni asincrone per garantire un'elaborazione efficiente delle operazioni di caricamento e generazione dei dati sintetici.

4.3 Panoramica dell'Applicazione

4.3.1 Generatore di Unit Test

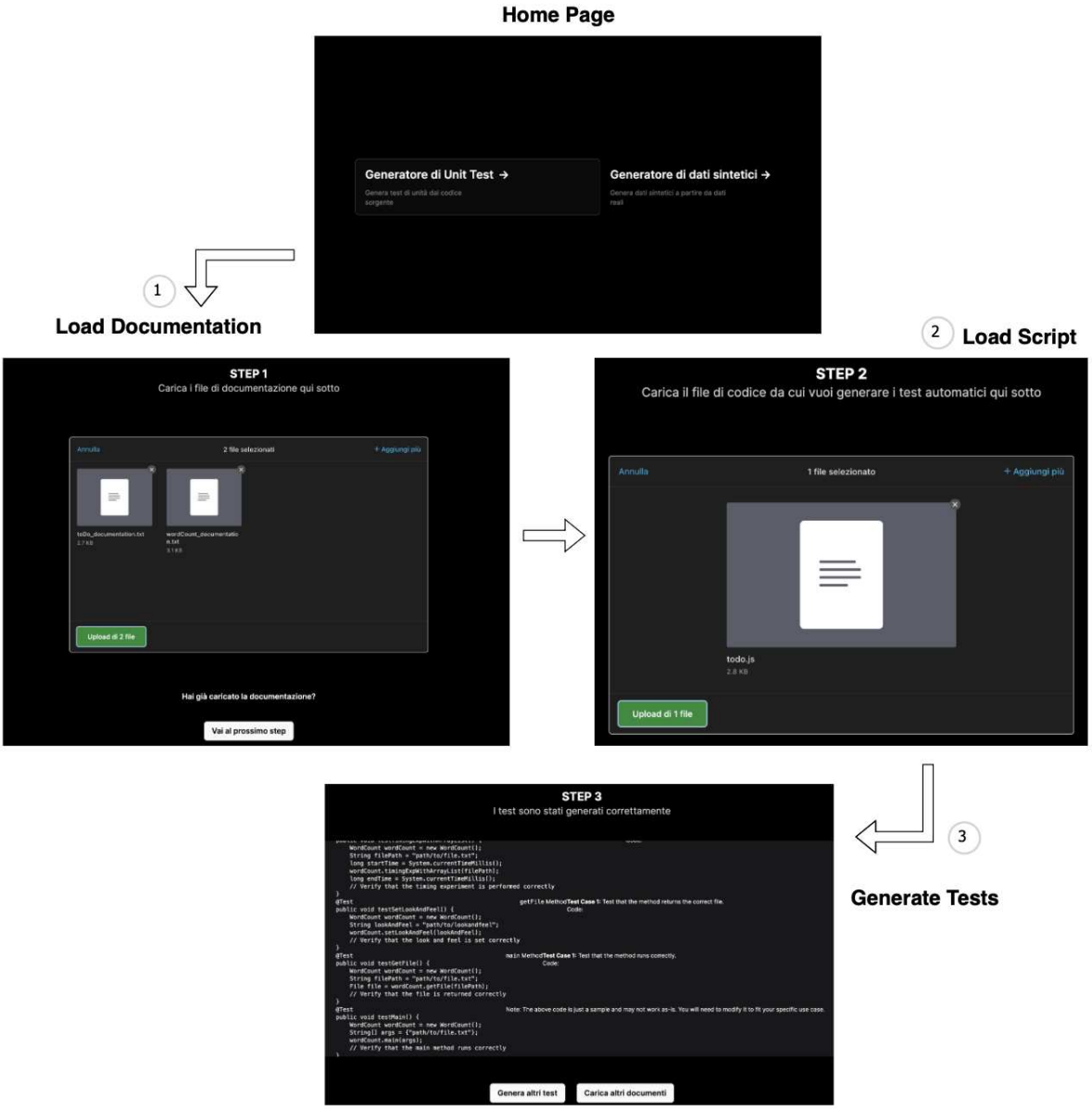


Figura 4.17: Generatore di Unit Test

In questo esempio, vengono mostrati gli step della funzionalità di generazione dei test di unità. Nella prima immagine viene mostrata la home page dell'applicazione, dove sono presenti due bottoni che portano alle due funzionalità principali. La seconda e la terza immagine illustrano rispettivamente il caricamento della documentazione e dello script di cui creare i test. È importante notare che, se la documentazione è già

stata caricata in precedenza, lo step 1 può essere saltato.
Infine, nell'ultima immagine, viene presentato un esempio di file di output, in cui sono generati i test di unità. A questo punto, l'utente può decidere se generare altri test utilizzando lo stesso script e documentazione, oppure caricare altra documentazione tornando a un passo precedente.

4.3.2 Generatore di Dati Sintetici

Figura 4.18: Prima parte del generatore di dati sintetici

In questa immagine, vengono illustrati i passaggi relativi alla funzionalità di generazione di dati sintetici. La seconda e la terza immagine mostrano rispettivamente il caricamento dei file descrittivi dei dati che si desidera generare e dei file contenenti informazioni specifiche, come le loro distribuzioni. Si sottolinea che, qualora i file descrittivi fossero già stati caricati, è possibile saltare il primo passaggio.

Figura 4.19: Seconda parte del generatore di dati sintetici

In questa immagine viene illustrato il terzo step dell'applicazione, in cui l'utente deve inserire la query che definirà le specifiche dei dati da generare. Viene inoltre mostrata la fase di caricamento del risultato finale. Nell'ultimo step, viene presentato il risultato finale: un file in formato JSON che contiene tutti i dati sintetici generati.

4.4 Verifica dei Risultati

L'analisi condotta per verificare la correttezza dell'applicazione web prodotta, ha riguardato due aspetti principali: la generazione dei test di unità e la generazione dei dati sintetici.

Per quanto concerne la generazione dei test di unità, la verifica ha incluso l'implementazione dei test generati e la successiva conferma che tali test coprissero tutti i metodi e le funzionalità dello script fornito. Questo ha assicurato che ogni parte del codice fosse adeguatamente testata, garantendo così una copertura completa.

Dall'altra parte, per la generazione dei dati sintetici, si è proceduto a controllare che i dati prodotti dall'applicazione corrispondessero esattamente alle specifiche richieste dall'utente.

La correttezza dei risultati ottenuti è stata ulteriormente confermata tramite una revisione eseguita dal tutor aziendale, il quale ha validato la precisione e l'affidabilità delle funzionalità implementate. Tuttavia, non sono state condotte analisi quantitative poiché non erano state richieste al momento della definizione dei requisiti di progetto. L'obiettivo principale era verificare la correttezza funzionale e qualitativa dell'applicazione piuttosto che misurare le prestazioni o altre metriche quantitative.

Capitolo 5

Federated Learning e Sicurezza dei Dati

Il capitolo seguente presenta la soluzione progettuale per l'implementazione di una versione semplificata del Federated Learning (FL).

5.1 Proposta di Soluzione

Per implementare una simulazione semplificata di Federated Learning, è stata adottata una serie di strategie volte a esplorare e dimostrare i principi fondamentali di questo approccio innovativo. L'obiettivo era creare un ambiente controllato in cui fosse possibile studiare l'efficacia e le sfide del Federated Learning senza la complessità aggiuntiva di un sistema distribuito su larga scala. Le principali strategie impiegate includono:

- **Dataset Suddiviso**: un dataset comune è stato suddiviso in più sottogruppi, ciascuno assegnato a un "nodo" simulato. Ogni nodo rappresenta un dispositivo locale che addestra il modello utilizzando solo il proprio subset di dati, riflettendo il comportamento dei dispositivi nell'architettura del Federated Learning;
- **Aggiornamenti Incrementali**: ogni nodo addestra il modello localmente per un numero definito di epoche e successivamente invia gli aggiornamenti dei

parametri (pesi del modello) a un server centrale. Questo server aggrega gli aggiornamenti ricevuti da tutti i nodi per migliorare il modello globale, mimando il ciclo di Federated Learning;

- **Simulazione di Ambiente Distribuito**: sebbene la simulazione sia stata eseguita su una singola macchina, abbiamo implementato meccanismi per simulare la comunicazione tra nodi e server centrale.

Queste strategie hanno permesso di creare una piattaforma di test robusta e flessibile, ideale per studiare i principi del Federated Learning e valutare le sue performance e complessità in un ambiente semplificato ma rappresentativo.

5.2 Strumenti utilizzati

Nel progetto di Federated Learning, sono stati impiegati diversi strumenti per gestire e ottimizzare il processo di sviluppo e addestramento del modello:

- **Hydra**: framework per la gestione delle configurazioni, permettendo di organizzare e le varie impostazioni del progetto in modo flessibile;
- **Flower**: framework open-source progettato per facilitare l'implementazione e l'esecuzione di progetti di Federated Learning (FL);
- **PyTorch**: libreria open-source di Machine Learning (ML);
- **Torchvision**: libreria complementare a PyTorch, serve a gestire dataset di immagini, modelli preaddestrati, e trasformazioni delle immagini;
- **MNIST**: database che offre un insieme di immagini di cifre scritte a mano, molto spesso usato per l'addestramento di modelli.

5.3 Codice Sviluppato

A seguire verranno mostrati alcuni di snippet di codice rappresentati gli snodi principali della soluzione proposta.

Creazione del modello

```
class Net(nn.Module):

    def __init__(self, num_classes: int) -> None:
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d( in_channels: 1,  out_channels: 6,  kernel_size: 5)
        self.pool = nn.MaxPool2d( kernel_size: 2,  stride: 2)
        self.conv2 = nn.Conv2d( in_channels: 6,  out_channels: 16,  kernel_size: 5)
        self.fc1 = nn.Linear(16 * 4 * 4,  out_features: 120)
        self.fc2 = nn.Linear( in_features: 120,  out_features: 84)
        self.fc3 = nn.Linear( in_features: 84, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figura 5.1: Implementazione di una rete neurale

Questo codice definisce una Convolutional Neural Network (CNN)$_G$in PyTorch per la classificazione di immagini, in questo caso provenienti da MNIST. Le CNN prendono in input un'immagine, trasformata in un tensore, e la processano attraverso vari strati convoluzionali che estraggono e combinano automaticamente le sue caratteristiche significative.

Training del modello

```python
def train(net, trainloader, optimizer, epochs, device: str):
    criterion = torch.nn.CrossEntropyLoss()
    net.train()
    net.to(device)
    for _ in range(epochs):
        for images, labels in trainloader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            loss = criterion(net(images), labels)
            loss.backward()
            optimizer.step()
```

Figura 5.2: Funzione di training del modello

Il codice illustra la funzione responsabile dell'addestramento completo della rete neurale. Inizialmente, viene definita la loss function, che in questo caso è la **CrossEntropyLoss**, adatta per problemi di classificazione. Successivamente, si itera attraverso il training dataset, calcolando e aggiornando i gradienti dei parametri del modello. I dati di addestramento sono forniti dalla struttura di dati denominata trainloader.

Download del dataset MNIST

```python
def get_mnist(data_path: str = "./data"):
    tr = Compose([ToTensor(), Normalize( mean: (0.1307,), std: (0.3081,))])
    trainset = MNIST(data_path, train=True, download=True, transform=tr)
    testset = MNIST(data_path, train=False, download=True, transform=tr)
    return trainset, testset
```

Figura 5.3: Funzione di download del dataset

Qui viene mostrato il download dei dataset di training e di test da MNIST, in particolare si nota la fase di pre-processing delle immagini che vengono trasformate in tensori e normalizzate.

Struttura del Client

```python
class FlowerClient(fl.client.NumPyClient):

    def __init__(self, trainloader, vallodaer, num_classes) -> None:
        super().__init__()
        self.trainloader = trainloader
        self.valloader = vallodaer
        self.model = Net(num_classes)
        self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    def evaluate(self, parameters: NDArrays, config: Dict[str, Scalar]):
        self.set_parameters(parameters)
        loss, accuracy = test(self.model, self.valloader, self.device)
        return float(loss), len(self.valloader), {"accuracy": accuracy}
```

Figura 5.4: Classe della struttura client

```python
def generate_client_fn(trainloaders, valloaders, num_classes):
    def client_fn(cid: str):
        return FlowerClient(
            trainloader=trainloaders[int(cid)],
            vallodaer=valloaders[int(cid)],
            num_classes=num_classes,
        ).to_client()

    return client_fn
```

Figura 5.5: Funzione che genera istanze del client

Il codice illustra come sono strutturati i client utilizzati in una simulazione di Federated Learning (FL). Vengono preparati i dati di addestramento (**trainloader**) e di validazione (**valloader**). Viene creato un oggetto Net per rappresentare il modello di rete neurale utilizzato per la classificazione. Nella seconda parte, viene mostrata una funzione responsabile di generare istanze della classe **FlowerClient** all'interno del sistema.

Aggiornamento del modello nel Server

```python
def get_evaluate_fn(num_classes: int, testloader):
    def evaluate_fn(server_round: int, parameters, config):

        model = Net(num_classes)

        device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

        params_dict = zip(model.state_dict().keys(), parameters)
        state_dict = OrderedDict({k: torch.Tensor(v) for k, v in params_dict})
        model.load_state_dict(state_dict, strict=True)

        loss, accuracy = test(model, testloader, device)
        return loss, {"accuracy": accuracy}

    return evaluate_fn
```

Figura 5.6: Funzione di aggiornamento del modello nel server

La funzione mostrata sopra, dato un set di parametri del modello, carica questi parametri in un modello **Net**, valuta le sue prestazioni su dati di test specificati e restituisce la loss e l'accuratezza ottenuta. Questo processo è utile per monitorare e valutare il miglioramento del modello nel corso del tempo.

Strategia usata nel Federated Learning

```
strategy = fl.server.strategy.FedAvg(
    fraction_fit=0.0,
    min_fit_clients=cfg.num_clients_per_round_fit,
    fraction_evaluate=0.0,
    min_evaluate_clients=cfg.num_clients_per_round_eval,
    min_available_clients=cfg.num_clients,
    on_fit_config_fn=get_on_fit_config(
        cfg.config_fit
    ),
    evaluate_fn=get_evaluate_fn(cfg.num_classes, testloader),
)

history = fl.simulation.start_simulation(
    client_fn=client_fn,
    num_clients=cfg.num_clients,
    config=fl.server.ServerConfig(
        num_rounds=cfg.num_rounds
    ),
    strategy=strategy,
    client_resources={
        "num_cpus": 2,
        "num_gpus": 0.0,
    },
)
```

Figura 5.7: Codice della strategia di FL

Il code snippet mostra la configurazione e l'avvio di una simulazione di Federated Learning (FL), i valori sono:

- **evaluate_fn**: utilizzata per valutare i modelli sul server utilizzando i dati di test (testloader);
- **client_fn**: funzione che genera le istanze di FlowerClient per ciascun client;
- **num_clients**: numero totale di clienti nel sistema;
- **config**: configurazione del server che specifica il numero di round di addestramento **(num_rounds)**;
- **strategy**: strategia di federated learning da utilizzare, già configurata sopra;
- **client_resources**: risorse disponibili per ciascun client, come il numero di CPU **(num_cpus)** e il numero di GPU **(num_gpus)**.

Risultati ottenuti

```
[2024-07-03 04:36:00,839][flwr][INFO] - [SUMMARY]
[2024-07-03 04:36:00,839][flwr][INFO] - Run finished 10 round(s) in 54.69s
[2024-07-03 04:36:00,839][flwr][INFO] -     History (loss, centralized):
[2024-07-03 04:36:00,840][flwr][INFO] -         round 0: 182.248277425766
[2024-07-03 04:36:00,840][flwr][INFO] -         round 1: 177.5578019618988
[2024-07-03 04:36:00,840][flwr][INFO] -         round 2: 141.0995738506317
[2024-07-03 04:36:00,840][flwr][INFO] -         round 3: 56.06191110610962
[2024-07-03 04:36:00,840][flwr][INFO] -         round 4: 42.310889184474945
[2024-07-03 04:36:00,840][flwr][INFO] -         round 5: 30.76065270602703
[2024-07-03 04:36:00,840][flwr][INFO] -         round 6: 25.429320849478245
[2024-07-03 04:36:00,840][flwr][INFO] -         round 7: 19.81338579952717
[2024-07-03 04:36:00,840][flwr][INFO] -         round 8: 17.074780855327845
[2024-07-03 04:36:00,840][flwr][INFO] -         round 9: 15.109348058700562
[2024-07-03 04:36:00,840][flwr][INFO] -         round 10: 13.902292463928461
[2024-07-03 04:36:00,840][flwr][INFO] -     History (metrics, centralized):
[2024-07-03 04:36:00,840][flwr][INFO] -     {'accuracy': [(0, 0.1063),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (1, 0.2855),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (2, 0.5607),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (3, 0.7641),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (4, 0.8295),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (5, 0.8907),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (6, 0.9097),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (7, 0.9241),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (8, 0.9347),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (9, 0.9416),
[2024-07-03 04:36:00,840][flwr][INFO] -                   (10, 0.944)]}
[2024-07-03 04:36:00,840][flwr][INFO] -
```

Figura 5.8: Stampa dei risultati ottenuti

Il log indica che la simulazione del Federated Learning (FL) è stata eseguita per 10 round, ciascuno dei quali ha migliorato l'accuratezza del modello e ridotto la loss. L'accuratezza finale raggiunta è stata del **94,37%** e il tempo totale impiegato per la simulazione è stato di circa **52,66** secondi.

La verifica della correttezza del programma si è basata su questi indicatori chiave: un decremento costante della loss function indica che il modello sta imparando correttamente dai dati, mentre un incremento dell'accuratezza suggerisce che il modello sta migliorando la sua capacità di fare previsioni accurate. I log dettagliati hanno permesso di confermare che il sistema di Federated Learning (FL) si comporta come previsto.

Capitolo 6

Conclusioni

Il capitolo contiene le considerazioni finali sull'esperienza di stage.

6.1 Raggiungimento degli obiettivi

Al termine del tirocinio gli obiettivi che erano stati fissati nella tabella aggiornata dei requisiti sono stati raggiunti, rispettando anche le ore preventivate a settimana per l'esecuzione delle attività programmate.

ID	Descrizione	Soddisfatto
ROB1	Configurare e implementare strategie di generazione di test tramite LLM utilizzando il codice sorgente.	Si
ROB2	Integrare l'LLM con la documentazione del progetto per la generazione dei test.	Si
ROB3	Implementare una soluzione per la creazione di dati sintetici attraverso LLM.	Si
ROB4	Valutare e confrontare i risultati ottenuti con dati sintetici rispetto a quelli reali.	Si
ROB5	Implementare una versione semplificata di Federated Learning.	Si
RDE1	Estendere l'utilizzo dell'LLM per la generazione di test anche ad altri linguaggi di programmazione oltre a Java e JavaScript.	Si
RDE2	Condurre analisi approfondite sul sistema di sicurezza e privacy dei dati che vengono utilizzati nel processo di addestramento di un modello nel Federated Learning.	Si

Tabella 6.1: Tabella del raggiungimento degli obiettivi

6.2 Conoscenze acquisite

Durante l'esperienza di stage presso l'azienda Zucchetti, ho avuto l'opportunità di approfondire significativamente le mie conoscenze in vari ambiti avanzati del Machine Learning (ML). In particolare, ho acquisito competenze nell'utilizzo e nel funzionamento delle RAG, nei Large Language Models (LLM) e nel Federated Learning (FL).
La generazione di dati sintetici è stata un'altra area di grande interesse durante il mio stage. Ho appreso come creare dati sintetici che possano simulare accuratamente le caratteristiche dei dati reali, fornendo un'alternativa preziosa per l'addestramento e la validazione dei modelli di ML in assenza di dati reali sufficienti o per proteggere la privacy degli utenti.
Nello sviluppo dell'applicazione "Coding Assistant" ho avuto l'occasione di imparare e approfondire nuovi framework e linguaggi di programmazione.
L'esperienza presso Zucchetti mi ha permesso di sviluppare competenze che saranno fondamentali per il mio futuro professionale.

6.3 Valutazione personale

L'esperienza di stage presso Zucchetti è stata di grande valore, permettendomi di crescere sia a livello di conoscenze che come persona. L'ambiente di lavoro tranquillo mi ha consentito di realizzare il progetto senza distrazioni. Inoltre, ho avuto l'opportunità di imparare a lavorare in team e a migliorare le mie capacità di problem solving, sviluppando competenze essenziali per il mio futuro professionale. Questa esperienza ha arricchito significativamente il mio percorso formativo e personale, preparandomi meglio alle sfide del mondo del lavoro.

Glossario

Convolutional Neural Network (CNN) E' un tipo di rete neurale artificiale progettata per elaborare dati come immagini. 48

Deep Learning (DL) Deep learning è un sottoinsieme del machine Learning (ML), basato su livelli di reti neurali che sono algoritmi che simulano il processo di apprendimento del cervello umano. 4

Embedding Un embedding è una rappresentazione numerica di un testo, come parole, frasi o interi documenti. Gli embeddings vengono utilizzati per trasformare il testo in vettori che catturano le caratteristiche semantiche e sintattiche. 8

Federated Learning (FL) Il Federated Learning è un approccio nel campo dell'apprendimento automatico che consente di addestrare modelli di intelligenza artificiale in modo collaborativo e distribuito, evitando la necessità di condividere i dati tra i vari partecipanti. 1, 9

K-Nearest Neighbour (KNN) K-Nearest Neighbors (KNN) è un algoritmo di machine learning usato principalmente per problemi di classificazione e regressione. Si basa sull'assunto che punti dati simili si trovano vicini nello spazio delle caratteristiche. 8

Large Language Model (LLM) I Large Language Models (LLM) rappresentano sofisticati sistemi di intelligenza artificiale (AI) concepiti per elaborare, comprendere e generare testo simile a quello prodotto dall'uomo. Fondati sulle tecniche di deep learning, sono formati su enormi dataset composti generalmente da miliardi di parole provenienti da varie fonti come siti web, libri e

articoli. Questo vasto addestramento permette agli LLM di catturare le sottili sfumature del linguaggio, della grammatica, del contesto e persino di alcuni aspetti della cultura generale. 1

Machine Learning (ML) Il Machine Learning è una branca dell'intelligenza artificiale (AI) che si occupa di creare sistemi che apprendono o migliorano le performance in base ai dati che utilizzano. 1

Natural Language Processing (NLP) Il Natural Language Processing (NLP), combina la linguistica computazionale (modellazione del linguaggio umano basata su regole) con modelli statistici e di machine learning per consentire a computer di riconoscere, comprendere e generare testo e parlato. 4

Neural Networks (NN) Una rete neurale è un programma di machine learning, o modello, che prende decisioni in modo simile al cervello umano, utilizzando processi che imitano il modo in cui i neuroni biologici lavorano insieme per identificare fenomeni, pesare le opzioni e arrivare alle conclusioni. 5

Proof of Concept (PoC) Un Proof of Concept è un prototipo o un'implementazione minimale utilizzata per verificare che un'idea, un concetto o una tecnologia sia tecnicamente fattibile e possa funzionare come previsto. 2

Query Richiesta di informazioni formulata da un utente. 8

Retrieval-Augmented Generation (RAG) La Retrieval-Augmented Generation (RAG) è il processo di ottimizzazione dell'output di un LLM, in modo che si faccia riferimento a una serie di dati e informazioni al di fuori dal training dataset con cui è stato addestrato. 7

Separation of Concerns (SoC) Il principio di separation of concerns è un concetto fondamentale in ingegneria del software e nella progettazione di sistemi. Questo principio suggerisce che un sistema complesso dovrebbe essere suddiviso in parti distinte, ognuna delle quali è responsabile di un singolo aspetto del problema o di una singola funzionalità. 24

Transformer I trasformers rappresentano un'architettura di reti neurali progettata per trasformare o modificare una sequenza di input in una sequenza di output. 5

Vector Database Un vector database è un tipo di database progettato specificamente per archiviare, indicizzare e cercare dati rappresentati come vettori. Questi vettori sono tipicamente generati da modelli di machine learning e rappresentano dati complessi come testo, immagini, audio e altre forme di dati non strutturati in uno spazio vettoriale multidimensionale. 8

Bibliografia

Articoli

[1] Humza Naveed et al. «A Comprehensive Overview of Large Language Models». In: *arXiv preprint arXiv:2307.06435* (2024).

[2] Ashish Vaswani et al. «Attention Is All You Need». In: *arXiv preprint arXiv:1706.03762* (2023).

[3] Yunpeng Huang et al. «Advancing Transformer Architecture in Long-Context Large Language Models: A Comprehensive Survey». In: *arXiv preprint* (2024). arXiv:2311.12351 [cs.CL].

[4] Wensheng Gan et al. «Large Language Models in Education: Vision and Opportunities». In: *arXiv preprint* (2023). arXiv:2311.13160 [cs.AI].

[5] Yuanchun Li et al. «Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security». In: *arXiv preprint* (2024). arXiv:2401.05459 [cs.HC].

[6] Xinyi Hou et al. «Large Language Models for Software Engineering: A Systematic Literature Review». In: *arXiv preprint* (2024). arXiv:2308.10620 [cs.SE].

[7] Junjie Wang et al. «Software Testing with Large Language Models: Survey, Landscape, and Vision». In: *arXiv preprint* (2024). arXiv:2307.07221 [cs.SE].

[8] Max Schäfer et al. «An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation». In: *arXiv preprint* (2023). arXiv:2302.06527 [cs.SE].

[9] Nicholas Asher et al. «Limits for Learning with Language Models». In: *arXiv preprint* (2023). arXiv:2306.12213 [cs.CL].

[10] Yotam Wolf et al. «Fundamental Limitations of Alignment in Large Language Models». In: *arXiv preprint* (2024). arXiv:2304.11082 [cs.CL].

[11] Shervin Minaee et al. «Large Language Models: A Survey». In: *arXiv preprint arXiv:2402.06196* (2024).

[12] Lei Huang et al. «A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions». In: *arXiv preprint* (2023). arXiv:2311.05232 [cs.CL].

[13] Banghao Chen et al. «Unleashing the potential of prompt engineering: a comprehensive review». In: *arXiv preprint* (2024). arXiv:2310.14735 [cs.CL].

[14] Yunfan Gao et al. «Retrieval-Augmented Generation for Large Language Models: A Survey». In: *arXiv preprint* (2024). arXiv:2312.10997 [cs.CL].

[15] Zhuoyan Li et al. «Synthetic Data Generation with Large Language Models for Text Classification: Potential and Limitations». In: *arXiv preprint arXiv:2310.07849* (2023). arXiv: 2310.07849 [cs.CL].

[16] Yukai Xu, Jingfeng Zhang e Yujie Gu. «Privacy-Preserving Heterogeneous Federated Learning for Sensitive Healthcare Data». In: *arXiv preprint arXiv:2406.10563* (2024).

[17] Linlin Wang et al. «Linkage on Security, Privacy and Fairness in Federated Learning: New Balances and New Perspectives». In: *arXiv preprint arXiv:2406.10884* (2024).

[18] Nadia Alshahwan et al. «Automated Unit Test Improvement using Large Language Models at Meta». In: *arXiv preprint* (2024). arXiv:2402.09171 [cs.SE].

[19] Zhiqiang Yuan et al. «No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation». In: *arXiv preprint* (2024). arXiv:2305.04207 [cs.SE].

[20] Yinghao Chen et al. «ChatUniTest: A Framework for LLM-Based Test Generation». In: *arXiv preprint* (2024). arXiv:2305.04764 [cs.SE].

[21] Xu Guo e Yiqiang Chen. «Generative AI for Synthetic Data Generation: Methods, Challenges and the Future». In: (2024). arXiv: 2403.04190 [cs.LG].

[22] Ajay Patel, Colin Raffel e Chris Callison-Burch. «DataDreamer: A Tool for Synthetic Data Generation and Reproducible LLM Workflows». In: *arXiv preprint arXiv:2402.10379* (2024). arXiv: 2402.10379 [cs.CL].

[23] Ainie Noruzman, Ngahzaifa Ab Ghani e N Zulkifli. «Gretel.ai: Open-Source Artificial Intelligence Tool To Generate New Synthetic Data». In: (mar. 2022).

[24] NVIDIA. «Nemotron-4 340B Technical Report». In: (2024).

[25] Holger R. Roth et al. «NVIDIA FLARE: Federated Learning from Simulation to Real-World». In: (2022). DOI: 10.48550/ARXIV.2210.13291. URL: https://arxiv.org/abs/2210.13291.

[26] Patrick Foley et al. «OpenFL: the open federated learning library». In: *Physics in Medicine amp; Biology* 67.21 (ott. 2022), p. 214001. ISSN: 1361-6560. DOI: 10.1088/1361-6560/ac97d9. URL: http://dx.doi.org/10.1088/1361-6560/ac97d9.

[41] Heiko Ludwig et al. «IBM Federated Learning: an Enterprise Framework White Paper V0.1». In: *arXiv preprint arXiv:2007.10987* (2020).

[42] Laiqiao Qin et al. «Knowledge Distillation in Federated Learning: a Survey on Long Lasting Challenges and New Solutions». In: *arXiv preprint arXiv:2406.10861* (2024).

Sitografia

[27] *Logo Zucchetti SPA*. URL: https://www.zucchetti.it/.

[28] *Logo LangChain*. URL: https://www.langchain.com.

[29] *Logo Ollama*. URL: https://ollama.com.

[30] *Logo ChromaDB*. URL: https://www.trychroma.com.

[31] *Immagine AI*. URL: https://www.appypie.com/blog/what-are-large-language-models.

[32] *Logo ChatGPT*. URL: https://openai.com/.

[33] *Logo LLaMA*. URL: https://llama.meta.com.

[34] *RAG Image*. URL: https://medium.com/enterprise-rag/an-introduction-to-rag-and-simple-complex-rag-9c3aa9bd017b.

[35] *Logo Google Scholar*. URL: https://scholar.google.com.

[36] *Logo Google Docs*. URL: https://www.google.com/docs/about/.

[37] *Logo GitHub*. URL: https://github.com/.

[38] *Logo PyCharm*. URL: https://www.jetbrains.com/pycharm/.

[39] *Logo Keynote*. URL: https://www.apple.com/keynote/.

[40] *Logo overleaf*. URL: https://www.overleaf.com/.