

UNIVERSITA' DEGLI STUDI DEL SANNIO

DIPARTIMENTO DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN BIOTECNOLOGIE

MATERIA DI TESI: BIOINFORMATICA

SPECIALIZZAZIONE DI LARGE LANGUAGE MODELS MEDIANTE LoRA: UN'APPLICAZIONE AL DOMINIO DELLA BIOINFORMATICA

RELATORE:

Ch.mo Prof.

FRANCESCO NAPOLITANO

CANDIDATO:

ROBERTO PADUANO

MATR. 171001779

Anno Accademico 2025-2026

INDICE

INTRODUZIONE 1

I. Stato dell'Arte

1. Bioinformatica moderna 2

1.1 Introduzione alla Bioinformatica e all'Era Omica .....

1.2 Principi di Analisi delle Sequenze e Allineamento .....

1.3 Analisi dell'Espressione Genica e Trascrittomica ... **Error! Bookmark not defined.**

1.4 Bioinformatica Strutturale e Filogenesi Molecolare 4

2. LLM in ambito scientifico 5

2.1 Modelli specialistici e loro limiti 6

2.2 Modelli generalisti e loro limiti .....

3. Tecniche di Fine-Tuning 7

3.1 Full Fine-Tuning .....

3.2 LoRA: teoria e principi 9

4. La famiglia di modelli Qwen2.5 11

II. Materiali e Metodologia

1. Dataset 13

1.1 Domini bioinformatici 14

1.2 Esempi di domande 15

2. Setup Sperimentale 16

2.1 Hardware e Software (CPU, RAM, OS, Linguaggio) 17

2.2 Configurazione LoRA: Fase di TRAINING 18

2.3 Parametri di inferenza: Fase di BENCHMARK 22

3. Metriche di Valutazione 24

3.1 Quantitativa: Token F1 Score, Exact Match (EM) .....

3.2 Qualitativa: Valutazione semantica LLM-as-a-Judge 26

III. Risultati Sperimentali

1. Valutazione Quantitativa (F1 Score) 27

1.1 Grafico a barre Base vs LoRA .....

2. Valutazione Qualitativa (Semantica) 29

2.1 Grafico a barre Base vs LoRA .....

2.2 Punto di Saturazione: Quando LoRA smette di essere utile 30

3. Prestazioni e Risorse 31

3.1 Requisiti hardware (RAM/VRAM) .....
3.2 Tempi di training e inferenza 32
IV. Analisi Economica e Ambientale
1. Metodologia di Calcolo dei Costi.....34
1.1 ROI (Return on Investment).....36
2. Analisi dell'impatto Ambientale.....37
2.1 Confronto dell'Impatto Ambientale tra i modelli .....
2.2 Grafico Impatto ambientale 38
V. Conclusioni.....39
BIBLIOGRAFIA/SITOGRAFIA 42

INTRODUZIONE

Negli ultimi anni l'intelligenza artificiale ha smesso di essere una promessa confinata ai laboratori di informatica e delle aziende, per diventare una presenza silenziosa ma costante nella quotidianità della maggior parte della popolazione mondiale. Sta cambiando il modo in cui lavoriamo, come cerchiamo informazioni, come ci curiamo e persino come interagiamo tra di noi. Si tratta di una trasformazione profonda, che va ben oltre la semplice innovazione tecnica e toccando da vicino la natura stessa della nostra società.

Da un lato, questo cambiamento porta con sé un entusiasmo innegabile: processi che prima richiedevano giorni ora avvengono in pochi secondi, e si aprono possibilità del tutto inedite per risolvere problemi complessi. Dall'altro, però, emerge una crescente inquietudine. Ci si interroga sul futuro del lavoro, sulla gestione della privacy, sulla trasparenza delle decisioni prese dagli algoritmi e sul rischio che la tecnologia finisca per amplificare le disuguaglianze anziché ridurle.

La sfida centrale di oggi non è più soltanto capire fin dove possa spingersi l'innovazione, ma comprendere come guidarla in modo che rimanga al servizio delle persone. È proprio all'interno di questo scenario dinamico e ricco di ombre e luci che si inserisce questo lavoro, con l'intento di esplorare da vicino una delle trasformazioni più significative della nostra epoca.

In questo scenario, il dibattito sull'impatto dell'intelligenza artificiale trova un'applicazione estremamente concreta nell'ambito della formazione universitaria e scientifica, un contesto in cui la precisione dei contenuti non è un dettaglio secondario ma un requisito fondamentale. Quando un ambito di studio si fa altamente specialistico e interdisciplinare, come accade per la bioinformatica, sorge naturale chiedersi se gli strumenti tecnologici attualmente a disposizione degli studenti siano davvero adeguati o se rischino invece di mostrare dei limiti strutturali.

Le intelligenze artificiali generaliste, per quanto straordinariamente versatili, si trovano spesso ad affrontare discipline complesse offrendo risposte superficiali o prive del rigore necessario per una preparazione d'esame. È proprio da questa criticità che nasce la necessità di esplorare strade alternative, valutando se la specializzazione di un modello possa fare la differenza nell'accompagnare uno studente durante il suo percorso di apprendimento.

Tra le diverse tecniche di personalizzazione, il metodo LoRA si impone come uno degli approcci più promettenti, poiché promette di adattare grandi modelli a domini specifici con un dispendio di risorse contenuto. Il cuore di questa ricerca risiede quindi nel verificare se l'addestramento mirato di un modello in ambito bioinformatico apporti un beneficio tangibile e misurabile rispetto alle soluzioni generiche. Attraverso questo confronto diretto, lo studio mira a stabilire se l'investimento necessario per creare e gestire un modello locale specializzato sia effettivamente giustificato da un miglioramento delle prestazioni e del supporto allo studio, oppure se l'utilizzo dei grandi modelli generalisti rimanga, al momento, la scelta più pratica ed efficace.

I. STATO DELL'ARTE

Nel seguente capitolo troveremo alcuni dei capisaldi della Bioinformatica moderna, punti chiave per la costruzione del dataset utilizzato per lo sviluppo della seguente tesi.

1.BIOINFORMATICA MODERNA

1.1 Introduzione alla Bioinformatica e all'Era Omica

Negli ultimi decenni il modo di fare ricerca nelle scienze della vita è cambiato radicalmente. Siamo passati da una biologia tradizionale, legata quasi esclusivamente al lavoro di laboratorio al bancone (*in vitro*), a una disciplina guidata dall'analisi massiva dei dati biologici (*in silico*). Questa svolta è stata resa possibile dall'avvento delle tecnologie "omiche" — come la genomica, la trascrittomica e la proteomica — capaci di generare ogni giorno una mole di informazioni enorme e complessa (Goodwin et al., 2016).

In questo scenario, la bioinformatica non è più un semplice strumento di supporto, ma un vero e proprio pilastro della biotecnologia moderna. Si tratta di un ambito fortemente interdisciplinare dove la biologia molecolare si intreccia con l'informatica, la matematica e la statistica, fornendo i metodi necessari per organizzare, analizzare e dare un senso biologico a questi Big Data.

Oggi, l'accesso a questi dati è reso possibile grazie a grandi banche dati pubbliche gestite da consorzi internazionali come il *National Center for Biotechnology Information* (NCBI), l'*European Bioinformatics Institute* (EMBL-EBI) o il *Protein Data Bank* (PDB) per le strutture tridimensionali (Sayers et al., 2022; The UniProt Consortium et al., 2023). Per chiunque si occupi di biotecnologie, saper interrogare queste risorse e orientarsi tra i vari database è diventata una competenza di base fondamentale.

1.2 Principi di Analisi delle Sequenze e Allineamento

Tra le attività principali della bioinformatica c'è senza dubbio il confronto tra sequenze di DNA, RNA e proteine. La logica biologica alla base di questo tipo di analisi si fonda sul concetto di omologia: sequenze che derivano da un antenato comune tendono a conservare nel tempo una struttura e una funzione simili. Quando si parla di omologia, però, è importante distinguere tra geni ortologhi (ovvero geni presenti in specie diverse separatisi a seguito di un evento di speciazione, che di solito mantengono la stessa funzione) e geni paralogi (originati da un evento di duplicazione all'interno dello stesso organismo) (Koonin, 2005;).

Per misurare quanto due sequenze si assomiglino si usa l'allineamento, una procedura matematica che sovrappone le sequenze per evidenziarne le regioni identiche o simili. Gli algoritmi di allineamento si dividono principalmente in due grandi famiglie:

- Allineamento Globale: Cerca di allineare le sequenze per tutta la loro lunghezza. L'algoritmo di riferimento è quello di Needleman-Wunsch, basato sulla programmazione dinamica. Questo metodo calcola l'allineamento globale migliore in base a un sistema di punteggi che tiene conto delle sostituzioni di amminoacidi o

nucleotidi (usando matrici come BLOSUM62) e delle penalità per l'inserimento di interruzioni o spazi (*gap*) (Needleman & Wunsch, 1970)

- Allineamento Locale: Invece di considerare la sequenza intera, va a caccia delle sole regioni che presentano un'alta similarità, ignorando le parti divergenti. L'algoritmo classico per questo approccio è quello di Smith-Waterman (Smith & Waterman, 1981).

Sebbene questi metodi matematici siano estremamente precisi, confrontare una sequenza contro un'intera banca dati di milioni di voci richiederebbe un tempo di calcolo enorme. Per superare questo ostacolo sono stati sviluppati algoritmi euristici, primo fra tutti BLAST (*Basic Local Alignment Search Tool*) (Altschul et al., 1990). BLAST velocizza immensamente la ricerca: anziché calcolare l'allineamento perfetto fin da subito, individua prima delle brevissime parole chiave (*words*) condivise tra le sequenze e poi le estende per ricostruire gli allineamenti locali significativi.

1.3 Analisi dell'Espressione Genica e Trascrittomica

Se lo studio del genoma ci dice com'è fatto il "libretto di istruzioni" di un organismo e quali risposte teoriche può dare, è la trascrittomica a dirci cosa sta succedendo davvero in un preciso momento. Analizzare il trascrittoma significa infatti capire quali geni sono effettivamente accesi o spenti in un determinato tessuto, in una specifica cellula o in risposta a un particolare stimolo esterno o patologico. Per diversi anni questo lavoro è stato affidato ai microarray a DNA. Pur essendo stati fondamentali, i microarray mostravano presto tutti i loro limiti: funzionavano solo per geni già noti (servivano sonde predisegnate) e avevano una capacità limitata di catturare sia le piccolissime che le grandissime variazioni di espressione.

La vera svolta c'est stata con l'arrivo dell'RNA-Seq (*RNA Sequencing*), basato sulle tecnologie di sequenziamento ad alto rendimento (NGS). Con l'RNA-Seq si è fatto un salto di qualità enorme: non solo si riesce a quantificare con grande precisione quanto un gene sia espresso, ma si possono anche scoprire nuovi trascritti mai visti prima, osservare eventi di splicing alternativo e identificare la presenza di RNA non codificanti (Love et al., 2014).

Tuttavia, passare dai dati grezzi del sequenziatore a un risultato biologico con un senso richiede una serie di passaggi informatici ben precisi, nota come pipeline di analisi:

1. Controllo di qualità e pulizia delle sequenze (*Trimming*): La prima cosa da fare è capire se i dati usciti dal sequenziatore sono affidabili. Tramite software come FastQC si controlla la qualità media di ogni singola lettura (*read*); se ci sono basi lette male o residui di adattatori usati durante il laboratorio, questi vengono eliminati prima di proseguire.

2. Mappatura sul genoma di riferimento: Una volta pulite, le *reads* devono essere riposizionate sul genoma di partenza per capire da quale gene provengono. Ma c'è un dettaglio fondamentale: l'RNA-Seq sequenzia mRNA maturo, che non ha più gli introni. Per questo servono software specifici definiti *splice-aware* (come STAR o HISAT2), capaci di capire che una sequenza può "scavalcare" un introne e collegare due esoni distinti.

3. Quantificazione e Normalizzazione: A questo punto si contano semplicemente quante letture sono finite su ciascun gene, ottenendo una tabella di conteggi (*count matrix*). Per poter confrontare campioni diversi tra loro o geni di lunghezza differente, questi numeri non possono essere usati così come sono, ma vanno normalizzati con metriche opportune.

4. Analisi dell'Espressione Differenziale: È il cuore di tutto lo studio. Qui si mettono a confronto due o più gruppi (ad esempio dei campioni sani contro dei campioni malati) per capire quali geni cambiano espressione in modo statisticamente significativo. Siccome i dati di conteggio sono molto variabili e non seguono una distribuzione classica, la statistica dietro questi calcoli è piuttosto sofisticata e si basa su modelli Binomiali Negativi, integrati in pacchetti R come DESeq2 (Love et al., 2014)ed edgeR (Robinson et al., 2010).

Negli ultimi anni, infine, c'è stato un altro passaggio chiave: l'avvento dell'RNA-Seq a singola cellula (*single-cell RNA sequencing*, scRNA-Seq). Fino a poco tempo fa, con il classico approccio *bulk*, l'RNA veniva estratto da un intero pezzo di tessuto, ottenendo una sorta di "media" dell'espressione di migliaia di cellule diverse mescolate insieme. L'scRNA-Seq ha invece premesso di guardare dentro ogni singola cellula isolatamente, svelando un livello di eterogeneità e permettendo di scoprire sottopopolazioni cellulari rare che prima rimanevano del tutto invisibili (Tang et al., 2016).

1.4 Bioinformatica Strutturale e Filogenesi Molecolare

Se la genomica e la trascrittomica ci permettono di interpretare il codice e la dinamica di espressione dei geni, la bioinformatica strutturale e la filogenesi molecolare ci spingono ancora più a fondo, esplorando la forma tridimensionale delle molecole e le loro relazioni evolutive nel tempo.

Bioinformatica Strutturale: dalla Sequenza alla Funzione

Capire come una proteina si ripiega nello spazio è fondamentale: la struttura 3D ne determina in modo diretto la funzione, il meccanismo catalitico e le modalità di interazione con altre molecole. Per decenni, la determinazione delle strutture proteiche si è basata quasi esclusivamente su tecniche sperimentali complesse e costose, come la cristallografia a raggi X, la risonanza magnetica nucleare (NMR) e, più di recente, la microscopia crioelettronica (Cryo-EM), i cui dati sono stati storicamente archiviati nel database pubblico Protein Data Bank (PDB) (Berman, 2000).

Tuttavia, il divario tra il numero di sequenze proteiche note e le strutture sperimentalmente risolte è rimasto immenso per anni. La svolta epocale in questo campo si è verificata con l'avvento dell'intelligenza artificiale e, in particolare, con lo sviluppo di AlphaFold. Sfruttando reti neurali profonde addestrate su decine di migliaia di strutture note e sulle informazioni di co-evoluzione delle sequenze, AlphaFold ha dimostrato di poter predire la struttura tridimensionale delle proteine a partire dalla sola sequenza amminoacidica con una precisione paragonabile a quella dei metodi sperimentali (Jumper et al., 2021).

Accanto alla predizione strutturale, un altro pilastro della bioinformatica strutturale è il docking molecolare. Questa tecnica computazionale permette di simulare e prevedere l'orientamento di legame e l'affinità tra una macromolecola (come un recettore proteico) e un piccolo ligando (come un farmaco o un metabolita). L'impiego del docking molecolare ha rivoluzionato il processo di *drug discovery*, consentendo lo screening virtuale di librerie di milioni di composti chimici per individuare i potenziali candidati terapeutici prima ancora di saggiarli in laboratorio (Morris et al., 2009).

Filogenesi Molecolare: Ricostruire la Storia Evolutiva

In parallelo all'analisi strutturale, la filogenesi molecolare utilizza le sequenze di DNA, RNA e proteine per ricostruire le relazioni evolutive tra gli organismi. Invece di affidarsi ai soli caratteri morfologici tradizionali, l'approccio molecolare confronta le mutazioni accumulate nelle sequenze biologiche nel corso del tempo, intese come veri e propri "orologi molecolari".

La costruzione di un albero filogenetico segue generalmente un percorso ben definito:

1. Allineamento Multiplo di Sequenze (MSA): Si allineano le sequenze omologhe di più specie (usando algoritmi come Clustal Omega o MUSCLE) per identificare i siti nucleotidici o amminoacidici corrispondenti.

2. Selezione del Modello Evolutivo: Si sceglie il modello matematico di sostituzione nucleotidica o amminoacidica che descrive meglio il tasso di mutazione osservato nei dati.

3. Ricostruzione dell'Albero: Si applicano algoritmi di inferenza filogenetica. Tra i metodi più utilizzati vi sono quelli basati sul principio di Massima Parsimonia, quelli basati sul punteggio di Massima Verosimiglianza (Maximum Likelihood) — integrati in software diffusi come RAxML e IQ-TREE — e gli approcci di Inferenza Bayesiana (Felsenstein, 2004).

L'analisi filogenetica non è soltanto una disciplina teorica: trova applicazione diretta nel tracciamento dell'origine e della diffusione di agenti patogeni, nell'identificazione di funzioni geniche conservate e nella comprensione dei meccanismi molecolari dell'adattamento evolutivo.

2. LLM IN AMBITO SCIENTIFICO

Negli ultimi anni, l'arrivo dei modelli di linguaggio di grande taglia (*Large Language Models*, LLM) basati sull'architettura Transformer ha cambiato radicalmente il nostro modo di interagire con le informazioni digitali. Pensati inizialmente per compiti linguistici generali, questi strumenti hanno dimostrato una capacità sorprendente nell'estrarre, sintetizzare e rielaborare testi complessi, trovando subito spazio anche nella ricerca scientifica e biomedica (Vaswani et al., 2017).

2.1 MODELLI SPECIALISTICI E I LORO LIMITI

Analizzare la letteratura scientifica non è banale: il linguaggio della biologia e della medicina è ricco di termini tecnici, acronimi e strutture sintattiche articolate. Per questo motivo, i primi sforzi si sono concentrati sullo sviluppo di modelli *specialistici*, ovvero sistemi addestrati da zero o perfezionati (*fine-tuning*) direttamente su milioni di articoli presi da banche dati come PubMed e PubMed Central.

Un esempio emblematico di questo approccio è BioBERT, una versione di BERT addestrata specificamente su testi biomedici. BioBERT si è rivelato un ottimo alleato in diverse mansioni bioinformatiche, come l'estrazione automatica di relazioni tra molecole o il riconoscimento delle entità biologiche (*Named Entity Recognition*), ovvero la capacità di individuare al volo nomi di geni, proteine e patologie all'interno di un testo (Lee et al., 2020).

Con l'evoluzione dei modelli generativi, la ricerca si è poi spinta verso strumenti ancora più complessi dedicati alla clinica, come Med-PaLM e la sua successiva versione Med-PaLM 2. Addestrato e valutato direttamente con il supporto di medici esperti, Med-PaLM è stato in grado di rispondere a quesiti clinici articolati superando con punteggi elevati i test di abilitazione medica (Singhal et al., 2023).

Tuttavia, anche i modelli specialistici mostrano limiti evidenti:

1. Rigidità e perdita di contesto (*Domain Shift*): Essendo focalizzati su nicchie specifiche di dati, questi modelli tendono ad andare in difficoltà quando si confrontano con problemi multidisciplinari o con dati non strutturati provenienti da contesti differenti.
2. Costi elevati di aggiornamento: La scienza progredisce ogni giorno. Mantenere aggiornato un modello specialistico richiede di riavviare cicli di addestramento complessi e molto costosi dal punto di vista computazionale.
3. Il problema delle allucinazioni: Anche se allenati su fonti ufficiali e controllate, questi modelli non sono del tutto immuni dalla generazione di risposte inventate o imprecise che, se prese per buone in ambito medico o biologico, possono portare a conclusioni errate (Singhal et al., 2023).

2.2 Modelli Generalisti e i Loro Limiti

Parallelamente ai modelli dedicati, l'aumento vertiginoso della potenza di calcolo ha portato all'affermazione dei modelli generalisti (come la serie GPT di OpenAI, Claude di Anthropic o LLaMA di Meta). A differenza di quelli specialistici, questi sistemi non studiano solo la letteratura scientifica, ma vengono addestrati su una massa enorme e variegata di dati: siti web, libri, articoli e codice di programmazione.

È interessante notare come l'enorme mole di dati abbia fatto emergere capacità del tutto inaspettate: abilità di ragionamento logico, traduzione da una lingua all'altra, scrittura di codice (es. in Python o R) e comprensione di istruzioni articolate al primo colpo (zero-shot learning), senza la necessità di un addestramento specifico per quel singolo compito (Brown et al., 2020).

In ambito bioinformatico, un modello generalista può rivelarsi estremamente utile per scrivere rapidamente uno script di analisi, schematizzare un protocollo o riassumere un articolo. Ma quando si tratta di fare ricerca scientifica con il dovuto rigore, emergono criticità importanti:

1. Allucinazioni e citazioni inventate: I modelli generalisti lavorano su base probabilistica, cercando di completare la frase nel modo più credibile possibile. In ambito accademico questo produce spesso un fenomeno insidioso: il modello genera citazioni, titoli di articoli o codici DOI apparentemente perfetti, ma del tutto inesistenti (Ji et al., 2023).

2. Mancanza di tracciabilità (Scatola Nera): Poiché l'addestramento avviene su volumi immensi di dati mescolati tra loro, è quasi impossibile risalire alla fonte esatta da cui il modello ha tratto una determinata informazione, venendo meno al principio fondamentale della verificabilità delle fonti.

3. Opacità dei dati e rischio di Data Leakage: Molte aziende non rendono pubblici i dettagli sul materiale usato per l'addestramento. Di conseguenza, è difficile sapere quanto sia aggiornato il modello o se le buone prestazioni mostrate in un test siano reali oppure dovute al fatto che il modello aveva già "visto" quel test durante la fase di addestramento (Ji et al., 2023).

4. Mancanza di un vero riscontro biologico: Il modello non "capisce" la biologia; riconosce solo schemi di parole. Può quindi proporre un'ipotesi o una combinazione di mutazioni genetiche che suona bene dal punto di vista grammaticale, ma che nella realtà biologica non ha alcun senso.

3. TECNICHE DI FINE-TUNING

Quando si prende un modello di linguaggio di grande taglia appena uscito dalla fase di addestramento primario, ci si trova davanti a uno strumento con una sterminata cultura generale e un’eccellente padronanza della lingua. Tuttavia, se questo stesso modello viene messo di fronte a compiti altamente specialistici — come l'interpretazione di dati di sequenziamento, l'analisi di varianti genetiche o la comprensione della letteratura bioinformatica — mostra subito dei limiti. Il pre-addestramento generico, infatti, non garantisce il livello di precisione e il rispetto delle regole rigide richiesti in ambito scientifico.

Il Fine-Tuning (o affinamento) è la strategia che permette di colmare questo divario. L'idea di fondo è semplice: invece di addestrare un modello da zero — un'operazione che richiede mesi di calcolo e budget milionari — si prende il modello base già esistente e lo si sottopone a una seconda fase di apprendimento guidato, utilizzando un set di dati più piccolo ma estremamente curato e focalizzato sullo specifico dominio di interesse (Howard & Ruder, 2018).

3.1 Full Fine-Tuning

Il Full Fine-Tuning (ovvero l'affinamento completo) rappresenta l'approccio originale e più profondo per adattare un modello di linguaggio a un nuovo compito. Per capire come

funziona, bisogna immaginare un LLM come una gigantesca rete di connessioni interne, dove miliardi di parametri regolano il flusso delle informazioni. Nella modalità Full Fine-Tuning, l'intero sistema di connessioni rimane "sbloccato": nessun parametro viene congelato.

Come Funziona il Processo di Apprendimento

Durante questa fase, il modello legge i testi forniti nel dataset specialistico ed effettua una serie continua di tentativi di previsione. Ogni volta che la risposta generata dal modello si discosta da quella corretta o desiderata, il sistema calcola la dimensione dell'errore commesso. Questo segnale di errore viene poi inviato a ritroso lungo tutta la struttura del modello.

Attraverso questo flusso all'indietro, l'ottimizzatore analizza il contributo di ciascun singolo parametro all'errore finale e ne modifica leggermente il valore. Poiché l'aggiornamento coinvolge contemporaneamente ogni singolo strato dell'architettura, l'intero modello riorganizza la propria struttura interna per sintonizzarsi sulle regole, sul lessico e sui ragionamenti tipici del nuovo contesto scientifico (Raffel et al., 2019).

I Vantaggi di un Adattamento Totale

L'aggiornamento dell'intera matrice di parametri offre vantaggi significativi sul piano delle prestazioni:

- Ristrutturazione delle rappresentazioni interne: Il modello non si limita a memorizzare nuovi vocaboli, ma ridisegna il modo in cui mappa i concetti complessi. In bioinformatica, questo gli permette di cogliere le sottili relazioni logiche tra sequenze geniche, strutture proteiche e percorsi metabolici.
- Padronanza di formati rigidi e sintassi complesse: Quando il modello deve imparare a generare output che non ammettono errori di sintassi — come codice in Python o R per la genomica, oppure file strutturati per la modellistica molecolare — l'intervento a tutto campo garantisce un livello di aderenza formale difficilmente raggiungibile con interventi più superficiali (Raffel et al., 2019).

I Grandi Limiti Operativi del Full Fine-Tuning

Nonostante l'elevata accuratezza, il Full Fine-Tuning presenta ostacoli pratici enormi che lo rendono quasi inaccessibile per la maggior parte dei laboratori e dei centri di ricerca universitari:

1. Il problema della memoria hardware: Modificare tutti i parametri del modello richiede una quantità di memoria grafica (VRAM) sproporzionata. La scheda video non deve ospitare soltanto il modello base, ma deve allocare uno spazio enorme per memorizzare i valori di errore calcolati a ogni passaggio, le informazioni storiche necessarie all'ottimizzatore e lo stato di ogni connessione aggiornata. Di conseguenza, per fare l'affinamento completo di un modello da 7 miliardi di parametri — considerata oggi una taglia medio-piccola — non bastano le schede video commerciali, ma servono server dedicati con cluster di GPU ad altissime prestazioni (Rajbhandari et al., 2019).
2. Dimenticanza catastrofica (*Catastrophic Forgetting*): È un rischio concreto e insidioso. Modificando indiscriminatamente l'intera rete di parametri, il modello corre il pericolo di "sovrascrivere" o cancellare gran parte delle conoscenze generali apprese durante la

prima fase di addestramento. Un LLM sottoposto a Full Fine-Tuning su dati di trascrittomica, ad esempio, potrebbe diventare impeccabile nell'analizzare quel tipo di tabelle, ma perdere completamente la capacità di ragionamento logico generale o la fluidità nella scrittura del testo (French, 1999).

3. Incapacità di scalare e problemi di gestione: Se un gruppo di ricerca ha bisogno di sviluppare tre strumenti differenti — per esempio uno dedicato alla genomica, uno alla bioinformatica strutturale e uno al riassunto di paper scientifici — con il Full Fine-Tuning è costretto a creare e salvare tre copie intere e distinte dell'azienda di parametri. Ogni copia richiede decine o centinaia di gigabyte di spazio sui server, rendendo la gestione delle infrastrutture estremamente costosa e complessa.

3.2 LoRA: LOW-RANK ADAPTATION

Per superare i pesanti colli di bottiglia computazionali del Full Fine-Tuning, la ricerca ha sviluppato una famiglia di tecniche note come *Parameter-Efficient Fine-Tuning* (PEFT). Tra queste, la metodologia che ha rivoluzionato il settore è senza dubbio LoRA (*Low-Rank Adaptation*, ovvero adattamento a basso rango) (Hu et al., 2021a).

L'idea fondamentale di LoRA è tanto semplice quanto geniale: anziché sbloccare e modificare i miliardi di parametri della rete neurale originale, il modello di partenza viene completamente "congelato" (*frozen*). Durante la fase di addestramento specialistico, le connessioni originali rimangono intatte e immutabili, mentre le modifiche necessarie per apprendere il nuovo compito vengono calcolate e salvate a parte in piccole strutture ausiliarie affiancate al modello principale.

Il Principio Concettuale della Decomposizione

Per comprendere come funziona LoRA senza addentrarsi nelle equazioni matematiche, occorre visualizzare l'addestramento come un processo di aggiornamento delle conoscenze. In un affinamento tradizionale, quando il modello impara un nuovo argomento, crea una grande "mappa di aggiornamento" con la stessa identica dimensione e complessità della rete di partenza.

Gli ideatori di LoRA sono partiti da un'osservazione empirica fondamentale: quando un modello di grande taglia impara un nuovo compito specifico, l'informazione *davvero nuova* non ha bisogno di occupare tutta la capacità della rete originale. In realtà, le modifiche effettive si concentrano in uno spazio molto più ristretto e semplificato (un concetto noto in algebra come "basso rango intrinseco") (Aghajanyan et al., 2020).

Invece di costruire una gigantesca mappa di aggiornamento piena di valori ridondanti, LoRA scompone l'aggiornamento in due passaggi sequenziali molto più leggeri attraverso due piccole matrici ausiliarie (comunemente chiamate matrice A e matrice B):

1. La prima matrice prende l'informazione in ingresso e ne riduce drasticamente la dimensione, "comprimendola" in un collo di bottiglia stretto.

2. La seconda matrice prende questo flusso compresso e lo ri-espande alla dimensione originale, proiettando il risultato finale all'uscita della rete.

Moltiplicando l'effetto di queste due piccole strutture trasversali, LoRA riesce a produrre un aggiornamento del tutto equivalente a quello di una mappa grande, ma utilizzando soltanto una frazione infinitesima di parametri (spesso meno dello 0,1% dell'intero modello) (Hu et al., 2021a).



"Immagine generata dall'IA"

I Parametri Chiave: Il Rango (r) e il Fattore di Scala (α)

Quando si configura un addestramento con LoRA, il ricercatore deve impostare due iperparametri fondamentali che regolano il comportamento di questo adattamento:

- Il Rango (r): Rappresenta l'ampiezza del "collo di bottiglia" attraverso cui viene compressa l'informazione. Un valore di rango basso (come 4 o 8) costringe il modello a imparare solo i pattern più generali e sintetici del nuovo dominio. Un valore di rango più alto (come 32 o 64) concede più spazio al modello per catturare sfumature complesse e dettagli sintattici minimi. Valori compresi tra 8 e 16 rappresentano solitamente il punto di equilibrio ideale per compiti di elaborazione del testo scientifico o bioinformatico (Hu et al., 2021a).

- Il Fattore di Scala Alpha (α): Agisce come un vero e proprio "manopola del volume" per l'adattamento. Controlla quanto le nuove informazioni apprese da LoRA debbano incidere sul comportamento finale del modello rispetto alle sue conoscenze originali congelate. In genere, si imposta un valore di α pari o doppio rispetto al rango scelto (ad esempio $\alpha = 16$ se $r = 8$), garantendo così che l'addestramento rimanga stabile anche se si modificano i tassi di apprendimento (*learning rate*).

I Vantaggi Operativi rispetto al Full Fine-Tuning

L'introduzione di LoRA garantisce benefici straordinari dal punto di vista pratico e computazionale:

1. Abbattimento Drastico dell'Uso di VRAM: Poiché l'ottimizzatore deve calcolare i gradienti e aggiornare lo stato di una manciata di parametri (le sole matrici ausiliarie), il consumo di memoria grafica crolla drasticamente. Questo permette di addestrare modelli da 7 o 13 miliardi di parametri su singole GPU di fascia commerciale o universitaria, senza la necessità di accedere a costosi cluster aziendali (Hu et al., 2021a).

2. Zero Latenza Aggiuntiva in Fase di Inferenza: A differenza di altri metodi PEFT che rallentano la risposta del modello perché aggiungono strati di calcolo durante l'uso, con LoRA è possibile "fondere" (*merge*) le piccole matrici di adattamento direttamente nei pesi originali prima della pubblicazione. Il modello finale risultante ha le stesse identiche dimensioni di quello di partenza e risponde con la medesima velocità.

3. Prevenzione della Dimenticanza Catastrofica: Mantenendo congelato il modello base, le conoscenze generali acquisite durante il pre-addestramento rimangono intatte e protette. Il modello acquisisce la specializzazione richiesta senza perdere la capacità di ragionamento di base.

4. Flessibilità e Modularità nell'Archiviazione: I file contenenti i pesi di un adattatore LoRA occupano comunemente solo poche decine di megabyte (contro i 14–30 gigabyte di un modello completo). Un laboratorio di ricerca può quindi mantenere un unico modello base immutato sui propri server e caricare "al volo" adattatori LoRA differenti a seconda del compito richiesto: uno per la trascrittomica, uno per la bioinformatica strutturale e uno per la stesura di articoli (Dettmers et al., 2023).

4. LA FAMIGLIA DEI MODELLI QWEN 2.5

Nel panorama dell'intelligenza artificiale generativa e dei modelli a pesi aperti (*open-weight*), la release della suite Qwen 2.5 sviluppata da Alibaba Cloud rappresenta uno dei traguardi ingegneristici più avanzati e significativi (Qwen et al., 2024a). Rispetto alle generazioni precedenti, la serie 2.5 non si limita a proporre un incremento quantitativo della scala dei parametri, ma introduce un raffinamento architetturale e metodologico strutturato per colmare in modo definitivo il divario con le soluzioni proprietarie di classe enterprise. Per questa ragione, la famiglia Qwen 2.5 si configura come la piattaforma ideale su cui innestare le procedure di adattamento specialistico e fine-tuning a basso rango descritte in questo lavoro di tesi.

Pre-Addestramento su Scala Multitrilione e Qualità dei Dati

Il fattore primario che determina le elevate capacità di ragionamento e di generalizzazione di Qwen 2.5 risiede nella natura del suo processo di pre-addestramento (Qwen et al., 2024a). Il modello fondazionale è stato alimentato con un corpus di dati imponente che supera i 18 mila miliardi di token (*18 trillion tokens*), segnando un incremento sostanziale rispetto ai dataset utilizzati per la generazione Qwen 2 (Yang et al., 2024).

La composizione di questo corpus non si distingue solo per la quantità, ma soprattutto per la rigorosa curatela qualitativa:

- Conoscenza scientifica e accademica: Ampie porzioni del dataset includono letteratura scientifica, archivi di preprint, documentazione tecnica e testi universitari specialistici.
- Codice e sintassi di programmazione: La presenza massiccia di codice sorgente multilingua ha potenziato le abilità di tracciamento delle istruzioni, di strutturazione logica e di elaborazione di algoritmi complessi.
- Dati strutturati e multilinguismo: Il modello ha sviluppato una naturale fluidità in oltre 29 lingue, mantenendo un'elevata sensibilità verso contesti sintattici articolati e linguaggi di markup strutturati.

Questa solida base di conoscenze pregresse garantisce che il modello possieda già una rappresentazione ricca dei concetti generali prima ancora di essere sottoposto a processi di affinamento verticale.

Gestione del Contesto Esteso e Architettura di Attenzione

Un altro pilastro innovativo della suite Qwen 2.5 è la capacità di gestire finestre di contesto di dimensioni straordinarie senza incorre nella degradazione delle prestazioni o nel fenomeno noto come "smarrimento nel mezzo" (*lost in the middle*) (Yang et al., 2025). Grazie all'implementazione di meccanismi avanzati di posizionamento dei token e al perfezionamento dell'attenzione con codifica posizionale rotatoria (*Rotary Position Embedding*, RoPE), i modelli della serie riescono a elaborare nativamente sequenze che vanno da 32.000 fino a oltre 128.000 token, con estensioni sperimentali che raggiungono il milione di token (Yang et al., 2025).

Per l'ambito di ricerca specialistico e bioinformatico affrontato in questa tesi, la disponibilità di una finestra di contesto così ampia offre vantaggi operativi decisivi:

1. Permette di inserire direttamente nel prompt articoli scientifici integrali, inclusi metodi, risultati e discussioni, senza dover ricorrere a frammentazioni o sintesi preventive.
2. Consente il caricamento di voluminosi database di annotazioni, sequenze genomiche o protocolli di laboratorio complessi all'interno di una singola sessione di lavoro.
3. Garantisce un tracciamento accurato delle dipendenze a lungo raggio nei compiti di estrazione di informazioni e sintesi documentale.

Granularità della Suite: Varianti e Scaling

La famiglia Qwen 2.5 è stata progettata con una struttura modulare e scalabile, offrendo soluzioni che coprono l'intero spettro delle esigenze computazionali, dalle applicazioni su dispositivi mobili fino ai cluster di calcolo ad alte prestazioni (Qwen et al., 2024a):

- I Modelli Compatti (0.5B, 1.5B e 3B): Progettati per scenari *edge* o ambienti ad altissima efficienza energetica, questi modelli offrono una velocità d'esecuzione

sorprendente e sono ideali per compiti di pre-elaborazione rapida, classificazione o estrazione di entità elementari.

- I Modelli Intermedi (7B e 14B): Rappresentano il punto di equilibrio perfetto per la ricerca scientifica e le applicazioni industriali su singola GPU. Le varianti da 7 e 14 miliardi di parametri combinano spiccate capacità di ragionamento con un'impronta di memoria contenuta, posizionandosi come i candidati ideali per le tecniche di adattamento a basso rango come LoRA e QLoRA.
- I Modelli di Grande Taglia (32B e 72B): Rappresentano l'eccellenza della serie per compiti di ragionamento complesso. Il modello da 32B ha dimostrato di poter eguagliare o superare architetture significativamente più estese nei benchmark di logica e codifica, mentre la variante da 72B si attesta ai vertici mondiali della categoria open-source.
- Le Declinazioni Specializzate: La suite comprende anche modelli ottimizzati per domini specifici, quali *Qwen 2.5-Coder* per l'ingegneria del software, *Qwen 2.5-Math* per la risoluzione di problemi matematici e simbolici, e architetture a miscela di esperti (*Mixture-of-Experts*, MoE) orientate al ragionamento multilivello.

Integrazione e Ruolo Strategico nel Progetto Tesi

La scelta di adottare la famiglia Qwen 2.5 come fondamento per il lavoro sperimentale di questa tesi si basa sulla scalabilità della famiglia e quindi della presenza di un gran numero di modelli di piccole dimensioni che permettono un'analisi in modo semplice e veloce.

Applicando la tecnica LoRA a un modello base della serie Qwen 2.5, la rete originale viene mantenuta congelata, preservandone intatta la struttura conoscitiva e la capacità di ragionamento. In questo lavoro, per garantire la piena compatibilità con l'esecuzione su CPU, i modelli sono stati impiegati nella loro precisione nativa a 32 bit. Gli adattatori a basso rango vengono poi addestrati in modo mirato sul nostro dataset di dominio, permettendo al sistema di acquisire il lessico specialistico, le sfumature stilistiche e la precisione concettuale richieste, senza dover modificare i miliardi di parametri originali. Questo approccio rende il ciclo di specializzazione estremamente rapido, sostenibile dal punto di vista energetico e accessibile anche in ambienti informatici a risorse limitate, dimostrando la concreta fattibilità di produrre modelli altamente specialistici a partire da grandi architetture open-source.

II. MATERIALI E METODOLOGIA

1.DATASET

Per una buona riuscita di un fine-tuning di questo genere è fondamentale possedere un set di dati (dataset) quanto più completo e strutturato possibile. Logicamente, considerando la tipologia di ricerca effettuata, la dimensione del dataset e la quantità di argomenti trattati, devono essere accurate e molto selettive. Per quanto la dimensione del dataset possa sembrare un qualcosa di trascurabile in relazione alla "pesantezza" del lavoro, in realtà è un fattore che

influisce in modo significativo. Si consideri, infatti, che la maggior parte del tempo impiegato durante un addestramento deriva principalmente dalla lunghezza del suddetto dataset e dal numero di volte che il modello dovrà "leggere" gli esempi contenuti in quest'ultimo.

Come detto in precedenza, la completezza e la struttura del dataset vanno di pari passo. Infatti, per un dataset che sia utile al fine di un Fine-tuning, c'è bisogno di impostarlo in modi specifici. Nel caso di studio in questione, è stata utilizzata la struttura più classica e utilizzata, ovvero: "INSTRUCTION (INPUT) – OUTPUT ". Sotto la voce INSTRUCTION si inserisce il quesito, mentre sotto la voce OUTPUT trova spazio la relativa risposta. Questa strutturazione permette al modello, durante l'addestramento, di leggere in sequenza domande e rispettive risposte in modo da apprenderle, al fine di imparare a replicare successivamente ai quesiti dell'utente.

Da ciò si può dedurre quanto sia cruciale la costruzione di un dataset forte e completo. Durante la fase di addestramento di un modello di IA, la fase cruciale è proprio quella della lettura dei dati. Un dataset strutturato male porterebbe inevitabilmente ad un modello che risponderebbe in modo errato ai quesiti dell'utente. A influenzare l'efficacia di un fine tuning è anche il numero di "passaggi" o "letture" del dataset che il modello esegue. Così come accade per un essere umano, anche i modelli di IA hanno bisogno di leggere più volte un testo per apprenderlo ed eventualmente commentarlo.

1.1 Domini Bioinformatici

Il dataset utilizzato durante questo progetto è composto da 500 esempi, tutti in ambito bioinformatico. Il set di dati è principalmente di natura teorica, arricchito però da diverse domande di "logica" che insegnano al modello a ragionare con i dati teorici acquisiti. Ciò al fine di dargli modo di poter rispondere anche a domande affini agli argomenti presenti ma che il modello non ha mai letto.

I principali argomenti presenti nel dataset sono:

- Algoritmi e Teoria dell'Allineamento
  Modelli di Allineamento: Differenze tra allineamento globale (Needleman-Wunsch) e locale (Smith-Waterman).
  Strategie di Scoring: Utilizzo delle matrici di sostituzione (PAM, BLOSUM), logica dei sistemi di punteggio logaritmici e gestione dei gap (penalità lineari vs. affini).
  Euristiche di Ricerca: Funzionamento del software BLAST, interpretazione dei parametri statistici (E-value, Bit Score) e concetti di Word Size e HSP (High-scoring Segment Pair).

- Genomica Strutturale e Funzionale
  Annotazione del Genoma: Identificazione di Open Reading Frames (ORF)
  Caratteristiche Sequenziali: Analisi del contenuto GC, GC-skew, Codon Usage Bias e regioni a bassa complessità.
  Sintenia ed Evoluzione Genomica: Studio della conservazione dell'ordine genico, pseudogeni, trasposoni e meccanismi di duplicazione (ortologia vs. paralogia).

- Tecnologie di Sequenziamento (NGS) e Data Processing
  Pipeline di Analisi: Controllo qualità (Phred scores, FASTQC), rimozione di adattatori (trimming) e normalizzazione dei dati.
  Mappatura e Assemblaggio: Allineamento delle reads su genoma di riferimento (mapping) vs. assemblaggio de novo, concetti di contig, scaffold e profondità di copertura (depth).
  Standard Informatici: Formati file per l'archiviazione di sequenze e allineamenti (FASTA, FASTQ, SAM, BAM, VCF, GFF3, BED).

- Trascrittomica e Analisi dell'Espressione Genica
  RNA-seq: Quantificazione dei trascritti, analisi dell'espressione differenziale, interpretazione di Volcano Plot e Heatmap.
  Unità di Misura: Metodi di normalizzazione per la lunghezza dei geni e la profondità di libreria (RPKM, TPM).
  Splicing: Identificazione di varianti di splicing alternativo e mappatura splice-aware.
  Single-cell RNA-seq: Introduzione alla risoluzione cellulare e al concetto di pseudotime.

- Filogenetica Molecolare
  Ricostruzione Evolutiva: Algoritmi per la creazione di alberi filogenetici, distinzione tra alberi con o senza radice e calcolo del supporto dei rami tramite Bootstrap.
  Dinamiche Evolutive: Concetto di orologio molecolare, evoluzione neutrale e attrazione dei rami lunghi (Long Branch Attraction).

- Bioinformatica Strutturale e Proteomica
  Analisi 3D: Formato PDB, coordinate atomiche, visualizzazione molecolare e calcolo dell'RMSD per la sovrapposizione strutturale.
  Proprietà Proteiche: Domini funzionali, motivi conservati, grafici di idropaticità e strutture secondarie (alfa-eliche e foglietti beta).
  Interazioni: Principi di Molecular Docking e legami proteina-ligando.

- Risorse Bioinformatiche e Database
  Repository Primari e Secondari: Differenze tra GenBank, RefSeq, UniProt e PDB.
  Ontologie e Pathway: Utilizzo di Gene Ontology (GO) per l'annotazione funzionale e KEGG per la mappatura dei pathway metabolici.
  Database Clinici: Consultazione di risorse per la medicina di precisione come ClinVar, gnomAD e OMIM.

- Bioinformatica Medica e Clinica
  Variant Calling: Identificazione di SNP, varianti strutturali (SV) e mutazioni de novo.
  Diagnostica Molecolare: Differenza tra varianti germinali e somatiche, interpretazione dell'impatto funzionale delle varianti e analisi di trio familiari.

- Metodologie Computazionali e Metagenomica
  Ambienti di Lavoro: Utilizzo della riga di comando (Bash/Linux), linguaggi di programmazione (Python, R) e sistemi di containerizzazione (Docker).

Analisi di Comunità: Principi di metagenomica shotgun e studio del microbioma per l'analisi di campioni ambientali complessi.

1.2 Esempi di domande

Qui sotto alcuni esempi di domande presenti nel dataset utilizzato per il Fine-Tuning:

- {"instruction": "Spiega la differenza tra un gap 'lineare' e un gap 'affine'. Quale modella meglio la biologia?", "output": "Il gap lineare assegna lo stesso costo a ogni base mancante. Il gap affine assegna un costo alto per aprire il gap e uno basso per estenderlo. Il modello affine è biologicamente superiore perché riflette il fatto che un singolo evento mutazionale può facilmente eliminare o inserire più basi contemporaneamente."}
- {"instruction": "Cos'è un 'Ortholog' e perché è la base per l'annotazione funzionale?", "output": "Gli ortologhi sono geni in specie diverse che derivano da un antenato comune in seguito a speciazione. Poiché solitamente mantengono la stessa funzione originale, identificare un ortologo tra una specie ignota e una ben studiata (come il lievito o l'uomo) permette di trasferire l'annotazione funzionale con alta confidenza."}
- {"instruction": "In un allineamento multiplo, cosa rappresenta una regione 'altamente conservata'?", "output": "Rappresenta solitamente un dominio funzionale o strutturale critico, come un sito catalitico enzimatico o un'interfaccia di legame. La selezione naturale ha eliminato gli individui con mutazioni in queste posizioni, indicando che quei residui sono indispensabili per la sopravvivenza della cellula."}
- {"instruction": "In un albero filogenetico, cos'è un 'Nodo Interno'?", "output": "Rappresenta l'ultimo antenato comune ipotetico tra le sequenze o le specie che si diramano da quel punto. La distanza tra i nodi indica il tempo evolutivo o il numero di cambiamenti accumulati dalla divergenza."}
- {"instruction": "Qual è il vantaggio di usare matrici BLOSUM rispetto alle matrici PAM?", "output": "Le matrici PAM si basano su un modello evolutivo di mutazioni puntiformi globali. Le matrici BLOSUM sono basate su allineamenti locali di blocchi conservati, risultando più efficaci nel trovare somiglianze funzionali distanti che non coinvolgono l'intera lunghezza della proteina."}
- {"instruction": "Cos'è un 'Database Primario'?", "output": "È un archivio che raccoglie i dati originali caricati dai laboratori di tutto il mondo, come GenBank."}
- {"instruction": "Perché il 'Bit Score' è meglio del punteggio grezzo?", "output": "Perché permette di confrontare risultati di ricerche diverse in modo equo e standardizzato."}
- {"instruction": "Perché BLAST è più veloce dell'algoritmo di Smith-Waterman?", "output": "Perché BLAST è un algoritmo euristico: non confronta ogni singola base, ma cerca prima 'parole' (match esatti corti) e le estende solo se promettenti. Smith-Waterman è esatto ma troppo lento per database di grandi dimensioni."}
- {"instruction": "Descrivi la struttura del formato file FASTA.", "output": "Il formato FASTA inizia con una riga di intestazione che comincia col simbolo '>' seguito dal nome della sequenza. Le righe successive contengono la sequenza di nucleotidi o amminoacidi in formato testuale."}

2. SETUP SPERIMENTALE

In questo paragrafo viene descritto l'hardware utilizzato sia per la fase di addestramento sia per quella di benchmarking dei modelli presi in esame. Come anticipato nel capitolo introduttivo, l'architettura di riferimento è rappresentata dalla famiglia di LLM Qwen2.5, di cui sono stati selezionati quattro modelli a dimensione parametrica crescente: Qwen2.5-0.5B, Qwen2.5-1.5B, Qwen2.5-3B e Qwen2.5-7B.

La scelta dei modelli risponde a una precisa esigenza metodologica: per effettuare un'analisi rigorosa è fondamentale un'accurata selezione che metta a confronto modelli di differenti dimensioni parametriche ma che siano stati programmati e sviluppati con gli stessi criteri. In particolare, la famiglia di modelli Qwen2.5 è stata selezionata in virtù dei seguenti fattori metodologici e pratici:

- Omogeneità: la disponibilità di quattro varianti scalari (0.5B, 1.5B, 3B, 7B) con architettura omogenea ha permesso di analizzare l'effetto della dimensione del modello in modo controllato, isolando questa variabile da differenze architetturali.
- Riproducibilità: la licenza aperta (Apache 2.0) e la disponibilità pubblica dei pesi su Hugging Face garantiscono la piena riproducibilità degli esperimenti, allineandosi ai principi di Open Science.
- Rendimenti di calcolo: l'efficienza computazionale di Qwen2.5, con supporto nativo a precisione float16 e ottimizzazioni per inference su CPU, lo rende adatto a contesti accademici con risorse hardware limitate.
- Solidità delle prestazioni di base: le elevate performance dei modelli base sui benchmark generali assicurano che eventuali limiti nelle risposte bioinformatiche siano attribuibili alla mancanza di conoscenza di dominio, e non a carenze linguistiche sottostanti.
- Integrabilità: La piena compatibilità con le librerie standard per il fine-tuning efficiente (PEFT/LoRA) ha semplificato l'implementazione sperimentale.

Nel loro complesso, questi criteri rendono Qwen2.5 una scelta metodologicamente solida per condurre un'analisi comparativa di fine-tuning in bioinformatica.

2.1 Hardware e Software

La scelta dell'hardware non è stata casuale, in quanto si voleva capire se e quanto fosse stato possibile sviluppare dei modelli utilizzabili in locale (ovvero senza utilizzare server esterni per usufruire dei servizi forniti da un modello di IA) senza alcuna limitazione in termini di utilizzo.

La macchina impiegata per le sperimentazioni è dotata delle seguenti specifiche:

- CPU AMD RYZEN 5 5600X, con 6 core (unità di elaborazione fisica che esegue le istruzioni) e 12 thread (unità logica di esecuzione che permette ad un core di alternare o parallelizzare più flussi di istruzione);

- Memoria RAM 32 GB (ormai presenti nella maggior parte di PC per uso domestico) considerata come soglia minima per poter caricare in memoria i pesi di modelli fino a 7 miliardi di parametri;

- Acceleratori grafici: non è stata utilizzata alcuna GPU, né per il training nè per il benchmark, al fine di ridurre il più possibile i costi dell'hardware ed approssimarsi il più possibile alle configurazioni a disposizione dell'utente medio.

Per quanto riguarda la dotazione software:

- Sistema Operativo: Windows 11 Pro (OS più diffuso tra gli utenti);

- Linguaggio di Programmazione: Python, selezionato sia per la chiarezza sintattica sia, soprattutto, perché tutti i principali framework di AI e Deep Learning sono progettati *Python-first* (ciò significa che una libreria è concepita pensando a Python come lingua principale).

2.2 Configurazione LoRA: Fase di TRAINING

Entriamo ora nella fase operativa e, dunque, cruciale del progetto che, di seguito, permetterà di vedere dettagliatamente la struttura dello script di addestramento utilizzato per eseguire il fine-tuning dei quattro i modelli della famiglia QWEN. Logicamente, lo *script di training* è stato mantenuto identico per tutti i modelli al fine di garantire la massima imparzialità sperimentale e ottenere risultati realistici durante la successiva fase di valutazione e confronto dei modelli.

Per l'adattamento dei modelli della famiglia Qwen 2.5 alle specifiche esigenze del dominio applicativo, è stata implementata una procedura di fine-tuning basata sulla libreria Hugging Face Transformers in combinazione con il framework PEFT (Parameter-Efficient Fine-Tuning). L'intero ciclo di addestramento è stato ingegnerizzato per operare all'interno di un ambiente computazionale vincolato all'uso della CPU. Tale vincolo hardware ha imposto l'adozione di specifiche scelte architetturali e di ottimizzazione, sia a livello di gestione della memoria RAM, sia nella configurazione delle componenti di tokenizzazione, caricamento dei pesi e aggiornamento dei parametri.

La fase iniziale del flusso di lavoro prevede la lettura di un dataset memorizzato nel formato JSON Lines (.jsonl). Lo script implementa un meccanismo di ispezione dinamica della struttura dei dati per identificare automaticamente le coppie di chiavi dedicate al testo di input e di output. Nello specifico, il sistema ricerca coppie di campi convenzionali quali

- instruction / output

- prompt / response

- question / answer

Qualora non vengano rintracciate tali corrispondenze predefinite, l'algoritmo rileva ed estrae automaticamente le prime due chiavi di tipo stringa disponibili nel primo record del dataset.

Una volta individuate le chiavi di testo, l'insieme dei dati viene suddiviso in una porzione destinata all'addestramento, la cui dimensione è impostata tramite l'argomento --train_size (con un valore predefinito di 400 campioni), e in una porzione residua riservata alla successiva fase di test. Ciascun elemento del sottoinsieme di training viene quindi convertito in un prompt testuale unico, formattato secondo lo schema seguente:

Instruction:

<testo_di_input>

Response:

<testo_di_output>

I testi così preparati vengono convertiti in un dataset Hugging Face e sottoposti a tokenizzazione. In questa fase, il tokenizer associato al modello base viene caricato esclusivamente da file locali tramite l'argomento local_files_only=True, avvalendosi dell'implementazione rapida basata su Rust (use_fast=True). Se il tokenizer non specifica un token di padding dedicato (pad_token), lo script imposta la coincidenza tra quest'ultimo e il token di fine sequenza (eos_token).

La funzione di tokenizzazione mappa il testo troncandolo alla lunghezza massima stabilita dall'argomento --max_seq_len (pari a 256 token) e applicando un riempimento fisso fino a tale limite (padding="max_length"). Ai fini dell'addestramento per la modellazione linguistica causale (Causal Language Modeling), le sequenze di ID ottenute (input_ids) vengono duplicate all'interno del campo labels, fornendo così la struttura target necessaria al calcolo della perdita (loss) da parte del DataCollatorForLanguageModeling.

Il caricamento del modello di linguaggio base avviene tramite l'interfaccia AutoModelForCausalLM. Per garantire l'esecuzione corretta e stabile su architettura CPU, vengono adottate configurazioni puntuali:

Python

```
model = AutoModelForCausalLM.from_pretrained(
    base_model_dir,
    dtype=torch.float32,
    device_map=None,
    local_files_only=True,
    low_cpu_mem_usage=True,
)
```

model.to("cpu")

model.config.use_cache = False

La scelta di impostare dtype=torch.float32 assicura la compatibilità nativa dei calcoli in virgola mobile a 32 bit sull'hardware della CPU, evitando errori di precisione e i rallentamenti tipici dei formati ridotti. Inoltre, l'impostazione low_cpu_mem_usage=True evita che il sistema duplichi i pesi del modello in RAM durante il caricamento, riducendo al minimo i picchi di memoria.

Un passaggio chiave riguarda la disattivazione del meccanismo di caching interno tramite l'istruzione model.config.use_cache = False. Mentre il KV-caching risulta fondamentale nell'ottimizzazione della velocità di generazione durante la fase di inferenza, la sua disattivazione è indispensabile nel corso dell'addestramento per consentire la corretta propagazione dei gradienti attraverso l'architettura. Parallelamente, lo script ottimizza le prestazioni di calcolo impostando il numero di thread di PyTorch (torch.set_num_threads) su un valore pari alla metà dei core CPU totali rilevati dal sistema tramite la formula max(1, os.cpu_count() // 2).

Per limitare drasticamente il numero di parametri da aggiornare ed evitare l'elevato costo computazionale del Full Fine-Tuning, viene applicata la tecnica LoRA (Low-Rank Adaptation). Lo script individua automaticamente a quali moduli applicare gli adattatori: scansiona la struttura della rete alla ricerca dei livelli chiave per l'attenzione e per le reti feed-forward (q_proj, k_proj, v_proj, o_proj, up_proj, down_proj, gate_proj). Se non trova tutti questi componenti, lo script applica un meccanismo di emergenza (fallback) limitandosi ai soli moduli q_proj e v_proj:

Python

```
def pick_lora_targets(model) -> List[str]:
candidates = [“q_proj”, “k_proj”, “v_proj”, “o_proj”, “up_proj”, “down_proj”, “gate_proj”]
present = set()
for name, _ in model.named_modules():
for c in candidates:
if name.endswith©:
present.add©
if not present:
return [“q_proj”, “v_proj”]
return sorted(present)
```

I moduli individuati vengono passati alla configurazione LoraConfig dell'infrastruttura PEFT, la cui istanziazione definisce puntualmente tutti gli iperparametri dell'adattatore:

Python

```
peft_cfg = LoraConfig(
```

```
task_type=TaskType.CAUSAL_LM,

r=args.lora_r,

lora_alpha=args.lora_alpha,

lora_dropout=args.lora_dropout,

target_modules=targets,

inference_mode=False,

)
```

I parametri specifici dell'adattatore sono regolabili da riga di comando e configurati di default sui seguenti valori espliciti:

- Rango delle matrici **(--lora_r = 8):** definisce il rango r delle matrici di decomposizione inserite nei moduli target.
- Fattore di scala **(--lora_alpha = 16):** rappresenta l'iperparametro α utilizzato per riscalare gli aggiornamenti dei pesi applicati al modello.
- Tasso di Dropout **(--lora_dropout = 0.05):** stabilisce la probabilità di dropout applicata agli strati LoRA per favorire la regolarizzazione ed evitare l'overfitting.
- Tipologia di Task **(TaskType.CAUSAL_LM):** definisce l'obiettivo di modellazione linguistica causale autoregressiva, mantenendo la modalità di addestramento attiva (inference_mode=False).

In questo modo, i parametri del modello base rimangono congelati e l'addestramento aggiorna esclusivamente la ridotta percentuale di parametri degli adattatori LoRA.

Il ciclo di addestramento viene eseguito tramite la classe Trainer di Hugging Face, la cui pianificazione è dettata dall'oggetto TrainingArguments:

Python

```
train_args = TrainingArguments(
    output_dir=output_lora_dir,
    per_device_train_batch_size=args.batch_size,
    gradient_accumulation_steps=args.grad_accum,
    num_train_epochs=args.epochs,
    learning_rate=args.learning_rate,
    logging_steps=args.logging_steps,
```

```
    save_steps=args.save_steps,
    save_total_limit=2,
    fp16=False,
    bf16=False,
    dataloader_num_workers=0,
    remove_unused_columns=False,
    report_to=[],
)
```

Tutti gli iperparametri e le metriche chiave dell'esecuzione sono specificati di default con i seguenti valori numerici:

- Numero di Epoche **(--epochs = 2.0):** l'addestramento viene eseguito per 2 passaggi completi sul sottoinsieme dei dati di training.
- Tasso di Apprendimento **(--learning_rate = 2e-4):** il valore del learning rate impostato per l'ottimizzatore è pari a 2×10^{-4}.
- Batch Size per Dispositivo **(--batch_size = 1):** la quantità di campioni elaborati per singolo passo elementare è limitata a 1 unità per contenere l'occupazione di memoria RAM su CPU.
- Passi di Accumulo del Gradiente **(--grad_accum = 8):** i gradienti vengono accumulati per 8 passaggi consecutivi, determinando una *batch size* efficace complessiva pari a $1 \times 8 = 8$ campioni prima di ogni aggiornamento effettivo dei pesi.
- Frequenza di Tracciamento **(--logging_steps = 10):** la registrazione delle metriche di perdita e stato dell'addestramento avviene ogni 10 passi.
- Frequenza e Limite di Salvataggio **(--save_steps = 100, save_total_limit = 2):** la creazione di un checkpoint del modello avviene ogni 100 passi di addestramento, mantenendo su disco un numero massimo di 2 salvataggi totali per evitare l'occupazione eccessiva dello spazio di archiviazione.
- Precisione Numerica **(fp16=False, bf16=False):** le opzioni di precisione ridotta sono disabilitate per garantire il calcolo a 32 bit nativo su CPU.
- Gestione dei Lavoratori del Dataloader **(dataloader_num_workers = 0):** il caricamento dei dati è eseguito nello stesso processo principale per evitare conflitti di multithreading.
- Integrazioni Esterne **(report_to=[]):** l'invio automatizzato di log verso piattaforme di terze parti è disattivato.

Al termine dell'esecuzione del comando trainer.train(), il modello adattato e il relativo tokenizer vengono salvati nella cartella di destinazione specificata (output_lora_dir). Infine,

viene generato il file di tracciamento split_meta.json contenente i metadati dell'esperimento, quali il numero esatto di campioni del train set (train_size), del test set (test_size), le chiavi dei campi identificati (input_key, output_key), oltre ai percorsi assoluti del modello base (base_model_dir) e del dataset (dataset_path).

2.3 Parametri di inferenza: Fase di Benchmark

La fase successiva l'addestramento dei vari modelli, secondo le modalità sopra descritte, si esplicita nella costruzione di uno script di benchmark (valutare le prestazioni di ogni singolo modello) al fine di confrontare i modelli "base" (o nativi), con i modelli "LoRA", che hanno assimilato i dati contenuti nel dataset.

Per rendere il processo più rapido e fluido, lo script automatizza l'intera valutazione caricando in memoria contemporaneamente sia il modello base sia la sua controparte LoRa.

In linea con l'addestramento, anche la fase di inferenza è eseguita interamente su CPU, senza fare ricorso ad acceleratori GPU.

Per tutte le taglie analizzate (0.5B, 1.5B, 3B e 7B), sia i modelli "base" (base_model) sia la versione fine-tuned (tuned_model) sono stati caricati in memoria utilizzando la precisione nativa a 32 bit (dtype=torch.float32), attingendo esclusivamente ai file salvati in locale (local_files_only=True) e sfruttato il parametro low_cpu_mem_usage=True per alleggerire l'impatto sulla RAM all'avvio.

Il caricamento degli adattatori presenta una differenza metodologica importante nello script del modello più grande, Qwen 2.5 7B:

- Modelli 0.5B, 1.5B e 3B: Gli adattatori LoRA vengono sovrapposti al modello base in memoria dinamica tramite la libreria PEFT, usando la funzione PeftModel.from_pretrained(). Durante la generazione, il modello calcola in parallelo i contributi della rete base e delle matrici a basso rango.
- Modello 7B: Per la versione da 7 miliardi di parametri, è stata invece applicata la fusione esplicita dei pesi attraverso l'istruzione tuned_model.merge_and_unload(). Questa operazione unisce matematicamente i pesi di LoRA direttamente nella struttura del modello base prima dell'inizio del test. Questa scelta ha permesso di ridurre notevolmente l'impatto sulla memoria RAM, evitando di dover gestire due grafi di calcolo distinti durante la generazione e rendendo l'inferenza molto più fluida per la taglia più pesante. Ovviamente questo non ha influito in alcun modo con le prestazioni del modello, lasciando la valutazione il quanto più imparziale possibile.

Per assicurare la massima coerenza con la fase di addestramento, ogni istruzione del test set (estratta a partire dall'indice --test_start_idx, impostato di default a 400) è stata formattata usando la stessa struttura. Questa formattazione ha permesso ai modelli di rispondere solo ed esclusivamente alle 100 domande lasciate "segrete", mai viste prima dai modelli. Questo procedimento permette di mettere praticamente i modelli "base" alla pari dei modelli "LoRA" mettendo entrambi davanti a dati del tutto inediti.

L'esecuzione della generazione è stata inserita all'interno del blocco @torch.no_grad(), disabilitando il calcolo dei gradienti per risparmiare risorse e velocizzare l'elaborazione.

I parametri impostati all'interno della funzione model.generate() sono stati mantenuti identici per tutte e quattro le taglie di modello, adottando una strategia di **Greedy Search deterministica**:

- do_sample = False: Disattiva il campionamento casuale dei token. Ad ogni passo di generazione, il modello sceglie sempre il token con la probabilità più alta, rendendo i risultati perfettamente riproducibili ad ogni esecuzione.
- temperature e top_p: Nello script del modello 0.5B la temperatura è impostata esplicitamente a 0.0, mentre per le altre taglie è stata omessa o affiancata da top_p = 1.0. Avendo impostato do_sample=False, questi parametri rimangono inattivi e non influenzano il processo di decodifica.
- max_new_tokens = 256: Stabilisce il numero massimo di nuovi token che il modello può generare per ciascuna risposta, evitando che la generazione prosegua all'infinito.
- Token di Stop e Padding (pad_token_id, eos_token_id): Entrambi i parametri sono stati assegnati all'ID del token di fine sequenza (tokenizer.eos_token_id)

Gli script includono un sistema di monitoraggio in tempo reale che calcola le tempistiche di esecuzione tramite la funzione fmt_seconds. A intervalli regolari, definiti dal parametro --log_every (impostato di default ogni 5 esempi), lo script stampa a schermo:

- Il tempo totale trascorso fino a quel momento (elapsed);
- Il tempo medio impiegato per elaborare un singolo esempio (avg/ex);
- Una stima del tempo rimanente alla fine del test (eta).

Al termine del test, tutti i dati vengono salvati nella cartella indicata da --output_dir (di default ./benchmark_results). La struttura dei file generati comprende:

- Summary JSON (<modello>_summary.json): Un file di riepilogo contenente le configurazioni usate (percorsi dei file, parametri di generazione, data e orario di esecuzione e tempo totale impiegato).
- CSV dettagliato (<modello>_per_example.csv): Una tabella contenente, per ciascun esempio, la domanda originale (question), la risposta di riferimento (reference) e le due risposte generate dal modello base (pred_base) e dal modello sintonizzato (pred_tuned).
- File JSON esteso per Qwen 7B (domande_risposte_qwen2.5-7b.json): Solo per il modello da 7B, lo script genera un ulteriore file JSON che raccoglie in modo strutturato le domande, le risposte ideali e i testi prodotti dai due modelli, rendendo più semplice l'ispezione manuale e l'analisi qualitativa delle risposte.

3. METRICHE DI VALUTAZIONE

Per misurare in modo oggettivo l'efficacia dei modelli e quantificare l'impatto dell'addestramento tramite LoRA rispetto alle rispettive baseline non sintonizzate, lo script di benchmark implementa due metriche chiave ampiamente diffuse nella letteratura del *Natural Language Processing*: l'**Exact Match (EM)** e il **Token-level F1 Score (F1)**.

Oltre a queste due metriche viene eseguita una valutazione semantica dei modelli in modo da poter calcolare effettivamente il numero di risposte corrette e il numero di risposte sbagliate.

3.1 Quantitativa: Token F1 Score, Exact Match (EM)

Entrambe le metriche lavorano confrontando la risposta generata dal modello (predizione) con la risposta ideale presente nel dataset di test (riferimento).

Prima di effettuare qualsiasi calcolo matematico, le stringhe di testo generato e quelle di riferimento vengono sottoposte a una fase di "pulizia" tramite la funzione normalize_text. Questa operazione è fondamentale per evitare che minime differenze (slegate dalla correttezza del contenuto) possano penalizzare i punteggi del modello.

Nello specifico, il processo di normalizzazione applica tre trasformazioni consecutive:

- Conversione in minuscolo: elimina la sensibilità alle maiuscole e minuscole (.lower());
- Rimozione della punteggiatura e dei caratteri speciali: vengono preservati unicamente le lettere, i numeri, gli spazi e le lettere accentate italiane, rimuovendo qualsiasi altro simbolo;
- Collasso degli spazi vuoti: riduce eventuali spazi multipli, tabulazioni o a capo ad un singolo spazio standard (\s+ → " ")

L'Exact Match rappresenta una metrica binaria e particolarmente severa. Essa verifica se, a seguito del processo di normalizzazione, la risposta prodotta dal modello coincide in modo identico e perfetto con il testo di riferimento:

$$\text{EM} = \begin{cases} 1.0 & \text{se normalize}(pred) = \text{normalize}(ref) \\ 0.0 & \text{altrimenti} \end{cases}$$

Se anche una singola parola differisce o se la struttura della frase varia pur mantenendo lo stesso significato semantico, il punteggio assegnato per quel singolo esempio sarà pari a zero. L'EM costituisce dunque un indicatore estremamente utile per misurare la capacità del modello di produrre risposte sintetiche, esatte e rigide (come entità specifiche, codici o dati).

Per compensare la rigidità dell'Exact Match e valutare in modo meno rigido la qualità delle risposte, viene impiegato il **Token-level F1 Score**. Questa metrica misura il grado di

sovrapposizione a livello di singole parole (token) tra la predizione e il riferimento, calcolandone la media armonica tra Precision e Recall.

Suddividendo la predizione P e il riferimento R nei rispettivi insiemi di parole, il sistema calcola:

- Precision (Precisione): la frazione di parole generate dal modello che sono effettivamente presenti nella risposta di riferimento.

$$\text{Precision} = \frac{|\ P \cap R\ |}{|\ P\ |}$$

- Recall (Richiamo): la frazione di parole della risposta di riferimento che il modello è riuscito a generare.

$$\text{Recall} = \frac{|\ P \cap R\ |}{|\ R\ |}$$

- F1 Score: la media armonica tra le due misure:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Qualora sia la predizione sia il riferimento risultino vuoti dopo la normalizzazione, il valore viene impostato convenzionalmente a 1.0; se invece non vi è alcuna parola in comune o uno dei due testi è vuoto, l'F1 sarà pari a 0.0.

A differenza dell'EM, l'F1 Score premia i modelli che riescono a catturare le informazioni chiave e i concetti corretti, anche se formulati con una struttura sintattica leggermente diversa da quella del dataset di test.

All'interno degli script di benchmark, entrambe le metriche vengono calcolate individualmente per ciascun esempio del test set (composto dagli esempi a partire dall'indice 400). Al termine dell'esecuzione, viene eseguita la media aritmetica dei punteggi ottenuti su tutti i campioni per calcolare l'**Exact Match Medio** e il **Token F1 Medio** sia del modello base sia del modello sintonizzato con LoRA.

Infine, per quantificare l'efficacia del fine-tuning, viene calcolato il delta di miglioramento (Δ):

$$\Delta\text{EM} = \text{EM}_{\text{tuned}} - \text{EM}_{\text{base}}$$

$$\Delta\text{F1} = \text{F1}_{\text{tuned}} - \text{F1}_{\text{base}}$$

Un valore di Δ positivo evidenzia direttamente l'apporto benefico dell'addestramento LoRA nel migliorare l'aderenza del modello alle istruzioni del dominio considerato.

3.2 Qualitativa: LLM-as-a-Judge

Con questa metodologia di valutazione non usiamo alcun tipo di script. Infatti, durante la fase di benchmark, è richiesto di "stampare" e salvare in un file JSON tutte le risposte generate dai vari modelli. Questo rende possibile la realizzazione di questa nuova fase di benchmark, probabilmente la più significativa.

Per giudicare i risultati viene utilizzata una intelligenza artificiale terza e imparziale che si occupa di valutare le risposte del singolo modello e decidere se considerarla corretta o meno in base alle proprie "conoscenze". Naturalmente, il modello utilizzato come giudice deve essere un modello generalista molto più potente dei modelli in esame, in questo caso la scelta è ricaduta su Gemini 3.6.

Leggendo il file JSON, Gemini analizza le risposte fornite ai singoli quesiti e le valuta unicamente dal punto di vista semantico (a differenza delle metriche puramente sintattiche viste nel paragrafo precedente).

Al modello giudice sono stati quindi forniti: un prompt che preparasse il modello ad eseguire il Benchmark, il file JSON del dataset, che permette di estrapolare le risposte direttamente dalla fonte di addestramento, questo ha permesso di avere un "Gold standard"; i file JSON dei modelli Base, per valutare la precisione e correttezza semantica dei modelli prima dell'affinamento; i file JSON dei modelli che hanno subito il Fine-Tuning.

III. RISULTATI SPERIMENTALI

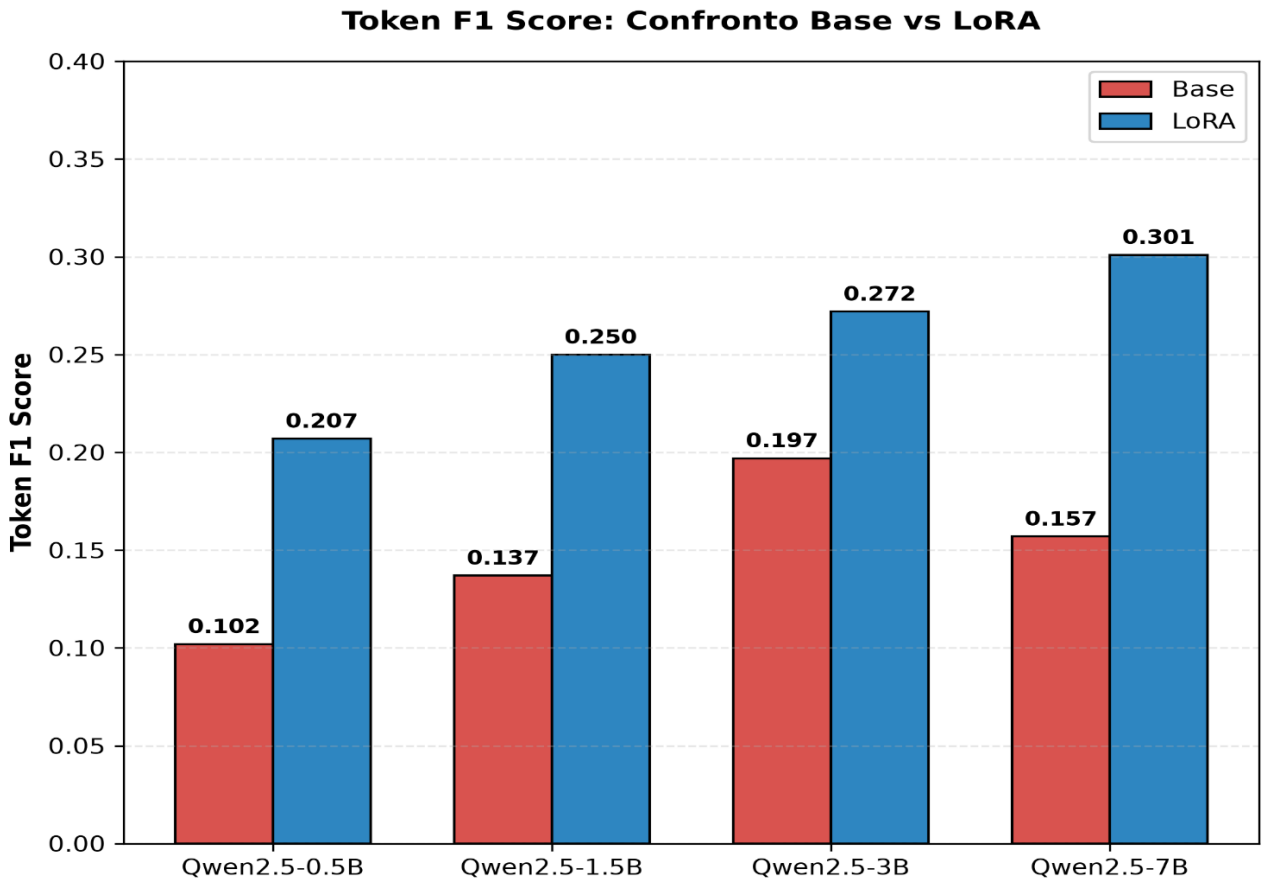
In questa fase andremo ad analizzare dettagliatamente e graficamente i risultati ottenuti dal processo di addestramento e dai processi di valutazione Quantitativa e Qualitativa, in modo da poter creare un quadro completo sull'utilità e sull'efficacia dell'intero progetto.

1. VALUTAZIONE QUANTITATIVA (TOKEN F1)

Utilizzando lo script automatizzato di Benchmark analizzato nel capitolo precedente, sono stati estrapolati gli score di tutti i modelli analizzati, quindi sia base che LoRA. Ovviamente in questa fase di benchmark, oltre che agli score, sono state estrapolate anche le domande e le rispettive risposte generate dai modelli. Questo ci permetterà successivamente di poter creare un archivio di domande e risposte a utilizzare nella seconda fase di benchmark.

1.1 Grafico a barre Base vs LoRA

Ora analizzeremo graficamente i risultati ottenuti dalla valutazione effettuata utilizzando il Token F1. Nel grafico si osserva: in rosso gli score dei modelli base, in blu gli score dei modelli che hanno subito l'affinamento. Possiamo immediatamente notare un miglioramento degno di nota da parte di tutti i modelli in esame



- +0.105 per il modello QWEN 2.5 0,5B;
- +0.113 per il modello QWEN 2.5 1.5B:
- +0.075 per il modello QWEN 2.5 3B;
- +0.144 per il modello QWN 2.5 7B;

Analizzando i risultati riportati in figura, si osserva un andamento apparentemente controintuitivo: il modello QWEN 2.5 7B Base ottiene un F1 Score pari a **0.157**, registrando una performance inferiore rispetto alla versione di taglia inferiore QWEN 2.5 3B Base, la quale si attesta a **0.197**. Sebbene i modelli dotati di un numero maggiore di parametri vantino generalmente una capacità di rappresentazione superiore, tale inversione di tendenza nei modelli di base può essere attribuita a diversi fattori:

- Sensibilità al prompt e verbosità non controllata**:** I modelli appartenenti alla classe *Base* non sono ottimizzati per seguire rigorosamente direttive o formati strutturati. Architetture più grandi, come quella da 7 miliardi di parametri, tendono a manifestare una maggiore propensione alla generazione. Qualora il task richieda un'estrazione mirata o un output rigidamente formattato, l'assenza di un allineamento specifico induce il modello a produrre risposte estese, penalizzando drasticamente la precisione e il richiamo (*Recall*), e di conseguenza il *Token F1 Score*.

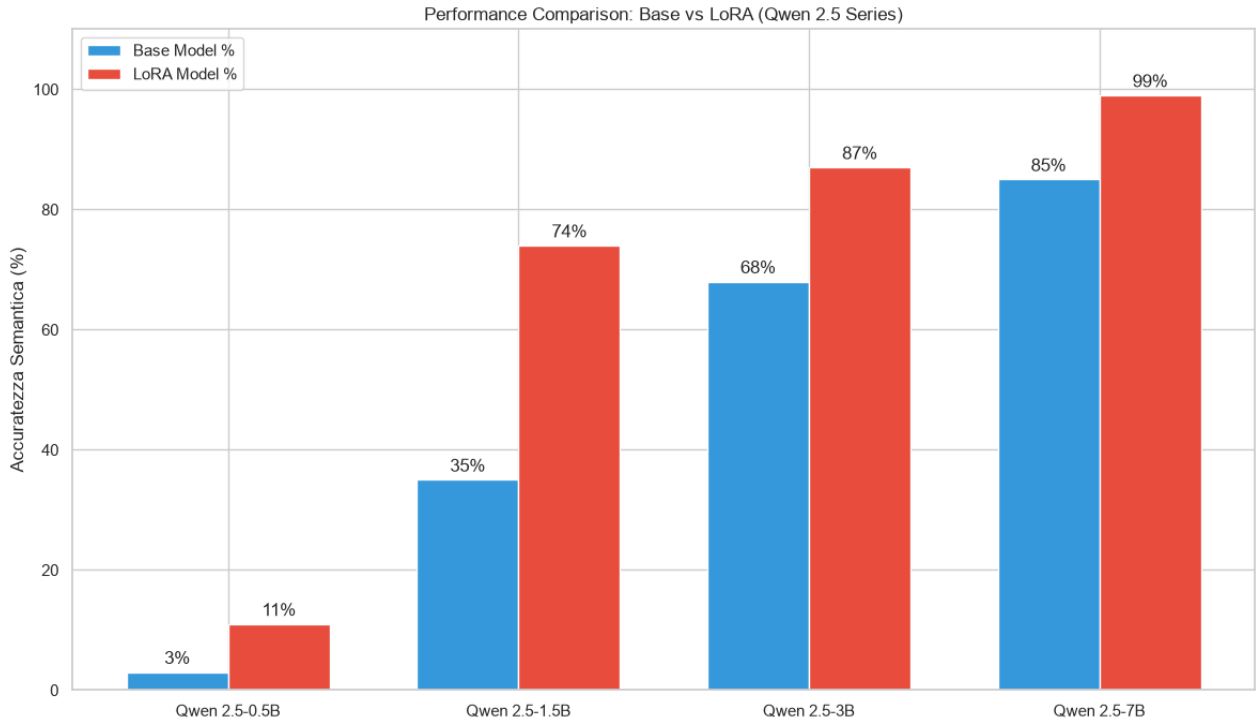
- Severità della metrica di valutazione**:** Il *Token F1* è una metrica particolarmente restrittiva che esige una corrispondenza puntuale a livello di singoli token. La minore tendenza del modello da 3B a "dilungarsi" può portarlo a generare stringhe più brevi e

morfologicamente più vicine al *"Gold Standard"* rispetto alla versione 7B, avvantaggiandolo sul piano metrico.

2. VALUTAZIONE QUALITATIVA (SEMANTICA)

In questo sottocapitolo andremo ad analizzare invece, la correttezza semantica dei modelli. Questi dati sono stati estrapolati da un'analisi eseguita secondo la tecnica de "LLM-as-a-Judge. Viene quindi utilizzato GEMINI 3.6 per analizzare le risposte generate dai modelli e attribuire un verdetto: GIUSTA o SBAGLIATA. Questo ha fornito dati fondamentali alle performance pratiche di ogni singolo modello analizzato.

2.1 Grafico a barre Base vs LoRA



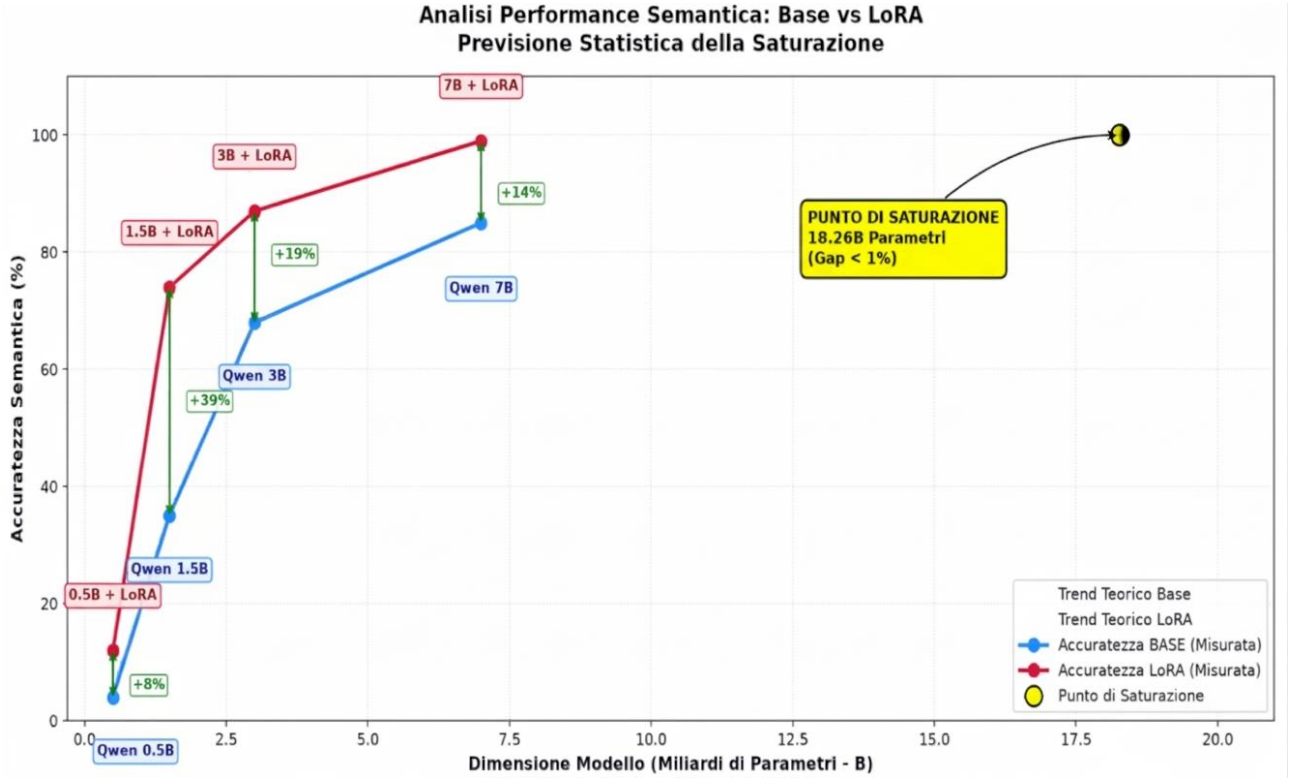
Nel grafico si osserva: in blu l'accuratezza semantica percentuale dei modelli Base, in rosso l'accuratezza semantica percentuale dei modelli LoRA. Analizzando il grafico, il dato cruciale e che salta subito all'occhio, è la normalizzazione del comportamento di QWEN 2.5 7B che, con questa tipologia di valutazione, non presenta alcuna anomalia e soprattutto ha performance superiori al modello più piccolo QWEN 2.5 3B, esattamente come ci si aspetterebbe.

- +8% per il modello QWEN 2.5 0,5B;
- +39% per il modello QWEN 2.5 1,5B;
- +19% per il modello QWEN 2.5 3B;
- +14% per i modello QWEN 2.5 7B.

Dai dati estrapolati possiamo evincere la forte propensione all'apprendimento, tramite questa metodologia di affinamento, del modello QWEN 2.5 1,5B, che ha ottenuto progressi più significativi in termini di accuratezza semantica facendo di lui lo "SWEET-SPOT" per Fine-Tuning LoRA della famiglia di modelli QWEN 2.5.

2.2 Punto di Saturazione: quando LoRA smette di essere utile

Il grafico seguente è stato creato per dare un'idea del punto di saturazione teorico. Per punto di saturazione si intende il punto in cui il modello base, relativo al numero di parametri corrispondente, raggiunge un'accuratezza semantica del 100%, equivalendosi con le prestazioni del modello che ha subito il Fine-Tuning LoRA. Per logica, nel momento in cui le prestazioni dei due modelli combaciano, l'affinamento non ha alcun effetto sul modello Base, di conseguenza l'addestramento con metodo LoRA smette di essere utile.



Nel grafico si evidenzia il punto di saturazione teorico, previsto a circa 18.26B di parametri.

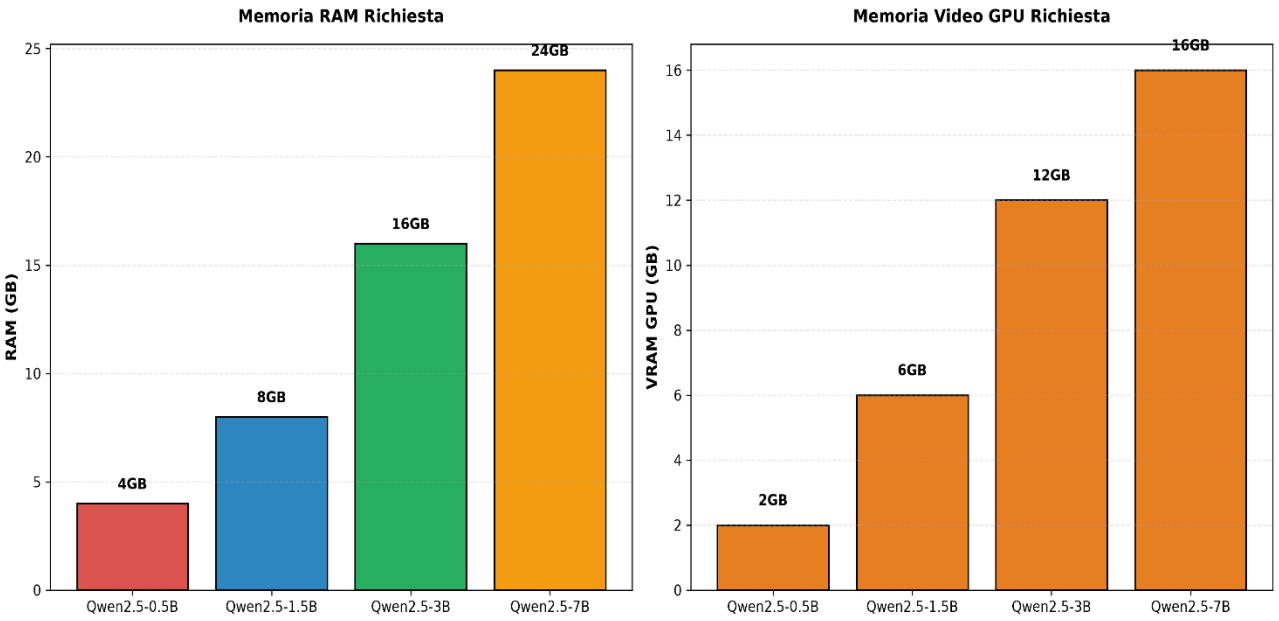
3. PRESTAZIONI E RISORSE

Introduciamo adesso tutto ciò che riguarda il lato Hardware, evidenziando quanto esso incida sulle performance e sulla buona riuscita dell'intera tesi.

3.1 Requisiti Hardware (RAM/VRAM)

Come già detto in precedenza, il suddetto progetto si è svolto completamente sulla stessa macchina, e senza l'ausilio di una GPU (scheda Video). Tutti i processi, dall'addestramento, ai benchmark, ai test pratici, si sono svolti in modalità CPU-Only, precisamente grazie ad un processore della famiglia AMD Ryzen, precisamente AMD Ryzen 5 5600x, processore a 6-Core (ovvero 6 unità fisiche di calcolo distinte) e 12-Thread (che rappresenta invece la componente software/logica del lavoro). Il tutto accompagnato da 32 GB di memoria RAM che hanno reso possibile il caricamento di modelli fino a 7 miliardi di parametri. La scelta del sistema operativo non è stata di certo casuale: la scelta è ricaduta su Windows 11 pro. Il fatto che non sia stata una scelta casuale lo dimostra il fatto che, utilizzare la piattaforma Linux, avrebbe potuto permettere di velocizzare esponenzialmente il processo, utilizzando la potenza di calcolo di una GPU, ma questo avrebbe ridotto la finestra di utenti che avrebbero potuto effettivamente mettere in atto un percorso simile, dato che Windows è certamente il sistema operativo più utilizzato in ambito professionale e no.

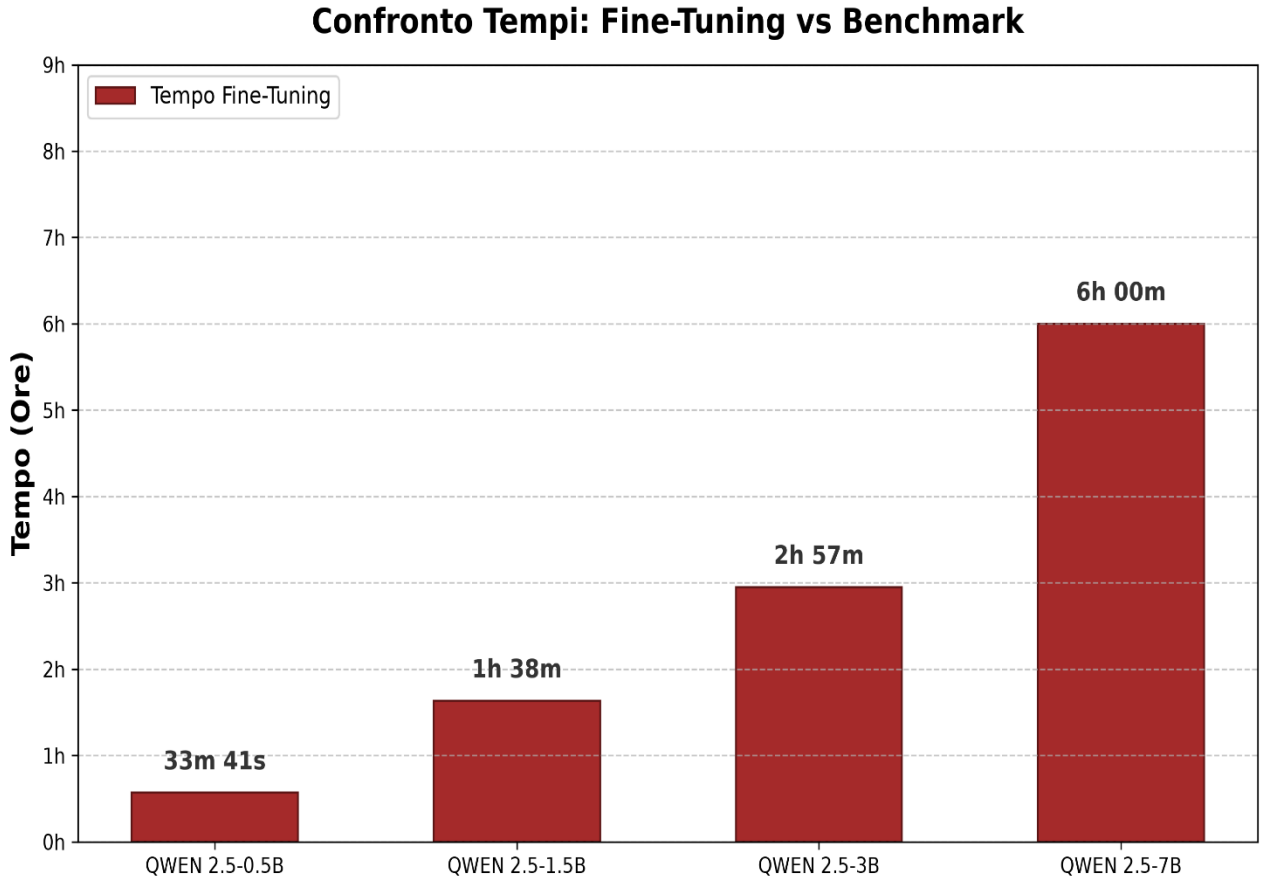
Analizziamo a livello visivo i requisiti hardware necessari a realizzare un progetto di questa tipologia:



Nell'immagine precedente troviamo una valutazione dell'hardware necessario, modello per modello. È da chiarire che, ogni modello necessita di una differente quantità di memoria e di un differente costo computazionale, questo perché le dimensioni fisiche dei modelli (GB) sono molto variabili e, di conseguenza, molto variabile sarà anche l'hardware necessario. Notiamo che per caricare i pesi del modello QWEN 2.5 0.5B in memoria, bastano 4 GB di memoria RAM e solo 2 GB di VRAM (qualora se ne volesse fare uso), mentre per il modello QWEN 2.5 7B sono richiesti addirittura 24 GB di memoria RAM e 16GB di VRAM. Questa differenza è data dal fatto che, i modelli per essere addestrati, necessitano di essere "caricati" in memoria e, ovviamente, più grandi sono i pesi, maggiore è il numero dei parametri, e maggiore sarà la quantità di memoria necessaria a "lavorare" sul modello.

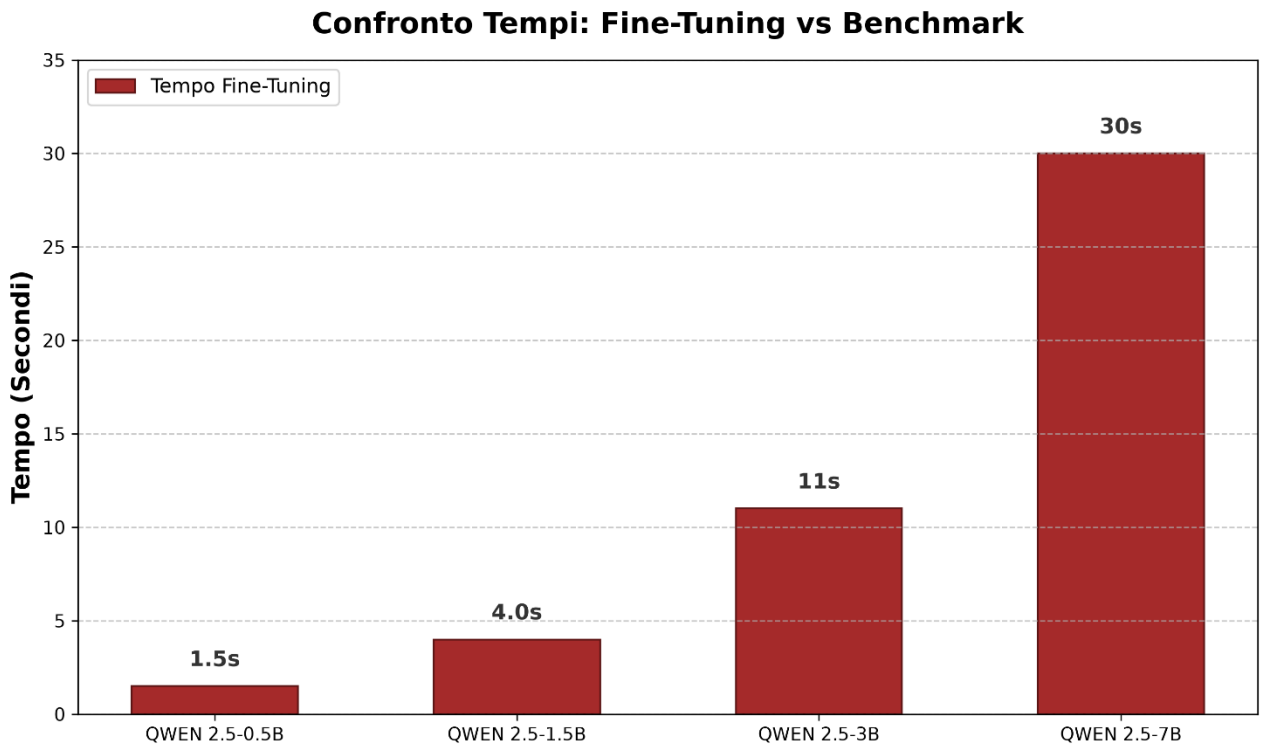
3.2 Tempi di Training e Inferenza

Per logica, le differenze hardware, sono uno specchio per le differenze nelle tempistiche di addestramento e di inferenza. Per tempi di training ci riferiamo al tempo che ogni modello impiega per leggere e memorizzare il dataset, per caricare i pesi degli adattatori LoRA e per generare le cartelle che contengono il modello stesso. Quando parliamo di "Tempi di Inferenza" invece, ci riferiamo alla fase di benchmarking dei modelli. Per inferenza si intende infatti il tempo che ogni modello impiega per generare una risposta.



Tempi di addestramento:

- QWEN 2.5 0.5B: 33 minuti e 41 secondi
- QWEN 2.5 1.5B: 1 ora e 38 minuti
- QWEN 2.5 3B: 2 ore e 57 minuti
- QWEN 2.5 7B: 6 ore



Tempi di inferenza:

- QWEN 2.5 0.5B: 1.5 secondi
- QWEN 2.5 1.5B: 4 secondi
- QWEN 2.5 3B: 11 secondi
- QWEN 2.5 7B: 30 secondi

Evidente è la crescita delle tempistiche in relazione alla crescita della grandezza dei modelli, questo è giustificato dal fatto che, durante l'addestramento, i modelli più grandi necessitano di tempi più lunghi per essere caricati in memoria e ovviamente, maggiore sarà la potenza di calcolo necessaria a far eseguire lo script, maggiore saranno i tempi per terminare i processi. Per quanto riguarda invece la fase di inferenza, ovviamente i modelli più grandi hanno prestazioni nettamente superiori ai più piccoli, ciò comporta una generazione di testi prolissi e certamente più specifici.

IV. ANALISI ECONOMICA E AMBIENTALE

1. METODOLOGIA DI CALCOLO DEI COSTI

Analizzare l'impatto economico legato all'addestramento e all'utilizzo di un modello di intelligenza artificiale applicato alla bioinformatica richiede un approccio differente rispetto ai tradizionali standard dell'informatica industriale. Spesso l'impiego dell'intelligenza artificiale viene associato a infrastrutture server particolarmente onerose o a servizi cloud terzi erogati tramite abbonamento. Al contrario, in questo studio l'intera sperimentazione è stata condotta su una stazione di lavoro desktop da studio, impiegando esclusivamente la potenza di calcolo del processore centrale e senza fare ricorso a schede grafiche dedicate.

Questa impostazione modifica radicalmente la struttura dei costi, poiché annulla la necessità di acquistare nuovo hardware dedicato o di sostenere canoni di servizio esterni. Di conseguenza, l'unica voce di spesa direttamente correlata al processo di apprendimento del modello è rappresentata dall'energia elettrica assorbita dalla rete per l'alimentazione del sistema durante le fasi di elaborazione.

Per determinare in modo rigoroso tale consumo, la spesa elettrica complessiva (C_e) viene calcolata moltiplicando la potenza media assorbita dal sistema per il tempo totale di elaborazione e per la tariffa dell'energia elettrica, secondo la seguente relazione:

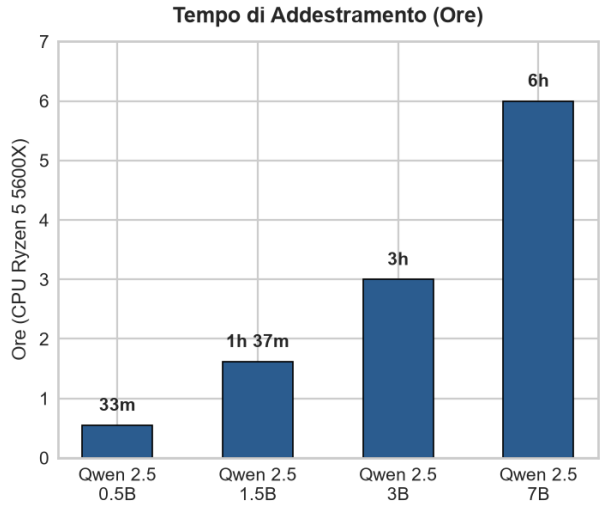
$$C_e = P \cdot t \cdot T_{kWh}$$

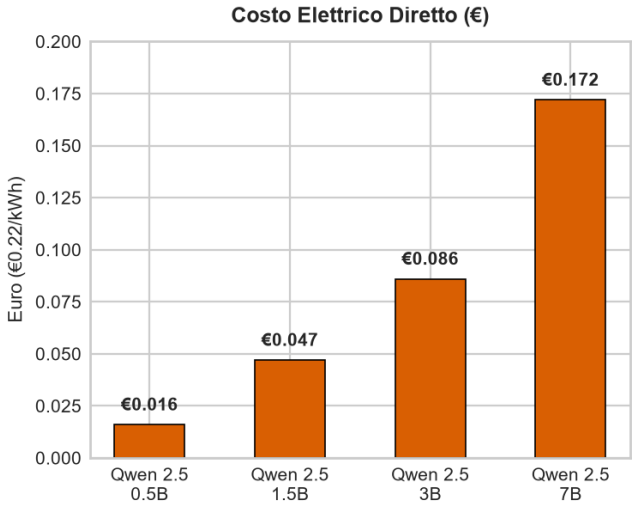
In questa formulazione, P rappresenta la potenza media assorbita dall'intero sistema espressa in chilowatt (kW), t indica la durata dell'elaborazione espressa in ore, e T_{kWh} corrisponde al costo unitario dell'energia elettrica espresso in euro per chilowattora.

Dal punto di vista hardware, la configurazione utilizzata si basa su un processore AMD Ryzen 5 5600X, caratterizzato da 6 core e 12 thread, accoppiato a 32 gigabyte di memoria RAM DDR4 e alimentato da un modulo da 750 Watt. Facendo riferimento alle specifiche tecniche ufficiali del produttore AMD e considerando il valore di Package Power Tracking (PPT) del socket, unitamente ai consumi della memoria di sistema e all'efficienza dell'alimentatore, durante le fasi di addestramento intensivo l'assorbimento medio dell'intero sistema si attesta su circa 130 Watt (0,13 kW).

Per la tariffa energetica (T_{kWh}), è stato adottato come valore di riferimento il costo medio stimato dell'energia elettrica per gli utenti domestici residenti in Italia, derivato dai dati e dai report ufficiali pubblicati dall'Autorità di Regolazione per Energia Reti e Ambiente (ARERA). Tale valore si attesta a circa 0,22 euro/kWh, una cifra comprensiva della componente materia energia, dei costi di trasporto, della gestione del contatore, degli oneri di sistema e delle relative imposte.

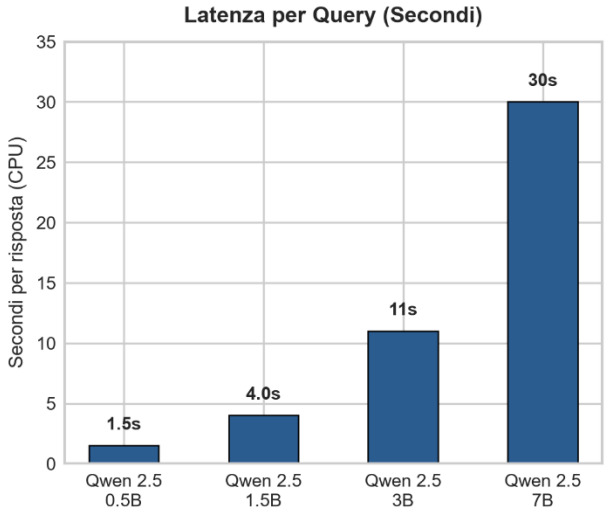
Applicando questi parametri, il calcolo della spesa economica per ciascuna prova eseguita sui modelli della famiglia Qwen 2.5 risulta del tutto trasparente. L'adozione della tecnica LoRA (Low-Rank Adaptation) — introdotta in letteratura da Edward Hu e collaboratori per consentire l'adattamento di grandi modelli linguistici riducendo drasticamente l'impiego di risorse computazionali — permette di contenere il costo dell'intero processo di fine-tuning nell'ordine di pochissimi centesimi di euro.

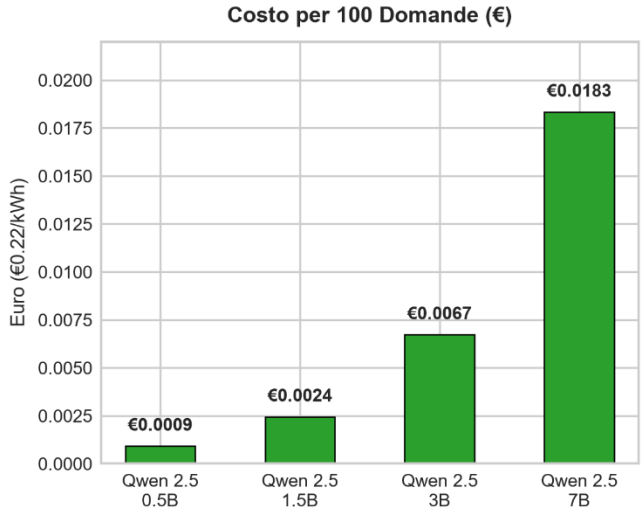




Come emerge dal grafico, la relazione tra la dimensione del modello (misurata in miliardi di parametri) e la spesa economica stimata evidenzia un trend lineare rigorosamente proporzionale al tempo di occupazione della CPU:

- **Qwen 2.5 0.5B:** 33 minuti → 0,016 euro
- **Qwen 2.5 1.5B:** 1 ora e 38 minuti → 0,047 euro
- **Qwen 2.5 3B:** 3 ore → 0,086 euro
- **Qwen 2.5 7B:** 6 ore → 0,172 euro





Sintesi dei dati d'inferenza (100 query)

- Qwen 2.5 0.5B: Latenza 1,5 s/query → 0,0009 euro per 100 domande.
- Qwen 2.5 1.5B: Latenza 4,0 s/query → 0,0024 euro per 100 domande.
- Qwen 2.5 3B: Latenza 11,0 s/query → 0,0067 euro per 100 domande.

- Qwen 2.5 7B: Latenza $30{,}0$ s/query $\rightarrow$ $0{,}0183$ euro per 100 domande.

1.1 ROI (Return-on-investment)

Per quantificare il valore economico dell'approccio proposto rispetto alle tradizionali infrastrutture *cloud* o all'acquisto di acceleratori dedicati, si applica la classica formulazione del Ritorno sull'Investimento (ROI):

$$\text{ROI} = \frac{\text{Beneficio Netto} - \text{Costo dell'Investimento}}{\text{Costo dell'Investimento}} \cdot 100$$

Nel contesto di questo studio, l'azzeramento delle spese di capitale ($\text{CapEx} = 0$, poiché non si acquista nuovo hardware) e il ridottissimo impatto dei costi operativi ($\text{OpEx} = 0{,}321$ euro per l'addestramento dell'intera suite) rendono il denominatore della formula prossimo allo zero. Questo genera un ROI teorico tendente all'infinito dal punto di vista contabile, annullando di fatto il periodo di ammortamento economico e garantendo un risparmio netto immediato rispetto al noleggio di istanze GPU commerciali o all'uso di API proprietarie a gettone.

Valutazione del Modello con il Miglior Rapporto Qualità/Prezzo

Mettendo in relazione i costi diretti di addestramento, la latenza di risposta in fase di inferenza e la capacità computazionale del sistema, emerge una netta distinzione tra i modelli valutati:

- **Qwen 2.5 0.5B e 1.5B:** Offrono i tempi e i costi più bassi (addestramento sotto i $0{,}05$ euro e latenza inferiore a 4 secondi), ma mostrano limiti nella comprensione di quesiti bioinformatici complessi.
- **Qwen 2.5 7B:** Garantisce le prestazioni qualitative più elevate, ma richiede 6 ore di addestramento e presenta una latenza di 30 secondi per risposta su CPU, risultando meno agile per applicazioni in tempo reale.
- **Qwen 2.5 3B (Scelta Ottimale):** Rappresenta il **miglior compromesso qualità/prezzo**. Con un costo di addestramento di soli $0{,}086$ euro (3 ore di occupazione della CPU) e una latenza d'inferenza contenuta a circa 11 secondi per domanda (pari a circa $0{,}0067$ euro per 100 risposte), il modello da 3 miliardi di parametri offre una solida comprensione del dominio scientifico mantenendo i tempi d'esecuzione e i consumi energetici entro limiti perfettamente gestibili su un comune processore desktop.

2. ANALISI DELL'IMPATTO AMBIENTALE

La sostenibilità dell'intelligenza artificiale moderna è diventata una questione centrale nel dibattito scientifico (Strubell et al., 2019). Quando si parla dell'impronta ecologica dei modelli linguistici di grandi dimensioni, l'attenzione cade quasi sempre sui consumi enormi dei data center aziendali, dove cluster di centinaia di GPU lavorano per settimane a pieno ritmo. In quei contesti, l'impatto ambientale è guidato da due fattori principali: l'elevata potenza elettrica assorbita dalle schede grafiche ad alte prestazioni e l'energia necessaria per alimentare gli impianti di raffreddamento industriale indispensabili per evitare il surriscaldamento dei server. Questo processo, noto come pre-addestramento, comporta la creazione del modello da zero e genera tonnellate di anidride carbonica equivalente.

Nel contesto di questa tesi, il motivo per cui l'impatto ambientale risulta drasticamente inferiore — quasi trascurabile rispetto ai parametri industriali — risiede in una scelta architetturale e metodologica ben precisa. Anziché riaddestrare l'intera rete neurale, l'esperimento adotta la tecnica dell'adattamento a basso rango, meglio nota come LoRA (Hu et al., 2021b) Questo approccio non modifica la totalità dei parametri del modello, ma *freeza* (congela) la struttura di base e aggiunge soltanto un piccolo insieme di matrici addestrabili, pari a meno dell'un per cento dei parametri complessivi. Riducendo in modo così drastico il carico computazionale, è diventato possibile spostare l'intero processo di fine-tuning da costosi server dedicati a un comune processore desktop, l'AMD Ryzen 5 5600X, lavorando esclusivamente su CPU in precisione nativa a 32 bit, senza ricorrere a tecniche di quantizzazione come il formato GGUF (Gerganov, 2023).

Per valutare la reale impronta ecologica di questo approccio, il calcolo è stato impostato partendo dall'assorbimento elettrico reale dell'hardware durante l'esecuzione. Sotto il massimo carico di lavoro, ovvero durante l'addestramento con la CPU al cento per cento delle sue capacità, il sistema registra un consumo medio di 130 Watt (0,13 kW). In fase di inferenza, cioè quando il modello elabora e risponde alle domande bioinformatiche, l'assorbimento scende a circa 100 Watt (0,10 kW).

La quantificazione dell'energia elettrica consumata (E), espressa in chilowattora (kWh), viene formulata come:

$$E_{\text{kWh}} = \frac{P_{\text{Watt}} \cdot t_{\text{ore}}}{1000}$$

Mentre l'impronta carbonica equivalente (CO_2e), espressa in grammi, applica il fattore di emissione medio della rete elettrica italiana ($F = 230$ g CO_2e/kWh) (Istituto Superiore per la Protezione e la Ricerca Ambientale (ISPRA), 2023):

$$\text{Emissioni } CO_2\text{e (g)} = E_{\text{kWh}} \cdot F_{\text{Italia}}$$

2.1 Confronto dell'Impatto Ambientale tra i modelli

Analizzando le otto configurazioni complessive considerate nello studio, emerse dal confronto tra le quattro dimensioni della suite Qwen 2.5 (0.5B, 1.5B, 3B e 7B) (Qwen et al., 2024b) sia nello stato nativo sia in quello adattato, si nota come l'impronta ecologica cresca

in modo direttamente proporzionale alla dimensione dei parametri e ai tempi di elaborazione richiesti dalla CPU.

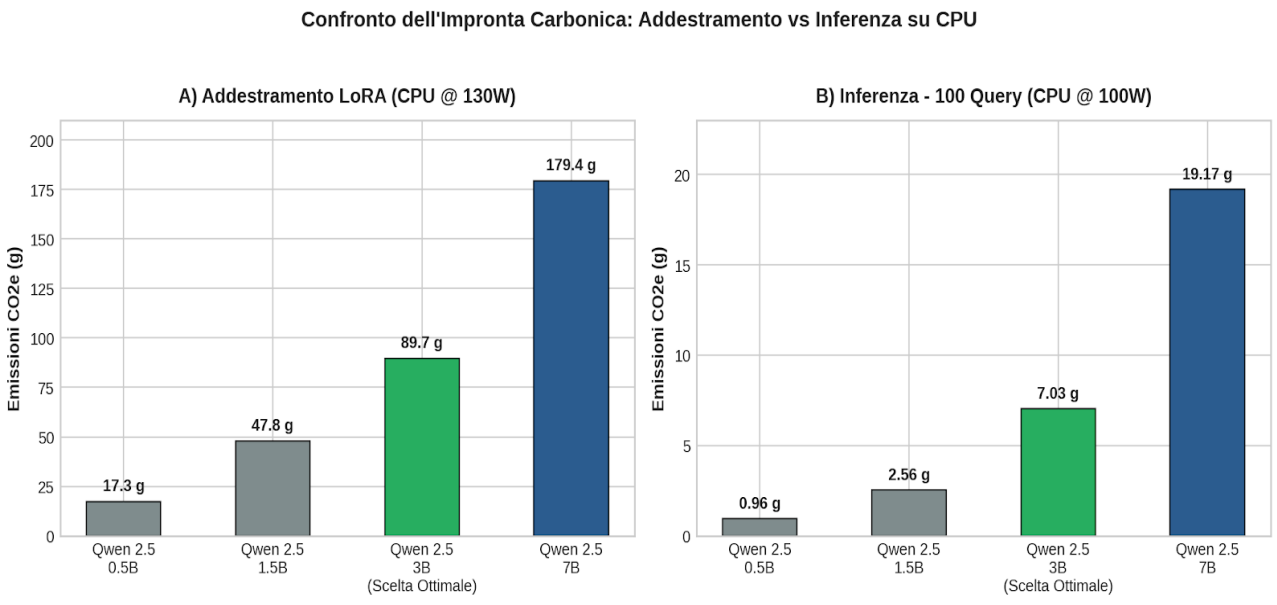
I modelli base privi di fine-tuning non comportano alcun impatto iniziale di addestramento (0 g CO_2e). Tuttavia, la loro incapacità di rispondere con precisione ai quesiti scientifici complessi porta spesso l'utente a dover formulare nuovamente le domande, generando un consumo indiretto prolungato in fase di inferenza.

Al contrario, il processo di adattamento LoRA mostra chiaramente la progressione dei consumi energetici e delle emissioni associate:

- **Qwen 2.5 0.5B:** Con circa 35 minuti di addestramento (0,58 ore, valore arrotondato per eccesso rispetto ai 33 minuti e 41 secondi misurati), la CPU assorbe 0,075 kWh, generando un'impronta carbonica di **17,3 grammi di** CO_2e.
- **Qwen 2.5 1.5B:** Richiede 1,6 ore di calcolo continuo, assorbendo 0,208 kWh con un'impronta pari a **47,8 grammi di** CO_2e.
- **Qwen 2.5 3B (Scelta Ottimale):** Con 3,0 ore di occupazione della CPU, il consumo energetico totale si attesta su 0,390 kWh, producendo **89,7 grammi di** CO_2e (un valore inferiore all'impatto di un'automobile a benzina che percorre un singolo chilometro).
- **Qwen 2.5 7B:** Con 6,0 ore di lavoro a pieno carico, l'assorbimento raggiunge 0,780 kWh ed emette **179,4 grammi di** CO_2e.

In fase di utilizzo, l'impatto dell'inferenza risulta ancora più contenuto. Elaborare un ciclo intensivo di cento domande scientifiche sul modello da 3 miliardi di parametri richiede una latenza complessiva ridotta e genera appena 7 grammi di CO_2e. Questo dimostra che l'accoppiamento tra adattamento a basso rango e addestramento su CPU costituisce un modello di intelligenza artificiale sostenibile.

2.2 Grafico impatto ambientale



I due grafici mostrano in modo distinto l'impronta carbonica prodotta durante le due fasi operative dei modelli:

1. Grafico A — Addestramento LoRA:

Mette in evidenza l'emissione totale espressa in grammi di CO_2e prodotta durante il processo di fine-tuning a pieno carico (CPU @ 130W):

- Qwen 2.5 0.5B: 17,3 g CO_2e (35 minuti)
- Qwen 2.5 1.5B: 47,8 g CO_2e (1,6 ore)
- Qwen 2.5 3B (Scelta Ottimale): 89,7 g CO_2e (3,0 ore)
- Qwen 2.5 7B: 179,4 g CO_2e (6,0 ore)

2. Grafico B — Inferenza (100 Query):

Mostra l'impatto generato dall'esecuzione di un carico di lavoro standard di 100 domande bioinformatiche in fase operativa (CPU @ 100W):

- Qwen 2.5 0.5B: 0,96 g CO_2e
- Qwen 2.5 1.5B: 2,56 g CO_2e
- Qwen 2.5 3B (Scelta Ottimale): 7,03 g CO_2e
- Qwen 2.5 7B: 19,17 g CO_2e

La separazione delle due fasi conferma come, pur crescendo in proporzione alla dimensione dei parametri, l'impatto ambientale sia per l'addestramento che per l'inferenza rimanga nell'ordine di poche decine o centinaia di grammi di CO_2e, dimostrando l'elevata sostenibilità della soluzione su CPU desktop.

V. CONCLUSIONI

Questo lavoro di tesi nasce da una domanda precisa: specializzare un modello linguistico di piccole o medie dimensioni su un dominio come la bioinformatica porta un vantaggio concreto e misurabile? E questo vantaggio giustifica lo sforzo di creare e gestire un modello locale, rispetto all'uso di soluzioni generaliste già pronte? Per rispondere sono stati confrontati quattro modelli della famiglia Qwen 2.5 (0.5B, 1.5B, 3B e 7B), ciascuno nella versione nativa e nella versione adattata con LoRA. L'addestramento ha usato un dataset di 500 coppie domanda-risposta che copre i principali ambiti della disciplina: allineamento di sequenze, genomica, NGS, trascrittomica, filogenesi, bioinformatica strutturale, database e bioinformatica clinica.

EFFICACIA DELLA SPECIALIZZAZIONE

I risultati rispondono in modo positivo alla domanda iniziale. Tutti i modelli adattati superano la propria versione di base, e lo fanno con entrambi i criteri di valutazione. Sul Token F1 i guadagni vanno da circa 0,075 a 0,144 punti. Il modello più grande arriva a 0,301, quasi il

doppio del valore di partenza. La valutazione semantica con LLM-as-a-Judge dà un'idea più fedele dell'utilità pratica, e qui il quadro è ancora più netto. L'accuratezza passa dal 3% all'11% per il modello da 0.5B, dal 35% al 74% per quello da 1.5B, dal 68% all'87% per il 3B e dall'85% al 99% per il 7B.

Il confronto tra le due metriche ha anche un valore metodologico. Con il Token F1 il modello 7B nativo ottiene un punteggio più basso del 3B. Questa anomalia scompare nella valutazione semantica, dove le prestazioni crescono regolarmente con la dimensione del modello. Ne segue che le metriche basate sulla sovrapposizione delle parole sono utili ma da sole non bastano. Tendono infatti a penalizzare le risposte corrette ma più lunghe o formulate in modo diverso dal riferimento. Per giudicare un modello su contenuti scientifici serve quindi anche una verifica del significato delle risposte, oltre alla loro forma.

RELAZIONE TRA DIMENSIONE DEL MODELLO E BENEFICIO DI LoRA

Il guadagno dovuto all'adattamento non cresce in modo lineare con il numero di parametri. Il modello da 0.5B migliora, ma resta lontano da un livello di accuratezza utilizzabile: con una capacità così ridotta il fine-tuning non riesce a compensare la scarsa conoscenza di partenza. Il miglioramento più marcato riguarda il modello da 1.5B, con un incremento di 39 punti percentuali, e per questo si può considerare lo *sweet spot* dell'adattamento LoRA in questa famiglia. Nei modelli più grandi il margine si riduce, perché la versione di base parte già da prestazioni elevate. Proiettando questo andamento con un modello di crescita asintotica, si stima un punto di saturazione teorico intorno ai 18 miliardi di parametri. Oltre questa soglia la specializzazione, almeno su un dataset di questo tipo, non darebbe più un vantaggio apprezzabile. La stima va letta come un'indicazione di tendenza più che come un valore esatto, perché deriva da soli quattro punti sperimentali. Indica comunque che LoRA è più utile proprio dove le risorse sono limitate.

SOSTENIBILITÀ ECONOMICA, AMBIENTALE E ACCESSIBILITÀ

Uno degli aspetti più rilevanti del lavoro è che l'intera sperimentazione si è svolta su un comune computer desktop, solo con la CPU, senza GPU e senza servizi cloud. Così si è dimostrato che specializzare un modello linguistico è alla portata anche di chi non dispone di infrastrutture dedicate, come uno studente, un piccolo laboratorio o un gruppo di ricerca con fondi limitati. I costi sono minimi. L'addestramento di tutti e quattro i modelli ha richiesto in totale circa 0,32 euro di energia elettrica, e rispondere a cento domande costa al massimo pochi centesimi. Anche l'impatto ambientale è contenuto: le emissioni vanno da circa 17 a 180 grammi di CO_2 equivalente per l'addestramento, e restano sotto i 20 grammi per cento risposte in fase d'uso. Sono valori lontanissimi dall'impronta del pre-addestramento nei grandi data center.

Il prezzo da pagare è il tempo. Senza accelerazione grafica, l'addestramento del modello più grande ha richiesto circa sei ore e ogni sua risposta impiega circa trenta secondi. Considerando insieme qualità, costi e tempi, il modello da 3B è risultato il miglior compromesso. Raggiunge un'accuratezza semantica dell'87%, richiede circa tre ore di addestramento e meno di nove centesimi di energia, e risponde in una decina di secondi.

LIMITI DELLO STUDIO E SVILUPPI FUTURI

I risultati vanno letti tenendo conto di alcuni limiti. Il dataset, accurato ma relativamente piccolo (400 esempi per l'addestramento e 100 per il test), copre la disciplina soprattutto dal punto di vista teorico. La valutazione semantica si basa su un solo modello giudice, che può avere i propri errori e le proprie tendenze. Infine, la scelta di lavorare solo su CPU, voluta per avvicinarsi alle condizioni di un utente medio, ha imposto configurazioni prudenti, come sequenze brevi, due sole epoche e un unico valore di rango.

Questi limiti indicano anche le direzioni per sviluppi futuri: ampliare e diversificare il dataset, anche con esercizi pratici su dati reali; confrontare diverse combinazioni di iperparametri; usare più modelli giudice o affiancare una revisione umana alla valutazione automatica; ripetere l'esperimento su hardware con GPU per misurare quanto si riducono i tempi.

CONSIDERAZIONI FINALI

In conclusione, il lavoro mostra che la specializzazione con LoRA è uno strumento efficace, economico e sostenibile per adattare modelli linguistici aperti a un dominio complesso come la bioinformatica. Un modello di piccole o medie dimensioni, addestrato in modo mirato ed eseguito interamente in locale, può offrire un supporto allo studio affidabile, con il vantaggio di garantire la riservatezza dei dati e il pieno controllo dello strumento. I modelli generalisti di grandi dimensioni restano una scelta pratica e immediata, ma i risultati ottenuti indicano che un modello specializzato e accessibile è un'alternativa concreta e valida, soprattutto nei contesti formativi e accademici. In questi contesti la precisione dei contenuti e la sostenibilità delle risorse contano quanto le prestazioni assolute.

BIBLIOGRAFIA

Aghajanyan, A., Zettlemoyer, L., & Gupta, S. (2020). *Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning* (Versione 1). arXiv. https://doi.org/10.48550/ARXIV.2012.13255

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

Berman, H. M. (2000). The Protein Data Bank. *Nucleic Acids Research*, *28*(1), 235–242. https://doi.org/10.1093/nar/28.1.235

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). *Language Models are Few-Shot Learners* (Versione 4). arXiv. https://doi.org/10.48550/ARXIV.2005.14165

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs* (Versione 1). arXiv. https://doi.org/10.48550/ARXIV.2305.14314

Felsenstein, J. (2004). *Inferring phylogenies*. Sinauer Associates.

French, R. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, *3*(4), 128–135. https://doi.org/10.1016/S1364-6613(99)01294-2

Gerganov, G. (2023). *llama.cpp: Port of Facebook's LLaMA model in C/C++*. GitHub Repository. https://github.com/ggerganov/llama.cpp

Goodwin, S., McPherson, J. D., & McCombie, W. R. (2016). Coming of age: Ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, *17*(6), 333–351. https://doi.org/10.1038/nrg.2016.49

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021a). *LoRA: Low-Rank Adaptation of Large Language Models* (Versione 2). arXiv. https://doi.org/10.48550/ARXIV.2106.09685

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021b). *LoRA: Low-Rank Adaptation of Large Language Models* (Versione 2). arXiv. https://doi.org/10.48550/ARXIV.2106.09685

Istituto Superiore per la Protezione e la Ricerca Ambientale (ISPRA). (2023). *Fattori di emissione per la produzione ed il consumo di energia elettrica in Italia*. ISPRA. https://www.isprambiente.gov.it/it/pubblicazioni/rapporti/fattori-di-emissione-per-la-produzione-ed-il-consumo-di-energia-elettrica-in-italia

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, *55*(12), 1–38. https://doi.org/10.1145/3571730

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., … Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Koonin, E. V. (2005). Orthologs, Paralogs, and Evolutionary Genomics. *Annual Review of Genetics*, *39*(1), 309–338. https://doi.org/10.1146/annurev.genet.39.073003.114725

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, *36*(4), 1234–1240. https://doi.org/10.1093/bioinformatics/btz682

Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, *15*(12), 550. https://doi.org/10.1186/s13059-014-0550-8

Morris, G. M., Huey, R., Lindstrom, W., Sanner, M. F., Belew, R. K., Goodsell, D. S., & Olson, A. J. (2009). AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry*, *30*(16), 2785–2791. https://doi.org/10.1002/jcc.21256

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., … Qiu, Z. (2024a). *Qwen2.5 Technical Report* (Versione 2). arXiv. https://doi.org/10.48550/ARXIV.2412.15115

Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., … Qiu, Z. (2024b). *Qwen2.5 Technical Report* (Versione 2). arXiv. https://doi.org/10.48550/ARXIV.2412.15115

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer* (Versione 4). arXiv. https://doi.org/10.48550/ARXIV.1910.10683

Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2019). *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models* (Versione 3). arXiv. https://doi.org/10.48550/ARXIV.1910.02054

Robinson, M. D., McCarthy, D. J., & Smyth, G. K. (2010). edgeR: A Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, *26*(1), 139–140. https://doi.org/10.1093/bioinformatics/btp616

Sayers, E. W., Bolton, E. E., Brister, J. R., Canese, K., Chan, J., Comeau, D. C., Connor, R., Funk, K., Kelly, C., Kim, S., Madej, T., Marchler-Bauer, A., Lanczycki, C., Lathrop, S., Lu, Z., Thibaud-Nissen, F., Murphy, T., Phan, L., Skripchenko, Y., … Sherry, S. T. (2022). Database resources of the national center for biotechnology information. *Nucleic Acids Research*, *50*(D1), D20–D26. https://doi.org/10.1093/nar/gkab1112

Singhal, K., Azizi, S., Tu, T., Mahdavi, S. S., Wei, J., Chung, H. W., Scales, N., Tanwani, A., Cole-Lewis, H., Pfohl, S., Payne, P., Seneviratne, M., Gamble, P., Kelly, C., Babiker, A., Schärli, N., Chowdhery, A., Mansfield, P., Demner-Fushman, D., … Natarajan, V. (2023). Large language models encode clinical knowledge. *Nature*, *620*(7972), 172–180. https://doi.org/10.1038/s41586-023-06291-2

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*(1), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3645–3650. https://doi.org/10.18653/v1/P19-1355

Tang, W. W. C., Kobayashi, T., Irie, N., Dietmann, S., & Surani, M. A. (2016). Specification and epigenetic programming of the human germ line. *Nature Reviews Genetics*, *17*(10), 585–600. https://doi.org/10.1038/nrg.2016.88

The UniProt Consortium, Bateman, A., Martin, M.-J., Orchard, S., Magrane, M., Ahmad, S., Alpi, E., Bowler-Barnett, E. H., Britto, R., Bye-A-Jee, H., Cukura, A., Denny, P., Dogan, T., Ebenezer, T., Fan, J., Garmiri, P., Da Costa Gonzales, L. J., Hatton-Ellis, E., Hussein, A., … Zhang, J. (2023). UniProt: The Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, *51*(D1), D523–D531. https://doi.org/10.1093/nar/gkac1052

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need* (Versione 7). arXiv. https://doi.org/10.48550/ARXIV.1706.03762

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., … Fan, Z. (2024). *Qwen2 Technical Report* (Versione 4). arXiv. https://doi.org/10.48550/ARXIV.2407.10671

Yang, A., Yu, B., Li, C., Liu, D., Huang, F., Huang, H., Jiang, J., Tu, J., Zhang, J., Zhou, J., Lin, J., Dang, K., Yang, K., Yu, L., Li, M., Sun, M., Zhu, Q., Men, R., He, T., … Zhang, Z. (2025). *Qwen2.5-1M Technical Report* (arXiv:2501.15383). arXiv. https://doi.org/10.48550/arXiv.2501.15383

Gerganov, G. (2023). *llama.cpp: Port of Facebook's LLaMA model in C/C++*. GitHub Repository. https://github.com/ggerganov/llama.cpp

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). *LoRA: Low-Rank Adaptation of Large Language Models* (Versione 2). arXiv. https://doi.org/10.48550/ARXIV.2106.09685

Istituto Superiore per la Protezione e la Ricerca Ambientale (ISPRA). (2023). *Fattori di emissione per la produzione ed il consumo di energia elettrica in Italia*. ISPRA. https://www.isprambiente.gov.it/it/pubblicazioni/rapporti/fattori-di-emissione-per-la-produzione-ed-il-consumo-di-energia-elettrica-in-italia

Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., … Qiu, Z. (2024). *Qwen2.5 Technical Report* (Versione 2). arXiv. https://doi.org/10.48550/ARXIV.2412.15115

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3645–3650. https://doi.org/10.18653/v1/P19-1355