ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea in Informatica

VALUTAZIONE DELLE IA GENERATIVE NELLA RISOLUZIONE DI ESERCIZI DI ALGORITMI E STRUTTURE DATI

Relatore:
Chiar.mo Prof.
Pietro Di Lena

Presentata da:
Pierpaolo Faustini

Sessione III
Anno Accademico 2024/2025

"Ergo vivida vis animi pervicit, et extra
processit longe flammantia moenia mundi"

— Tito Lucrezio Caro, *De Rerum Natura*, I, 72-73

Abstract

La pervasiva presenza dei Large Language Models (LLM) in vari contesti ha sollevato interrogativi sulle loro reali capacità. Questa tesi analizza le prestazioni di quattro modelli di intelligenza artificiale generativa (Gemini, GPT-OSS, DeepSeek, Grok) nel dominio di un dataset inedito ed italiano, composto da 34 compiti del corso di Algoritmi e Strutture Dati dell'Università di Bologna, esaminandoli in un duplice ruolo: come risolutori di compiti e come valutatori di quelli svolti dagli altri modelli (paradigma *LLM-as-a-Judge*).
I risultati sperimentali evidenziano una netta asimmetria prestazionale. Mentre in fase di esecuzione i modelli dimostrano notevoli capacità di *problem-solving* e risoluzione algoritmica, in veste di giudici emergono criticità notevoli. Le IA analizzate falliscono mediamente nell'assegnazione consistente di valutazioni intermedie (voto 1, parziale). Non riuscendo a quantificare l'intento dietro l'errore della soluzione, ripiegano spesso su logiche di giudizio binarie (voto 0 o 2). L'analisi rileva inoltre forti distorsioni sistemiche, tra cui un deciso *leniency bias* ("bias di clemenza") per i modelli minori e di criticità tecniche in alcune loro architetture.
Infine, l'esame disaggregato delle metriche di valutazione smentisce l'esistenza di un valutatore universale, dimostrando le specializzazioni di ognuno: architetture dotate di capacità avanzate di *code reasoning* si esprimono al meglio nella valutazione di esercizi di *Coding*, mentre modelli orientati al ragionamento risultano leggermente superiori nella verifica di dimostrazioni logico-matematiche, come per gli esercizi di *Analisi Asintotica*.
Alla luce di tali problematiche, si conclude che l'automazione della valutazione didattica tramite un modello sia fortemente sconsigliata, suggerendolo al più come un supporto nella correzione.

Parole chiave: *Large Language Models, LLM-as-a-Judge, Leniency Bias, Algoritmi e Strutture Dati*

Indice

Capitolo 1

Fondamenti Teorici e Stato dell'Arte

1.1 La Nascita dell'Intelligenza Artificiale

Agli albori dell'Informatica, la risoluzione automatica dei problemi era legata unicamente alla logica formale e alla deduzione matematica. Nell'estate del 1956, però, un gruppo di scienziati, riuniti al Dartmouth College negli Stati Uniti, coniò ufficialmente il termine "Intelligenza Artificiale". L'idea di quei pionieri era che qualsiasi aspetto dell'apprendimento umano potesse essere descritto in modo talmente preciso da consentire ad una macchina di simularlo [1]. Nacque la cosiddetta "IA Simbolica" (o GOFAI, *Good Old-Fashioned AI*). In questo paradigma, l'intelligenza veniva codificata attraverso l'uso di simboli e regole logiche esplicite: i programmatori creavano enormi alberi decisionali basati su infinite catene di istruzioni condizionali. Questo approccio fu eccezionale all'epoca: molti ricercatori si cimentarono nella creazione di software della famiglia GOFAI (i cosiddetti *Sistemi Esperti*) per simulare, appunto, esperti umani in un campo ristretto, come per la dimostrazione di teoremi matematici, per giocare a scacchi o per analisi finanziarie. Tuttavia, essendo pensati per ambienti dalle regole ben definite, si scontrarono presto con il muro della complessità ed ambiguità del mondo reale. Era impossibile scrivere un algoritmo classico che elencasse tutte le regole necessarie per far riconoscere a un computer un volto umano o per fargli comprendere il tono di una frase. Così, di fronte a questa limitazione strutturale, la comunità scientifica cambiò radicalmente paradigma, passando dalla deduzione logica all'induzione statistica. Invece di programmare la macchina con regole rigide, i ricercatori iniziarono a fornirle enormi quantità di dati, lasciando che fosse il sistema stesso a estrapolare i pattern e le regole sottostanti.

1.2 I Cambi di Paradigma: Il Machine Learning e il Deep Learning

Questo approccio, chiamato Machine Learning, fu una svolta poiché rappresenta un insieme di metodi statistici e computazionali che permettono ai computer di "imparare" ad eseguire compiti senza essere istruiti esplicitamente per ogni singola eventualità.
Il cuore di questo approccio risiede nell'induzione: analizzando un dataset di addestramento (*training set*), il modello cerca di minimizzare una funzione di errore, regolando i propri parametri interni finché non diventa capace di prevedere correttamente l'output per input mai visti in precedenza. Questa capacità prende il nome di **generalizzazione**.

Negli anni successivi, poi, con l'aumento esponenziale della potenza di calcolo, l'uso massiccio delle GPU e la disponibilità di ampissimi dataset nell'era di Internet, questo nuovo approccio ha trovato la sua massima espressione nel *Deep Learning* ("Apprendimento Profondo"). Basato sulla struttura del cervello umano, una serie di reti, composte da strati di "neuroni" artificiali interconnessi, sono in grado di riconoscere pattern estremamente complessi di testi, immagini, suoni. Senza questa capacità di apprendere autonomamente dalle enormi moli di materiale, lo sviluppo dei Transformer e dei modelli utilizzati in questa tesi non sarebbe mai stato possibile.

1.3 L'Evoluzione del Linguaggio Naturale ed i Transformer

Come ulteriore specializzazione, all'interno del Deep Learning, l'Elaborazione del Linguaggio Naturale (*Natural Language Processing, NLP*) è la disciplina che insegna alle macchine a comprendere e generare testo umano. Fino a pochi anni fa, lo standard industriale era dominato dalle Reti Neurali Ricorrenti (RNN). Questi sistemi "leggevano" il testo una parola alla volta, in sequenza. Il loro più grande difetto era la "memoria corta": in una frase molto lunga, tendevano a dimenticare l'argomento iniziale, perdendo il filo del discorso. Nel 2017, però, con un documento scientifico pubblicato da alcuni ricercatori di Google [2], il settore fu stravolto dall'introduzione dell'architettura *Transformer*. Il Transformer abbandona la lettura sequenziale in favore di un approccio parallelo. Il suo segreto risiede nel meccanismo di *Self-Attention* ("Auto-Attenzione"). Per spiegarlo in termini semplici, quando un Transformer legge una parola, calcola matematicamente quanto ogni altra parola della frase sia legata ad essa, assegnando dei "pesi" di importanza. Questo permette alla macchina di comprendere il contesto esatto di una parola anche se i termini collegati si trovano all'inizio e alla fine di una pagina, risolvendo dunque il problema della memoria corta.

1.3.1 Approfondimento sull'Architettura: Embedding e Positional Encoding

Perché un'architettura basata su Transformer possa elaborare il linguaggio, il primo passo fondamentale è la vettorizzazione dei token. Un token è un'unità minima di una parola, come può essere una singola lettera. Dal momento che i computer non sono in grado di elaborare direttamente i caratteri alfabetici, ogni token deve essere proiettato in uno spazio vettoriale continuo, indicato come $\mathbb{R}^{d_{model}}$. Questo processo, noto come *Word Embedding*, trasforma ogni parola in un vettore di numeri reali (spesso migliaia di dimensioni) dove la posizione del punto nello spazio non è casuale. In questo spazio ad altissima dimensionalità, le relazioni semantiche tra i termini vengono tradotte in relazioni geometriche: attraverso misure come la distanza euclidea o la *cosine similarity*, il modello "comprende" che termini come "algoritmo" e "procedura" sono concettualmente vicini, poiché i loro vettori puntano in direzioni simili. Tuttavia, sorge un problema strutturale: a differenza delle reti neurali classiche (come le RNN) che leggono le parole una dopo l'altra, il Transformer analizza l'intera frase simultaneamente per guadagnare velocità.

Senza un accorgimento specifico, il modello non avrebbe modo di distinguere la frase "il ciclo contiene l'errore" da "l'errore contiene il ciclo", poiché per lui sarebbero solo un insieme di vettori disordinati. Per risolvere questo limite, si utilizza il *Positional Encoding*. Si tratta di iniettare nei vettori di embedding dei segnali matematici specifici, basati su funzioni sinusoidali, che fungono da "etichetta posizionale". Queste funzioni permettono alla rete di distinguere l'ordine dei termini senza doverli processare in sequenza [2]:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{1.1}$$

Qui, *pos* rappresenta la posizione del token e i l'indice della dimensione. L'uso di frequenze diverse (seno e coseno) crea una sorta di "impronta digitale" unica per ogni posizione. Questo garantisce che il modello possa distinguere, ad esempio, la struttura di un ciclo `for` rispetto a una condizione `if` semplicemente analizzando la loro collocazione relativa all'interno del blocco di codice.

1.3.2 Il Meccanismo di Multi-Head Attention

Il cuore del Transformer è il meccanismo di attenzione, basato sulla trasformazione dell'input in tre matrici fondamentali: Query (Q), Key (K) e Value (V).
Per ogni token elaborato, la Query rappresenta l'informazione che il modello sta cercando, la Key rappresenta ciò che quel token ha da offrire come input al contesto, e il Value contiene il suo significato effettivo. Quando la rete analizza una frase, calcola matematicamente le affinità tra le Query e le Key di tutti i token per capire quali parole sono strettamente collegate tra loro. Questo calcolo prende il nome di *Scaled Dot-Product Attention* ed è definito dalla seguente formula [2]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.2}$$

La funzione *softmax* trasforma i risultati di questa operazione in probabilità (valori compresi tra 0 e 1), decidendo esattamente quanta "attenzione" dare a ogni parola.
Un dettaglio tecnico importante è la divisione per $\sqrt{d_k}$ (la radice quadrata della dimensione dei vettori). Questa normalizzazione serve a mantenere i numeri piccoli e gestibili: se i valori del calcolo diventassero troppo grandi, la rete si bloccherebbe, restituendo gradienti quasi nulli e smettendo di imparare durante la fase di addestramento. Infine, la *Multi-Head Attention* espande questa architettura parallelizzando il processo.
Invece di eseguire il calcolo una sola volta, il modello utilizza diverse "teste" di attenzione indipendenti. In questo modo, l'IA può osservare lo stesso testo da più prospettive contemporaneamente. Ad esempio, analizzando un frammento di codice algoritmico, una testa potrebbe concentrarsi esclusivamente sulla sintassi, come la chiusura delle parentesi, una seconda sul tracciamento delle variabili all'interno di un ciclo, e una terza sulle operazioni matematiche necessarie per l'analisi asintotica. I risultati di tutte queste teste vengono poi uniti per ottenere una comprensione completa dell'esercizio.

1.3.3 Modelli Densi e Mixture of Experts (MoE)

Dopo aver processato le informazioni e il contesto tramite la *Multi-Head Attention*, i vettori all'interno del Transformer devono compiere un ultimo passaggio: attraversare una rete neurale definita *Feed-Forward* (FFN).
Nei modelli tradizionali, questa architettura FFN viene definita **densa** (*Dense*). In un modello denso, la rete neurale è un blocco unico: questo significa che ogni singolo token in ingresso deve necessariamente attraversare e attivare tutti i parametri del modello per poter essere elaborato. Ciò crea un collo di bottiglia: per rendere un modello denso più intelligente, è necessario aumentarne le dimensioni, ma così facendo i costi di calcolo e i tempi di risposta crescono in modo insostenibile, poiché per ogni minima operazione viene mobilitato l'intero sistema.
Per superare questo limite fisico ed economico, i modelli di ultima generazione hanno adottato un'architettura rivoluzionaria chiamata **Mixture of Experts (MoE)** [3]. Invece di avere un'unica grande rete che fa tutto, lo strato *Feed-Forward* viene suddiviso in tante sotto-reti neurali più piccole ed indipendenti. Ognuna di queste sotto-reti è chiamata, appunto, "esperto" (*expert*), poiché si specializza nel tempo su compiti o concetti differenti. A coordinare questi esperti interviene un meccanismo noto come *router* (o *gating network*). Quando un nuovo token entra in questo livello, non viene più inviato a tutta la rete. Il *router* analizza il token e decide in modo dinamico a quali esperti specifici inoltrarlo. In un modello MoE moderno, per ogni input vengono attivati solo alcuni esperti alla volta.
Il vantaggio di questo paradigma è la netta separazione tra la "conoscenza" del modello e il suo "sforzo" di calcolo. Un modello MoE può ospitare centinaia di miliardi di parametri totali, possedendo così una cultura enciclopedica, ma consumare pochissima energia durante la generazione del testo, poiché la stragrande maggioranza della rete rimane inattiva, producendo soluzioni complesse in modo molto più veloce ed efficiente.

1.4 I Large Language Models (LLM)

L'architettura dei Transformer costituisce le fondamenta dei moderni "Large Language Models" (LLM). Un modello linguistico è una complessa struttura dati contenente miliardi di parametri, ovvero i pesi che la rete neurale ha ottimizzato durante la fase di addestramento. Gli LLM vengono addestrati elaborando una quantità colossale ed eterogenea di dati: durante questo processo, il modello impara a svolgere un compito base ma di vitale importanza, ovvero prevedere il token successivo all'interno di una sequenza.
Quando un utente fornisce un input (il cosiddetto *prompt*), l'LLM calcola la probabilità statistica di tutti i possibili token successivi e seleziona quello più coerente, ripetendo il processo ad altissima velocità fino al completamento della risposta [4].

1.4.1 Probabilità e Generazione Autoregressiva

La selezione del token è, come anticipato precedentemente, l'esito di un calcolo statistico. Prima di generare l'output testuale, l'ultimo strato della rete neurale produce un vettore di numeri chiamati *logits* (z_i). Questi valori rappresentano il punteggio assegnato dal modello a ciascun possibile token, ma richiedono una normalizzazione per essere inter-

pretati come probabilità. A questo scopo interviene la funzione *softmax*, che trasforma i logits in una distribuzione di probabilità vera e propria [5]:

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{1.3}$$

Dove q_i rappresenta la probabilità assegnata all'i-esimo token.
All'interno di questa formula gioca un ruolo cruciale il parametro T, noto come **Temperatura**. Questa variabile controlla la "creatività" delle risposte generate. Impostando T a un valore prossimo allo zero, la distribuzione di probabilità si estremizza, spingendo il modello a scegliere in modo quasi deterministico il token con il punteggio più alto (approccio *greedy*). Viceversa, valori di T più elevati appiattiscono la distribuzione, permettendo l'estrazione di token meno probabili e generando risposte più variegate e creative. E' scontato sottolineare che per compiti di natura algoritmica, come quelli analizzati in questa tesi, l'uso di una temperatura bassa è essenziale per garantire rigore logico.
Una volta estratto il token, il processo entra nella sua fase **autoregressiva**. Il modello genera l'output iterativamente, utilizzando l'intera sequenza prodotta fino a quel momento come input per l'iterazione successiva. Il ciclo segue questi passaggi:

1. il modello elabora il prompt iniziale e prevede il primo token;
2. il token generato viene concatenato al prompt originale;
3. l'intera stringa aggiornata diventa il nuovo input per la previsione del token seguente;
4. il ciclo si ripete fino alla generazione di un token di stop (*End-Of-Sequence*) o al raggiungimento del limite massimo di token consentito.

Questo meccanismo iterativo permette di costruire dimostrazioni matematiche e strutture dati complesse in modo incrementale. Tuttavia, esso porta con sé una vulnerabilità intrinseca: l'**accumulo dell'errore**. Se l'IA produce un passaggio logico errato durante i primi cicli, quel frammento viene re-immesso nel modello come parte della "verità" contestuale. In questa dinamica risiede l'origine delle cosiddette **allucinazioni**: si tratta di un problema statistico in cui il modello, cercando di mantenere coerenza con una premessa sbagliata, genera soluzioni all'apparenza ineccepibili ma algoritmicamente errate.

1.4.2 Il Processo di Addestramento

La capacità degli LLM di risolvere problemi complessi è il risultato di un processo di addestramento che richiede mesi di calcolo su migliaia di GPU. E' possibile suddividere questo percorso in tre fasi principali:

Fase 1: Pre-addestramento (Self-Supervised Learning)

In questa fase iniziale, il modello viene esposto a dataset testuali di dimensioni colossali (nell'ordine dei trilioni di token), che includono archivi di enciclopedie, letteratura scientifica e vaste repository pubbliche di codice. L'obiettivo è quello di addestrarlo a

prevedere il token successivo in una sequenza. Attraverso un meccanismo di apprendimento auto-supervisionato (*Self-Supervised Learning*), la rete neurale minimizza una funzione di perdita (loss function, o cosiddetta *Negative Log-Likelihood*) basata sull'errore di previsione. Formalmente, il modello ottimizza i propri parametri θ per approssimare la distribuzione di probabilità congiunta del linguaggio [6]:

$$\mathcal{L} = -\sum_{i} \log P(x_i | x_{i-k}, \ldots, x_{i-1}; \theta) \tag{1.4}$$

Al termine di questa fase, il modello acquisisce una vasta rappresentazione statistica della sintassi e della semantica dei linguaggi naturali e di programmazione, comportandosi tuttavia come un puro generatore di testo, non ancora in grado di seguire istruzioni.

Fase 2: Supervised Fine-Tuning (Instruction Tuning)

Per trasformare questo generatore statistico in un assistente interattivo, si applica il *Supervised Fine-Tuning* (SFT). Il modello viene riaddestrato su un dataset di dimensioni inferiori ma di altissima qualità, composto da coppie "Prompt-Risposta" curate da esperti umani. In questo stadio, l'IA apprende il formato dialogico e, soprattutto, interiorizza la struttura logica necessaria per risolvere compiti specifici. Ad esempio, impara che la richiesta di calcolare la complessità di un ciclo `for` richiede un'analisi formale del numero di iterazioni, piuttosto che la semplice generazione di codice sintatticamente coerente ma privo di logica.

Fase 3: RLHF e Allineamento (Reinforcement Learning from Human Feedback)

L'ultima fase, fondamentale per i modelli di ultima generazione, è l'allineamento tramite feedback umano. Poiché risulta complesso definire una funzione matematica che quantifichi rigorosamente la qualità di una spiegazione algoritmica, vi si affida a valutatori umani per classificare le diverse risposte generate dal modello.
Queste preferenze alimentano un "Modello di Ricompensa" (*Reward Model*), il quale a sua volta guida l'ottimizzazione dell'LLM principale tramite algoritmi di apprendimento per rinforzo, come il *Proximal Policy Optimization* (PPO) [7].
Questo passaggio è vitale per mitigare il fenomeno delle allucinazioni: il modello apprende che fornire una risposta matematicamente scorretta, seppur scritta in un italiano perfetto, comporta una forte penalità nel suo sistema di ricompensa.

Scaling Laws e Capacità Emergenti

Infine, un concetto essenziale per giustificare l'impiego di modelli massicci (nell'ordine di decine o centinaia di miliardi di parametri) è definito dalle *Scaling Laws* [8]. La letteratura scientifica ha dimostrato che le prestazioni di un modello migliorano in modo prevedibile, seguendo una legge di potenza basata su tre fattori: il numero di parametri, il volume dei dati di addestramento e la potenza di calcolo impiegata. Superate determinate soglie dimensionali critiche, i modelli manifestano le cosiddette "capacità emergenti": abilità di autoapprendimento non osservabili nelle reti più piccole.

Capitolo 2

Metodi e Strumenti

2.1 Il Ragionamento Algoritmico nell'IA

L'applicazione degli LLM a materie formali, come l'Algoritmica, la branca dell'Informatica preposta allo studio ed all'analisi formale degli Algoritmi, introduce sfide diverse rispetto alla semplice produzione testuale. Risolvere un esercizio di Algoritmi e Strutture Dati impone un ragionamento sequenziale e il rispetto di rigide regole matematiche. La difficoltà principale risiede in questo: mentre nel linguaggio naturale esiste una certa tolleranza all'ambiguità (una frase con un errore grammaticale può comunque essere compresa), negli algoritmi anche un singolo errore logico in un passaggio intermedio (come l'errata inizializzazione di un indice in un ciclo) invalida l'intera soluzione. Per descrivere questo limite molti ricercatori ricorrono all'analogia tra i processi cognitivi umani proposta dallo psicologo Daniel Kahneman: gli LLM operano normalmente secondo un "Sistema 1" (intuitivo, veloce e probabilistico), ideale per la scrittura e la conversazione. Al contrario, il ragionamento algoritmico richiede l'attivazione di un "Sistema 2" (analitico, sequenziale, lento e logico) [9]. Un modello deve, ad esempio, essere in grado di valutare la complessità computazionale di un frammento di codice rispettando la definizione formale di limite asintotico superiore (*Notazione O-grande*):

$$f(n) = \mathcal{O}(g(n)) \iff \exists c > 0, n_0 > 0 : 0 \leq f(n) \leq cg(n) \quad \forall n \geq n_0 \tag{2.1}$$

Questa valutazione impone all'IA di proiettare il codice nel modello di costo RAM (Random Access Machine). In questo contesto, l'IA deve generare sintassi corretta, ma deve soprattutto dimostrare di saper isolare e conteggiare le operazioni elementari, come assegnamenti di variabili e operazioni aritmetiche di base, o confronti logici in strutture di controllo. Dimostrare che un algoritmo rispetti tali vincoli richiede che il modello comprenda il numero di queste operazioni eseguite nel caso peggiore (*worst-case scenario*).

Negli ultimi anni, la comunità scientifica ha sviluppato tecniche di addestramento specifiche per colmare questo divario. In particolare, l'addestramento su vasti dataset di codice sorgente e forum tecnici (come StackOverflow) ha permesso alle IA di apprendere pattern risolutivi complessi, come l'applicazione del *Master Theorem* per la risoluzione delle equazioni di ricorrenza del tipo:

$$T(n) = aT(n/b) + f(n) \tag{2.2}$$

Inoltre, tecniche avanzate, come il *Chain-of-Thought* (CoT), inducono il modello a esplicitare i propri passaggi logici intermedi prima di giungere alla soluzione finale.
Poiché il Transformer ha una "memoria di lavoro" limitata allo spazio dei token già generati, costringerlo a scrivere ogni singolo passaggio (analisi del caso base, scomposizione ricorsiva, calcolo dei costi) permette alla rete di mantenere il filo logico.
Questo approccio "passo-passo" riduce drasticamente il tasso di "allucinazioni" matematiche e permette all'utente di verificare la correttezza di ogni singolo anello della catena deduttiva.

2.2 I Modelli Utilizzati

All'inizio dell'esperimento sono stati testati dei modelli di base ed utilizzabili gratuitamente, in particolare Qwen 3 32B e LLama 3.3-70B, modelli che ben presto hanno mostrato i loro limiti strutturali, tramite allucinazioni, soluzioni poco formali o totalmente errate. Successivamente, dunque, si è deciso di utilizzare modelli più potenti, che offrissero un bilanciamento tra forte capacità di ragionamento logico e sostenibilità economica d'uso tramite chiamate API.
Un altro fattore fondamentale per la scelta è stata la natura del dataset. Come vedremo successivamente, siccome gli esercizi non sono stati tutti uguali — alcuni di solo testo, mentre altri con anche immagini, come alberi o schemi — i modelli "text-only" non sarebbero bastati per coprire tutto il lavoro. Per questo motivo si è deciso di dividere la scelta: sono stati selezionati due modelli classici "text-to-text" per le sole tracce scritte e due modelli "Multimodali". Questi ultimi, grazie alla funzione di "Vision", sono in grado di analizzare anche la parte visiva degli esercizi, permettendo di testare l'IA anche su quella parte di dataset che richiede l'interpretazione di un'immagine per arrivare alla soluzione.
Ogni modello selezionato, poi, è stato scelto perché avesse dei valori sopra la media specialmente nel campo del coding e del ragionamento analitico. La scelta era vasta, ma molti modelli "avversari" non erano abbastanza avanzati in queste abilità che, ovviamente, sono state necessarie al fine dell'esperimento. Si è deciso di utilizzare modelli di punta delle rispettive aziende di produzione, che fossero inoltre lo **"state-of-the-art"** al 2026.
Infine, prima di scendere nel dettaglio delle singole architetture, è d'obbligo una precisazione sulla natura distributiva di questi modelli. Due modelli scelti, sono Closed-Source, non modificabili, non addestrabili e non scaricabili in locale. Altri due modeli scelti, invece, sono **"Open-Weights"**: un modello Open-Weights (a pesi aperti) permette il download completo dei parametri della rete neurale e della sua architettura, consentendone l'esecuzione, la modifica e l'addestramento su infrastrutture hardware private. Questa

distinzione serve per mostrare la flessibilità tra le due distribuzioni, fare paragoni in base ai risultati ottenuti, ed espandere il ragionamento della tesi a successive modifiche, per chi volesse addestrare questi modelli e vedere come reagiscono agli input.

Si presentano ora i "candidati" scelti per l'esperimento in esame:

2.2.1 Gemini 3.1 Pro

Il primo modello è Gemini 3.1 Pro, sviluppato da Google DeepMind, l'unità d'élite di Google dedicata alla ricerca avanzata in Intelligenza Artificiale [10]. Si tratta di un modello strettamente closed-source, non sottoponibile a fine-tuning da parte dell'utente finale. Tuttavia, compensa questa sua rigidità strutturale con la sua caratteristica cardine, che è la multimodalità nativa: a differenza di architetture che si affidano a moduli visivi esterni, Gemini elabora sia i token testuali che i pixel delle immagini nello stesso spazio, migliorando notevolmente la sua capacità di comprensione visiva.
Dal punto di vista architetturale, Google ha implementato una complessa rete **MoE ("Mixture of Experts")**, con parametri stimati tra i 10 e i 15 trilioni (tra i 10000 e i 15000 miliardi). Vi è una Context Window enorme da 1 milione di token, che lo rende un modello in grado di accettare prompt molto lunghi, supportato da un addestramento massiccio basato sul Reinforcement Learning ed un sistema migliorato di *Long-context Retrieval*, per evitare la degradazione dell'attenzione su contesti così dilatati.

2.2.2 GPT-OSS-120b

GPT-OSS-120b è un modello **text-only** dell'azienda americana OpenAI [11]. E' composto da un'architettura **MoE** da 120 miliardi di parametri, attivandone 5,1 miliardi per token, con una Context Window di 131 mila token. Totalmente Open-Weights, sottoponibile dunque a fine-tuning. Nonostante sia ospitato su diverse piattaforme, è stata scelta quella del sito Groq.com. In primis, perché permette di usarlo gratuitamente fino ad una soglia elevata di richieste e token, ma anche per il fatto che il sito sfrutti le **LPU (Language Processing Units)**, anziché le GPU tradizionali. Ciò significa che i calcoli vengono eseguiti asincronicamente a velocità altissime, essendo le LPU ottimizzate per le IA. Inoltre, su Groq viene usata una versione **quantizzata** del modello, che riduce la precisione solo nelle aree che non richiedono accuratezza totale, velocizzando ulteriormente le risposte.
Dal punto di vista dell'addestramento, utilizza le tecniche di pre e post addestramento più avanzate di OpenAI, con particolare attenzione al ragionamento, all'efficienza e all'usabilità nel mondo reale in un'ampia gamma di ambienti di distribuzione. Sorpassa molti modelli closed-source su svariati benchmark, ponendosi al di sopra di modelli stessi proprietari di OpenAI, come o4-mini.

2.2.3 DeepSeek v3.2

DeepSeek v3.2 è il modello principale della azienda cinese omonima [12]. Con architettura **MoE** con 671 miliardi di parametri totali, di cui circa 37 miliardi attivati per token 128K di Context Window, e pre-addestrato massicciamente su circa 15 trilioni di diversi e qualitativi token, rappresenta un modello eccezionale al 2026. Inoltre, anch'esso

text-only e completamente **open-weights**, rilasciato con licenza MIT. È ospitato su diverse piattaforme, insieme anche a sue varianti quantizzate. Le performance, secondo i benchmark principali, competono con modelli closed-source come GPT-5 e Claude 4 in scenari di ragionamento complesso e problem-solving matematico, superandoli in specifici benchmark come MATH e GSM8K.
Una caratteristica distintiva di DeepSeek V3.2 è la possibilità di operare in due modalità distinte, accessibili tramite due diversi endpoint API: deepseek-chat e deepseek-reasoner. La prima offre risposte veloci e dirette, ideale per query semplici. La seconda, invece, quella scelta per questa tesi, attiva il processo di ragionamento "passo-passo", consumando più token ma garantendo una profondità analitica superiore per problemi complessi come quelli algoritmici.

2.2.4 Grok 4.1

Grok 4.1, sviluppato dall'azienda americana xAI [13], introduce il concetto di **fast-reasoning**, a metà tra velocità di risposta e ragionamento computazionale profondo, ottenendo i benefici di entrambi i campi. Come Gemini, è un modello closed-source e con parametri non divulgati (ma stimabili intorno a 1.7 trilioni, basandoci su Grok 4), e con una Context Window di 2 ben milioni di token. Invece, così come DeepSeek, rispetto alla versione Fast non-reasoning, la modalità reasoning comporta un pensiero preliminare prima della risposta, impiegando token di thinking per elaborare catene logiche più complesse, riducendo le allucinazioni e migliorando l'accuratezza.
Dal punto di vista dell'addestramento, impiega tecniche avanzate di xAI non divulgate con enfasi sul ragionamento, l'efficienza e l'usabilità anche in ambienti reali, come ricerca dati e supporto clienti.
Per ultimo, non per importanza, è anch'esso un modello Multimodale, supportando input di testo, immagini e altri formati, caratterstica che lo fa scontrare direttamente con Gemini nel benchmark della Tesi negli esercizi che richiedono la "Vision".

2.3 Confronto Riassuntivo dei Modelli

Per riassumerne le differenze implementative a colpo d'occhio, viene esposta una tabella sulle caratteristiche principali dei modelli scelti, quali: Parametri di Addestramento, Context-Window, Architettura, Distribuzione e Modalità, "text-only" vs Multimodale.

Modello	**Parametri**	**Context Window**	**Architettura**	**Distribuzione**	**Modalità**
Gemini 3.1 Pro	10-15 Trilioni	1M	MoE	Closed-source	Multimodale
GPT-OSS-120b	120B (5.1B attivi)	131K	MoE	Open-Weights	Text-only
DeepSeek v3.2	671B (37B attivi)	128K	MoE	Open-Weights	Text-only
Grok 4.1	∼1.7 Trilioni	2M	MoE	Closed-source	Multimodale

Da notare, infine, per quanto riguarda l'aspetto economico, che modelli come GPT-OSS, DeepSeek e Grok presentano costi estremamente contenuti, aggirandosi tra i 20 e 30 centesimi di dollari per milione di token in input, e un massimo di 50 centesimi per milione di token in output. Grazie a queste tariffe, l'analisi dell'intero dataset e la successiva valutazione incrociata sono costate mediamente 1 dollaro per modello.
Un discorso a parte va fatto per Gemini 3.1 Pro: al momento della stesura di questa tesi, Google offriva ai nuovi utenti API un pacchetto di benvenuto del valore di 250 dollari. Questa promozione ha permesso di sfruttare la potenza di calcolo di uno dei modelli più avanzati sul mercato praticamente a costo zero, rendendo la completa sperimentazione estremamente sostenibile dal punto di vista finanziario.

Capitolo 3

Dati

3.1 Composizione del Dataset

Il dataset è stato costruito interamente su tracce d'esame reali. Nello specifico, è composto da **34 compiti ufficiali** tratti dal corso di Algoritmi e Strutture Dati dell'Università di Bologna, a partire dall'anno accademico 2021/2022, dei Professori Pietro Di Lena e Gianluigi Zavattaro. Essendo ogni compito strutturato su 4 esercizi, il bacino di test complessivo ammonta a **136 esercizi**.
In ambito IA, quando si valuta un modello, c'è un rischio enorme chiamato **Data Contamination** (contaminazione dei dati). Se si testa un'IA usando esercizi presi da siti come LeetCode.com, è probabile che l'IA li abbia già visti durante l'addestramento e li sappia a memoria. Usare esclusivamente i compiti d'esame dell'università, che non sono sparsi su Internet ma hanno un accesso vincolato al portale universitario *virtuale.unibo.it*, sebbene parte di questo materiale didattico possa essere presente su portali studenteschi locali (ma che, statisticamente, sarebbe irrilevante al fine del loro addestramento), garantisce un maggior ragionamento da parte delle IA.
Inoltre, un altro punto di forza di questo dataset è la lingua. I benchmark classici sono tutti in inglese, mentre in questo caso è in italiano: questo costringe i modelli a forzare la logica in una lingua secondaria a quella del loro addestramento, enfatizzando ulteriormente le loro reali capacità.

3.2 Formato dei Dati

Un aspetto tecnico rilevante nella preparazione dei dati riguarda il formato in cui questi sono stati sottoposti ai modelli. I compiti d'esame erano nativamente disponibili in duplice formato: codice sorgente Latex e documento PDF compilato. Questa doppia modalità è stata sfruttata per adattare l'input alle diverse architetture testate.
I file Latex sono stati forniti ai modelli *text-only* (GPT-OSS e DeepSeek). Questa scelta ha permesso di testare la loro capacità di interpretare direttamente la notazione matematica pura. I file PDF, invece, sono stati impiegati per interrogare i modelli dotati di multimodalità nativa (Gemini e Grok). Sfruttando le capacità di *Vision*, questi modelli hanno potuto elaborare visivamente l'esercizio d'esame contentente anche l'immagine, direttamente dal PDF. Tale approccio si è reso indispensabile per la risoluzione di quegli esercizi in cui la traccia non era puramente testuale, ma faceva affidamento su alberi, ta-

belle hash o liste esposti graficamente, i quali sarebbero risultati impossibili da risolvere altrimenti.

3.3 La Complessità delle Tracce Visive

Come anticipato, una frazione del dataset richiede l'interpretazione di input visivi.
È importante chiarire perché questi specifici esercizi rappresentino un test estremamente severo, anche per i modelli dotati di capacità multimodali native.
Quando un'IA analizza un'immagine comune (ad esempio, la foto di un animale), il suo compito è generalmente quello di riconoscerlo e, se presente, estrarre testo tramite *OCR (Optical Character Recognition)*. Al contrario, quando le viene sottoposta, per esempio, l'immagine di un Albero Binario di Ricerca, il modello deve ricostruire l'intera topologia della struttura: deve capire con esattezza quale nodo sia padre, figlio destro o figlio sinistro. Una singola errata interpretazione spaziale (come non visualizzare correttamente ogni singolo nodo dell'albero, e come è effettivamente capitato nel benchmark), invalida a cascata l'intero esercizio. Sottoporre ai modelli questo tipo di input significa testare il limite attuale della computer vision applicata al ragionamento logico.

3.4 Classificazione degli Esercizi

La struttura standardizzata dei compiti presi in esame ha permesso di classificare i 136 esercizi in tre macro-categorie ben distinte:

3.4.1 Categoria 1: Analisi

Questa tipologia di esercizi è incentrata sull'analisi asintotica. Una parte di questi quesiti richiede, partendo da frammenti di pseudocodice, di determinarne la complessità temporale estraendo l'equazione di ricorrenza e risolvendola tramite strumenti quali il Master Theorem o il metodo della sostituzione. A questi si affiancano esercizi di natura comparativa: viene richiesto di mettere a confronto algoritmi noti (come, ad esempio, Merge Sort e Insertion Sort), valutandone il costo computazionale nel caso ottimo e pessimo a fronte di specifici input.

3.4.2 Categoria 2: Esecuzione

Nella seconda categoria si testa la capacità di manipolazione delle strutture dati fondamentali, tra cui, ad esempio, alberi binari di ricerca (BST), alberi bilanciati (AVL), heap o tabelle hash. Bisogna simulare l'evoluzione di una struttura dati in seguito a determinate operazioni, come quelle di inserimento o di rimozione in un albero AVL e di stabilirne le corrette rotazioni per il ripristino delle proprietà strutturali. All'interno di questa tipologia di esercizi si concentra la quasi totalità dei problemi contenenti figure (es. lo stato iniziale di un albero fornito come input), rappresentando il banco di prova per i modelli multimodali.

3.4.3 Categoria 3: Coding

La terza categoria copre il restante degli esercizi, tutti quelli in cui è richiesta una progettazione algoritmica vera e propria. Si necessita di saper produrre il relativo pseudocodice per la risoluzione del problema, assieme all'analisi formale della complessità temporale dell'algoritmo.

3.5 Analisi Riassuntiva del Dataset

Anche al termine di questo capitolo, al fine di fornire una visione chiara della mole di dati elaborata durante il benchmark, viene riportata una tabella che riassume la distribuzione degli esercizi delle varie tipologie descritte:

Categoria	**Quantità**	**Note**
Totale Compiti	34	Prodotti dai Prof. P. Di Lena e G. Zavattaro
Totale Esercizi	136	4 esercizi per ogni compito
Famiglia 1: Analisi	40	Calcolo della complessità asintotica e ragionamento matematico
Famiglia 2: Esecuzione	21	Evoluzione passo-passo di strutture dati
Famiglia 3: Coding	75	Progettazione e stesura di pseudocodice con relativa complessità
Esercizi con Vision	10	Immagini necessarie per la risoluzione della traccia

Capitolo 4

Raccolta Dati

4.1 Setup dell'Esperimento

Al fine di analizzare in modo riproducibile i 136 esercizi attraverso i quattro modelli scelti, è stato necessario sviluppare un'infrastruttura software che ne permettesse l'automazione. Si è scelto di usare il linguaggio Python nella sua più recente release 3.14, per la sua velocità di produzione e semplicità d'uso. Al centro di questo esperimento vi è lo script `run_benchmark.py`. Esso si occupa di leggere i compiti d'esame e di interrogare i server di Google, OpenAI, DeepSeek e xAI per ottenere la risposta agli esercizi forniti. La risposta viene fornita in due formati: .tex, per un debug immediato delle risposte, ed in .json, più strutturato, da poter passare al valutatore successivo. Per comunicare con questi modelli, il codice fa uso dei rispettivi endpoint delle API ufficiali, tranne nel caso di GPT-OSS-120B, che, come anticipato precedentemente, si è deciso di testarlo tramite l'endpoint di Groq.com.

4.1.1 Regole del Prompt

Per prima cosa, si è azzerata la "creatività" dei modelli modificabili, impostando il parametro della temperatura al valore minimo (0.0 per DeepSeek e per GPT-OSS).
In questo modo, le macchine sono state costrette a fornire la risposta statisticamente e logicamente più probabile, evitando di inventare passaggi inesistenti.
Successivamente, è stato fornito a tutti i modelli lo stesso identico prompt iniziale:

```
SYSTEM_PROMPT = (
    "Sei un professore universitario di Algoritmi.\n"
    "Risolvi l'esercizio in modo rigoroso partendo dalla traccia
       fornita.\n"
    "Usa Latex standard e 'algorithm2e' per lo pseudocodice.\n"
    "Restituisci SOLO il corpo della soluzione (sezioni, testo,
       formule).\n"
    "NON includere \\documentclass, \\begin{document}, \\end{
       document}, \\usepackage o tag <think>.\n"
    "Non usare ```, #, **, o Markdown. Scrivi Latex puro.\n"
    "Scrivi in Italiano accademico corretto."
)
```

Questo blocco di istruzioni è stato formulato in iterazioni continue, fino al suo perfezionamento:

- **Assegnazione del ruolo:** dando l'input di comportarsi come un docente, si è permesso ai modelli di risolvere gli esercizi al massimo del rigore matematico possibile.
- **Output in Latex:** la richiesta di rispondere in formato Latex è servita al fine di sfruttare la semplicità successiva di debug. Inoltre, le IA comprendono il Latex al pari del linguaggio naturale, da cui è stata una conseguenza scontata scegliere una tale risposta in un contesto di sintassi prettamente matematica.
- **Isolamento del contenuto:** la richiesta di rimandare solamente il corpo della soluzione è servita per il punto precedente, ovvero per poter salvare la risposta testuale pulita nel file `.json` che sarebbe servita poi al valutatore.
- **Restrizioni di formattazione:** è stato vietato esplicitamente l'uso di pacchetti diversi da quelli già in uso, l'uso di Markdown o di simboli particolari. Si è notato infatti che, senza precise richieste, ogni IA rispondeva utilizzando un proprio stile in base all'addestramento ricevuto, rompendo spesso la compilazione del Latex di controllo.
- **Vincolo linguistico:** infine, è stato richiesto di scrivere in italiano corretto ed accademico, in primis per mantenere una risposta coerente con quella di un docente universitario, ed in secondo luogo poiché le IA, essendo addestrate primariamente in inglese, tendevano a procedere in tale lingua nei casi in cui era richiesto un ragionamento intensivo.

4.2 Il Problema della Multimodalità: Testo ed Immagini

Come descritto nel capitolo precedente, il dataset contiene due tipologie di esercizi: quelli puramente testuali e quelli che includono anche immagini di strutture dati. Inizialmente, nonostante si pensasse già ad una soluzione come quella effettivamente adottata, si è provato a dare in input a tutti i quattro modelli utilizzati gli stessi identici esercizi, in diversi modi, tramite ad esempio serializzazione delle strutture dati degli esercizi con immagini sfruttando la Vision di Gemini per la loro comprensione. Tuttavia, questa soluzione si è rivelata inefficace dal momento che, mancando di Vision, non solo di input, ma anche in output, ha subito condotto a problematiche di formattazione o incompatibilità testuali. Dunque, alla fine la differenza di input ha richiesto un approccio su un doppio binario. Per gli esercizi testuali, lo script legge semplicemente il file sorgente `.tex` del compito, rimuove le istruzioni generali per gli studenti (ad esempio "Tempo disponibile: 2 ore") e invia la traccia pulita ai modelli. Per gli esercizi con immagini, invece, i modelli "text-only" (DeepSeek e GPT-OSS) non sarebbero stati in grado di comprendere la traccia. Per evitare che i programmi andassero in errore, è stata creata una lista di eccezioni chiamata `VISION_OVERRIDES`. Quando lo script ha incontrato uno di questi esercizi visivi, i modelli testuali sono stati saltati automaticamente, registrando la dicitura nel .json di risposta: "N/A. Richiesta Vision".

4.2.1 Estrazione Dinamica delle Immagini

Per permettere ai modelli multimodali (Gemini e Grok) di "vedere" questi esercizi, è stato sviluppato un secondo script di supporto: `extract_from_pdf.py`.
Questo programma apre il file PDF compilato del compito e cerca di ritagliare esattamente l'area dello schermo che contiene l'esercizio e l'immagine. La logica di ritaglio è la seguente: il programma ignora la parte superiore della prima pagina per evitare di "fotografare" le regole d'esame, cerca il numero dell'esercizio (segnalato in `VISION_OVERRIDES`) e calcola l'altezza dello spazio da ritagliare fermandosi non appena incontra la parola "Soluzione". L'immagine ritagliata viene poi convertita in formato testuale *Base64* per poter essere spedita, assieme al testo dell'esercizio, tramite API ai server di Google e xAI.

4.3 Gestione degli Errori

Lavorare con le IA e le loro API ha comportato spesso problemi di connessione o di sovraccarico dei server, essendo ad oggi servizi ad alto traffico, specie sui modelli più richiesti, come Gemini 3.1 Pro. Durante i primi test, è emerso che inviare decine di richieste consecutive causava frequentemente errori di tipo *429 - "Too Many Requests"* o *503 - "Service Unavailable"*. Per evitare che il programma si bloccasse durante l'esecuzione, è stata implementata una funzione di sicurezza chiamata `call_with_retry`.
Questa funzione riprova ad inviare la richiesta fino a cinque volte: se il server risponde con un errore di sovraccarico, il programma si mette in pausa da solo (da 10 fino a 60 secondi) prima di riprovare, garantendo un'esecuzione fluida.
Inoltre, per ottenere i risultati migliori, le esecuzioni finali del benchmark sono state lan-

ciate durante le ore notturne. È stato osservato, infatti, che durante i picchi di traffico diurno alcuni provider tendono a fornire risposte più frettolose o a dirottare le richieste verso modelli leggermente meno potenti per smaltire le code, oppure a ridurre il tempo di timeout per la generazione dei ragionamenti, o addirittura a ridurre i tokens massimi di risposta. Lavorare di notte (dalle 23 alle 5 circa, ora italiana) ha garantito che i server dedicassero la massima potenza di calcolo disponibile.

4.4 Il Sistema di Valutazione: "LLM-as-a-Judge"

"LLM-as-a-Judge" è un test noto all'interno della ricerca in Intelligenza Artificiale [14]. In questo contesto, si pongono i modelli l'uno contro l'altro per ottenere delle risposte incrociate sulle soluzioni di ognuno. Un'altra modalità è quella di eleggere a "giudice" il modello migliore, più intelligente o risultato più capace su uno specifico benchmark, e far valutare a lui le soluzioni di tutti gli altri candidati. Nel nostro caso, abbiamo adottato la prima. Una volta raccolte tutte le 136 soluzioni per ciascun modello, si è deciso di automatizzarne anche la correzione, utilizzando uno script separato chiamato `evaluator.py`. In questo script, si è chiesto a ciascun modello di comportarsi da professore e di valutare i compiti degli altri modelli. Il prompt è simile a quello dello script di risposta, con la differenza delle votazioni. Naturalmente, il codice è strutturato in modo che un modello non possa mai valutare se stesso. Inoltre, i modelli senza Vision sono stati esclusi dalla valutazione dagli esercizi con le immagini, poiché i risultati, a causa della loro carenza, sarebbero stati sicuramente errati.

4.4.1 Votazioni

Al modello "Professore" è stata fornita la traccia dell'esercizio e la soluzione generata dal modello "Studente". Gli è stato poi chiesto di fornire un commento critico e un voto numerico: 0 (totalmente sbagliato), 1 (parzialmente corretto) o 2 (totalmente corretto). La votazione in 0, 1 e 2 è risultata essenziale per l'analisi delle metriche successive.
Inoltre, il voto 1 poteva ricadere all'interno di una logica di ragionamento sbagliata, ma con pseudocodice e/o complessità corrette, o viceversa.

Capitolo 5

Risultati

5.1 Valutazione Manuale

Per poter valutare le performance dei modelli scelti nel ruolo di giudici, è servito prima costruire un parametro di riferimento oggettivo: sono state dunque valutate manualmente le 136 risposte. Durante questa fase, è stata assegnato un punteggio di 0, 1 o 2, applicando le stesse regole di correzione fornite nel prompt esposto precedentemente. Questo passaggio è stato fondamentale, per poter iniziare l'analisi statistica, avendo sottomano una tabella di riferimento dei risultati con cui incrociare i risultati delle IA, in base ad alcune metriche.

5.2 Metriche di Performance

Definito il dataset di controllo tramite la valutazione personale, si è posto il problema di selezionare gli strumenti statistici più adeguati per misurare l'affidabilità dei valutatori artificiali. In un esperimento di classificazione come questo, limitarsi a contare la percentuale di risposte esatte fornisce una visione parziale e potenzialmente distorta della realtà. Come verrà dimostrato più avanti dall'analisi dei dati, il dataset risulta fisiologicamente sbilanciato: essendo i modelli molto performanti, la maggior parte delle soluzioni merita il punteggio massimo (2), rendendo le imprecisioni (1) e gli errori gravi (0) statisticamente minori. Per evitare che questo sbilanciamento avvantaggi i modelli che tendono ad assegnare per inerzia il voto più alto, l'analisi statistica è stata strutturata in tre fasi progressive:

- Inizialmente, i risultati sono stati scomposti attraverso la **Matrice di Confusione** [15], essenziale per mappare la quantità e la "direzione" degli errori;
- Successivamente, da tale matrice sono stati estratti gli indicatori per ogni singola classe di voto: **Precision**, **Recall** e **F1-Score** [15];
- Infine, sono state calcolate le metriche aggregate, tra cui la **Macro-F1** e il **Mean Absolute Error (MAE)** [15].

5.2.1 Matrice di Confusione

La Matrice di Confusione è una tabella a doppia entrata che mette a confronto i voti reali (quelli "umani", scelti manualmente) disposti sulle righe, con i voti predetti dal modello valutatore, disposti sulle colonne. Poiché le classi di voto sono tre (0, 1 e 2), la matrice risultante è una 3×3. Di seguito viene mostrato il formato di tale matrice:

		Voto Predetto (IA)		
		0	**1**	**2**
Voto Reale	**0**	*Corretto* (TP)	Errore	Errore
	1	Errore	*Corretto* (TP)	Errore
	2	Errore	Errore	*Corretto* (TP)

Come si evince dallo schema, la diagonale principale contiene tutte le valutazioni esatte, mentre le celle esterne rappresentano le discordanze. Da questa matrice, isolando una specifica classe (ad esempio, analizzando esclusivamente la colonna e la riga del voto "2"), si estraggono tre concetti base per il calcolo delle metriche successive:

- **Veri Positivi (TP):** Il modello assegna il voto "2" ed era stato effettivamente assegnato "2" (l'incrocio esatto sulla diagonale);
- **Falsi Positivi (FP):** Il modello assegna il voto "2", ma era stato dato "0" o "1" (calcolati sommando il resto della colonna dell'IA);
- **Falsi Negativi (FN):** Era stato dato "2", ma il modello ha assegnato un voto diverso (calcolati sommando il resto della riga valutata personalmente).

5.2.2 Precision, Recall e F1-Score

La **Precision** misura l'affidabiltà del giudizio dell'IA. Essa guarda quante volte l'IA ha messo un voto di una classe e si chiede: "se ha dato un voto per una classe, quante volte ci ha preso?". In formula:

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c} \tag{5.1}$$

La **Recall** misura invece la sensibilità dell'IA rispetto al valutatore umano. Essa guarda quante volte è stato messo un voto di una classe, e si chiede: "se l'umano ha dato 0, quante volte l'IA se n'è accorta?". In formula:

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c} \tag{5.2}$$

L'**F1-Score** rappresenta la media armonica tra le due.
A differenza della media classica, la media armonica penalizza molto i valori bassi: se, ad esempio, un modello ha una Recall perfetta ma una Precisione disastrosa, l'F1-Score crollerà, smascherando il suo squilibrio:

$$\mathrm{F1}_c = 2 \times \frac{\mathrm{Precision}_c \times \mathrm{Recall}_c}{\mathrm{Precision}_c + \mathrm{Recall}_c} \tag{5.3}$$

5.2.3 Macro-F1, Weighted-F1, Accuratezza e MAE

La **Macro-F1** calcola la media aritmetica degli F1-Score di ogni singola classe, trattandole tutte con lo stesso peso. Questo significa che, affinché la Macro-F1 sia alta, il modello deve essere bravo non solo a riconoscere i "2", ma anche gli "0" e gli "1", nonostante siano numericamente inferiori:

$$\text{Macro-F1} = \frac{1}{N} \sum_{c=0}^{N-1} \mathrm{F1}_c \quad \text{con } N = 3 \tag{5.4}$$

La **Weighted-F1** calcola invece la media degli F1-Score pesata sul numero effettivo di esercizi appartenenti a ciascuna classe. Essendo C_c il numero totale di esercizi valutati personalmente con il voto c (0, 1 o 2), e T il numero totale delle valutazioni (ovvero $T = C_0 + C_1 + C_2$), la formula è:

$$\text{Weighted-F1} = \frac{1}{T} \sum_{c=0}^{N-1} (C_c \cdot \mathrm{F1}_c) \quad \text{con } N = 3 \tag{5.5}$$

L'**Accuratezza Globale** (Accuracy) è la metrica più intuitiva: rappresenta la semplice percentuale di risposte esatte rispetto al totale delle valutazioni (T). Tuttavia, in un contesto sbilanciato, può risultare in un'illusione: un modello che assegna sempre "2" otterrebbe un'Accuratezza altissima solo per inerzia, pur essendo un pessimo valutatore:

$$\text{Accuracy} = \frac{1}{T} \sum_{c=0}^{N-1} TP_c \quad \text{con } N = 3 \tag{5.6}$$

Infine, il **Mean Absolute Error** (MAE) ci dice *quanto* il modello abbia sbagliato in media. Non tutti gli errori hanno lo stesso peso: se un compito da 2 riceve un 1, l'errore è di un solo punto; ma se un compito da 2 riceve uno 0, l'errore è doppio e concettualmente molto più grave. Il MAE cattura esattamente questa "distanza" tra il voto predetto ($\hat{y}_i$) e quello reale (y_i), in valore assoluto:

$$\text{MAE} = \frac{1}{T} \sum_{i=1}^{T} |y_i - \hat{y}_i| \tag{5.7}$$

5.3 Metodologia di Estrazione Dati

Il calcolo delle metriche appena descritte è stato anch'esso automatizzato attraverso una serie di script in Python. L'elaborazione è stata divisa in due fasi principali: nella

prima, è stata analizzata l'intera directory contenente i log delle valutazioni in formato .json isolando i giudizi espressi da ciascun modello, per poi strutturare i dati in quattro file .csv distinti in output (uno per ogni Intelligenza Artificiale). La seconda fase ha riguardato il calcolo statistico vero e proprio. Tramite la libreria *Pandas*, le valutazioni artificiali sono state incrociate con il dataset di verità (i voti personali). Durante la fase preliminare, sono stati allineati i compiti e scartate automaticamente le coppie di dati non omogenee in cui era presente un valore nullo ("N/A") da parte della valutazione personale o del modello. Successivamente, la parte dell'analisi è stato affidata a **scikit-learn**, la libreria di Machine Learning più conosciuta ed utilizzata in ambito statistico, nata in contesto accademico, con algoritmi ottimizzati e *peer-reviewed* per il calcolo delle metriche suddette.

5.4 Performance dei modelli come Esecutori

Prima di esaminare le capacità dei modelli nella valutazione, si mostrano i loro rendimenti nella risoluzione degli esercizi proposti. Il punteggio medio ottenuto (calcolato su una scala da 0 a 2) è stato diviso per le tre categorie di esercizi definite precedentemente: Analisi, Esecuzione e Coding. Ciò mostra le specifiche aree di bravura o le lacune di ciascuno in veste di "studente". Infine, è stata aggiunta una colonna "Globale", che rappresenta la media aritmetica dei punteggi ottenuti, senza suddivisione in categorie.

Modello	**Analisi**	**Esecuzione**	**Coding**	**Globale**
GEMINI	1.95	1.90	1.95	1.94
DEEPSEEK	1.65	1.85	1.77	1.74
GROK	1.80	1.29	1.80	1.72
GPT-OSS	1.45	1.75	1.68	1.62

Tabella 5.1: Punteggio medio (0-2) ottenuto dai modelli come risolutori, suddiviso per categorie di esercizi

5.5 Performance dei Modelli come Valutatori

Dopo aver stabilito le capacità di base dei modelli nella risoluzione, l'attenzione si sposta sul loro rendimento in veste di "professori". Per valutare l'affidabilità di ciascun modello nella correzione automatica, le predizioni sono state confrontate con le valutazioni umane di riferimento. I risultati sono stati suddivisi in due macro-famiglie. Nella prima, **Performance per Categoria**, al fine di fornire un'analisi dettagliata, le performance sono state disaggregate secondo due fattori: la capacità di valutazione in base alla *categoria dell'esercizio* e la precisione nell'individuare la specifica *classe di voto* (0, 1 o 2). Nella seconda macro-area, **Performance Globali**, sono state divise solo per classe di voto, senza distinzione sulla categoria di esercizio.

5.5.1 Performance per Categoria

Modello: GEMINI

Categoria: Analisi

GEMINI	IA: 0	IA: 1	IA: 2
Reale: 0	1	9	11
Reale: 1	0	1	1
Reale: 2	0	12	85

Tabella 5.2: Matrice - Analisi

Classe	Precision	Recall	F1-Score
0	1.00	0.05	0.09
1	0.05	0.50	0.08
2	0.88	0.88	0.88
Macro-F1			**0.35**
Weighted-F1			**0.73**
MAE			**0.37**
Accuratezza			**0.72**

Tabella 5.3: Metriche - Analisi

Categoria: Esecuzione

GEMINI	IA: 0	IA: 1	IA: 2
Reale: 0	4	2	3
Reale: 1	0	2	0
Reale: 2	0	5	30

Tabella 5.4: Matrice - Esecuzione

Classe	Precision	Recall	F1-Score
0	1.00	0.44	0.62
1	0.22	1.00	0.36
2	0.91	0.86	0.88
Macro-F1			**0.62**
Weighted-F1			**0.81**
MAE			**0.28**
Accuratezza			**0.78**

Tabella 5.5: Metriche - Esecuzione

Categoria: Coding

GEMINI	IA: 0	IA: 1	IA: 2
Reale: 0	4	1	0
Reale: 1	0	21	24
Reale: 2	0	22	149

Tabella 5.6: Matrice - Coding

Classe	Precision	Recall	F1-Score
0	1.00	0.80	0.89
1	0.48	0.47	0.47
2	0.86	0.87	0.87
Macro-F1			**0.74**
Weighted-F1			**0.79**
MAE			**0.21**
Accuratezza			**0.79**

Tabella 5.7: Metriche - Coding

Modello: GPT-OSS

Categoria: Analisi

GPT-OSS	IA: 0	IA: 1	IA: 2
Reale: 0	3	2	7
Reale: 1	0	0	0
Reale: 2	7	3	98

Tabella 5.8: Matrice - Analisi

Classe	Precision	Recall	F1-Score
0	0.30	0.25	0.27
1	0.00	0.00	0.00
2	0.93	0.91	0.92
Macro-F1			**0.40**
Weighted-F1			**0.86**
MAE			**0.28**
Accuratezza			**0.84**

Tabella 5.9: Metriche - Analisi

Categoria: Esecuzione

GPT-OSS	IA: 0	IA: 1	IA: 2
Reale: 0	0	1	0
Reale: 1	0	0	0
Reale: 2	0	1	37

Tabella 5.10: Matrice - Esecuzione

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.00	0.00	0.00
2	1.00	0.97	0.99
Macro-F1			**0.33**
Weighted-F1			**0.96**
MAE			**0.05**
Accuratezza			**0.95**

Tabella 5.11: Metriche - Esecuzione

Categoria: Coding

GPT-OSS	IA: 0	IA: 1	IA: 2
Reale: 0	0	1	0
Reale: 1	1	1	28
Reale: 2	1	10	177

Tabella 5.12: Matrice - Coding

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.08	0.03	0.05
2	0.86	0.94	0.90
Macro-F1			**0.32**
Weighted-F1			**0.78**
MAE			**0.19**
Accuratezza			**0.81**

Tabella 5.13: Metriche - Coding

Modello: DEEPSEEK

Categoria: Analisi

DEEPSEEK	IA: 0	IA: 1	IA: 2
Reale: 0	2	2	11
Reale: 1	0	2	0
Reale: 2	4	11	88

Tabella 5.14: Matrice - Analisi

Classe	Precision	Recall	F1-Score
0	0.33	0.13	0.19
1	0.13	1.00	0.24
2	0.89	0.85	0.87
Macro-F1			**0.43**
Weighted-F1			**0.78**
MAE			**0.36**
Accuratezza			**0.77**

Tabella 5.15: Metriche - Analisi

Categoria: Esecuzione

DEEPSEEK	IA: 0	IA: 1	IA: 2
Reale: 0	0	1	0
Reale: 1	0	1	0
Reale: 2	0	2	34

Tabella 5.16: Matrice - Esecuzione

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.25	1.00	0.40
2	1.00	0.94	0.97
Macro-F1			**0.46**
Weighted-F1			**0.93**
MAE			**0.08**
Accuratezza			**0.92**

Tabella 5.17: Metriche - Esecuzione

Categoria: Coding

DEEPSEEK	IA: 0	IA: 1	IA: 2
Reale: 0	0	1	1
Reale: 1	1	12	21
Reale: 2	1	19	163

Tabella 5.18: Matrice - Coding

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.38	0.35	0.36
2	0.88	0.89	0.89
Macro-F1			**0.42**
Weighted-F1			**0.80**
MAE			**0.21**
Accuratezza			**0.80**

Tabella 5.19: Metriche - Coding

Modello: GROK

Categoria: Analisi

GROK	IA: 0	IA: 1	IA: 2
Reale: 0	0	3	15
Reale: 1	0	1	1
Reale: 2	5	5	90

Tabella 5.20: Matrice - Analisi

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.11	0.50	0.18
2	0.85	0.90	0.87
Macro-F1			**0.35**
Weighted-F1			**0.73**
MAE			**0.41**
Accuratezza			**0.76**

Tabella 5.21: Metriche - Analisi

Categoria: Esecuzione

GROK	IA: 0	IA: 1	IA: 2
Reale: 0	0	3	0
Reale: 1	0	1	0
Reale: 2	4	2	36

Tabella 5.22: Matrice - Esecuzione

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.17	1.00	0.29
2	1.00	0.86	0.92
Macro-F1			**0.40**
Weighted-F1			**0.85**
MAE			**0.28**
Accuratezza			**0.80**

Tabella 5.23: Metriche - Esecuzione

Categoria: Coding

GROK	IA: 0	IA: 1	IA: 2
Reale: 0	0	3	0
Reale: 1	0	12	26
Reale: 2	3	15	162

Tabella 5.24: Matrice - Coding

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.40	0.32	0.35
2	0.86	0.90	0.88
Macro-F1			**0.41**
Weighted-F1			**0.78**
MAE			**0.23**
Accuratezza			**0.79**

Tabella 5.25: Metriche - Coding

5.5.2 Performance Globali

Modello: GEMINI

GEMINI	IA: 0	IA: 1	IA: 2
Reale: 0	9	12	14
Reale: 1	0	24	25
Reale: 2	0	39	264

Tabella 5.26: Matrice Globale

Classe	Precision	Recall	F1-Score
0	1.00	0.26	0.41
1	0.32	0.49	0.39
2	0.87	0.87	0.87
Macro-F1			**0.56**
Weighted-F1			**0.77**
MAE			**0.27**
Accuratezza			**0.77**

Tabella 5.27: Metriche Globali

Modello: GPT-OSS

GPT-OSS	IA: 0	IA: 1	IA: 2
Reale: 0	3	4	7
Reale: 1	1	1	28
Reale: 2	8	14	312

Tabella 5.28: Matrice Globale

Classe	Precision	Recall	F1-Score
0	0.25	0.21	0.23
1	0.05	0.03	0.04
2	0.90	0.93	0.92
Macro-F1			**0.40**
Weighted-F1			**0.82**
MAE			**0.20**
Accuratezza			**0.84**

Tabella 5.29: Metriche Globali

Modello: DEEPSEEK

DEEPSEEK	IA: 0	IA: 1	IA: 2
Reale: 0	2	4	12
Reale: 1	1	15	21
Reale: 2	5	32	285

Tabella 5.30: Matrice Globale

Classe	Precision	Recall	F1-Score
0	0.25	0.11	0.15
1	0.29	0.41	0.34
2	0.90	0.89	0.89
Macro-F1			**0.46**
Weighted-F1			**0.80**
MAE			**0.24**
Accuratezza			**0.80**

Tabella 5.31: Metriche Globali

Modello: GROK

GROK	IA: 0	IA: 1	IA: 2
Reale: 0	0	9	15
Reale: 1	0	14	27
Reale: 2	12	22	288

Tabella 5.32: Matrice Globale

Classe	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.31	0.34	0.33
2	0.87	0.89	0.88
Macro-F1			**0.40**
Weighted-F1			**0.77**
MAE			**0.29**
Accuratezza			**0.78**

Tabella 5.33: Metriche Globali

5.6 Analisi dei Risultati

Anche la discussione dei risultati è stata divisa in due sezioni fondamentali, riflettendo la doppia natura dell'esperimento: la prima, **Esecuzione**, indaga le capacità dei modelli in veste di risolutori degli esercizi proposti, mentre la seconda, **Valutazione**, ne esamina l'affidabilità come "valutatori".

5.6.1 Esecuzione

Analizzando innanzitutto i modelli in veste di "studenti", emerge un ottimo livello di comprensione delle tracce e della loro capacità di risolvere i problemi dati. Gemini si posiziona come il risolutore migliore, ottenendo un punteggio medio di ben 1.94 su 2. DeepSeek e Grok, con un punteggio rispettivamente di 1.74 ed 1.72, mostrano prestazioni comunque buone ma con alcune incertezze strutturali. Il modello GPT-OSS, infine, chiude la classifica con un punteggio medio di 1.62.

Complessità relativa delle tipologie di esercizio

Osservando i risultati, si nota chiaramente come la categoria dell'esercizio influenzi la capacità di successo dell'IA. Gli esercizi di *Coding* si sono rivelati, in media, i più semplici da risolvere. Questo dato, seppur apparentemente controintuitivo per uno studente reale, trova la sua giustificazione nella natura dell'addestramento dei modelli: i problemi algoritmici classici sono sempre presenti nei dataset di training. Al contrario, gli esercizi di *Analisi* si sono confermati i più ostici, in quanto richiedono un ragionamento algebrico più profondo, competenze su cui i modelli mostrano ancora difficoltà.

Trappole matematiche

Negli esercizi di Analisi (per la maggior parte incentrati sul calcolo della complessità asintotica), si è registrata una vulnerabilità ai "trabocchetti" matematici. In particolare, di fronte a costanti ingannevoli come n^3 inserito in un contesto non dominante, o a notazioni come $2 \times \text{FUNC}(n)$, la quasi totalità dei modelli ha fallito l'interpretazione. Molti hanno calcolato erroneamente chiamate ricorsive multiple al posto di semplici costi costanti. L'unica eccezione degna di nota è rappresentata da Gemini, che ha dimostrato

una capacità superiore, evitando gli errori che gli altri modelli hanno ripetutamente compiuto, tranne che in un caso sporadico.

Carenze Strutturali

Specialmente negli esercizi di Esecuzione, si sono presentate interessanti problematiche da analizzare. Grok, modello che è stato scelto appositamente per scontrarsi con Gemini negli input multimodali, ha mostrato un forte difetto nella sua capacità di Vision. In molteplici casi, invece di analizzare sia il testo che la figura forniti in input nell'immagine inviata ai loro server, ha prodotto risposte basate esclusivamente sulla figura o sui valori numerici forniti come esempio nella traccia, fallendo così l'implementazione della soluzione corretta. Ciò, oltre a riflettersi nel voto più basso in assoluto nella sua categoria (1.29), fa comprendere anche le differenze di implementazione dei moduli OCR: molto più avanzati in Google, probabilmente grazie ad anni di sviluppo della loro tecnologia (si pensi, ad esempio, alla decifrazione di immagini durante lo sviluppo di Google Maps) e più indietro per xAI, con la produzione di un modello potente algoritmicamente ma ancora agli albori per il riconoscimento visivo avanzato.

Dimostrazioni Forzate

GPT-OSS ha esibito un particolare comportamento "scorretto" nelle dimostrazioni matematiche: pur di "far tornare i conti" per raggiungere la soluzione richiesta, ha inventato ricorrenze prive di senso o ha generato dimostrazioni che si contraddicevano palesemente nel giro di poche righe (ad esempio, deducendo l'esistenza di alberi di copertura con un numero logicamente impossibile di archi). Ciò è avvenuto nonostante l'aver forzato la *Temperatura* a 0.0, il minimo, il che mostra un errore ancora più grave.

Trascuratezza Sintattica

Infine, DeepSeek ha evidenziato una modalità di azione duplice. Se da un lato il modello si è dimostrato ottimo nel ragionamento, dall'altro ha generato spesso codice mal formattato o affetto da errori sintattici. Questa trascuratezza contrasta con la logica delle sue soluzioni, suggerendo che l'elevata capacità di ragionamento non si traduce sempre in una stesura del codice pulita.

5.6.2 Valutazione

Il panorama cambia notevolmente quando i modelli sono chiamati ad essere valutatori, un compito che richiede abilità diverse rispetto alla mera esecuzione. Dalle matrici di confusione si evince una tendenza a sovrastimare la bontà delle soluzioni rispetto alla valutazione reale. Tutte le IA faticano molto a inquadrare la classe di voto "1" (come soluzione parzialmente corretta), tendendo a polarizzare il giudizio verso gli estremi, o assegnando tale valutazione erroneamente, nonostate siano state istruite nel prompt a dare quella votazione nei casi intermedi.

Modello più affidabile

Gemini conferma la sua superiorità anche in fase di valutazione, registrando la Macro-F1 Globale più alta (0.56), indice di una capacità valutativa generale maggiore in un dataset simile, specialmente nel dominio del Coding (0.74).

Andamenti altalenanti

DeepSeek mostra, come già accennato, un comportamento speculare rispetto a Gemini: registra un picco più alto nella categoria Analisi (Macro-F1 0.43, il migliore in questa specifica categoria), ma si perde, nell'analisi specifica del suo modello, nella valutazione del Coding (0.42), scambiando frequentemente soluzioni ottime per soluzioni mediocri. Grok invece mantiene una costanza piatta e poco brillante su tutte le categorie (valori di Macro-F1 compresi tra 0.35 e 0.41).

Bassa affidabilità dell'Accuratezza e della Weighted-F1

Come spiegato invece nella sezione delle metriche, l'Accuratezza è una metrica fuorviante senza un'analisi approfondita dei modelli e del dataset di riferimento. Per dimostarlo si può prendere come esempio GPT-OSS: esso possiede l'Accuratezza Globale più alta di tutti (0.84) e l'errore medio (MAE) più basso (0.20). Da queste metriche potrebbe sembrare il migliore, ma tali valori sono prodotti dal forte sbilanciamento del dataset verso la classe maggiore (il voto 2). Inoltre, essendo il modello meno performante, si è notato che possa soffrire del cosiddetto **Leniency Bias** [16], ovvero eccedere con la clemenza nel dare i voti, perché gli altri modelli ai suoi occhi sembrano eccellenti. Stesso fenomeno affligge la metrica Weighted-F1: calcolando la media ponderata in base alle istanze per ogni classe, questa metrica viene quasi totalmente assorbita dalle ottime performance sul voto 2, sovrapponendosi, spesso, all'Accuratezza.

Capitolo 6

Conclusioni

6.1 L'Asimmetria tra Esecuzione e Valutazione

Al termine di questa tesi si è delineato un quadro piuttosto chiaro e divisivo sulle attuali capacità di questi modelli. Il risultato più evidente è l' asimmetria prestazionale tra la fase di esecuzione e quella di valutazione. Da un lato i modelli dimostrano di essere ottimi "studenti", capaci di risolvere complessi problemi algoritmici di difficoltà universitaria, dall'altro invece si rivelano valutatori ancora poco performanti.

6.1.1 Profili e Specializzazioni dei Modelli

L'esperimento ha inoltre sfatato il mito dell'IA "universale", evidenziando specifiche caratteristiche per ciascun modello:

- **Gemini** si conferma il modello più completo ed equilibrato. Domina la fase di esecuzione e si attesta come il miglior valutatore assoluto per il Coding, pur mostrando limiti nella correzione di dimostrazioni logico-matematiche altrui.
- **DeepSeek** fa emergere la sua caratteristica di *reasoning*. È l'unico modello ad avere i risultati migliori, seppur mediocri, nel valutare i passaggi algebrici degli esercizi della categoria Analisi, grazie al suo addestramento prettamente di stampo matematico.
- **Grok** mostra i limiti di un'architettura multimodale ancora acerba. Vittima del suo modulo OCR poco sviluppato, non è riuscito a dare il meglio di sé proprio per lo scopo per cui era stato selezionato.
- **GPT-OSS** rappresenta l'anello debole a causa della sua inferiorità prestazionale rispetto agli altri modelli. In veste di esecutore, è stato il peggiore globalmente, mentre, in quello di professore, è vittima di un severo *leniency bias*: messo "in soggezione" da modelli più performanti, si limita ad assegnare voti massimi, azzerando la propria utilità valutativa.

6.2 Prospettive Future

In conclusione, affidare la correzione automatica di elaborati di questo tipo a un singolo modello risulta oggi un approccio fallimentare. Alcune idee di sviluppo per espandere questa tesi potrebbero essere:

- **Ampliamento della scelta dei Modelli**: una naturale estensione consisterebbe nel confrontare modelli base con le loro forme più avanzate per osservare il loro comportamento di fronte allo stesso problema. Tuttavia la possibilità di questo approccio si scontra con un limite di tempo: il dataset di confronto iniziale richiede una valutazione manuale. Non essendo questo processo automatizzabile (non si può far fare all'IA da valutatrice delle sue stesse soluzioni), la ricerca rimane vincolata alla durata necessaria di questo passaggio intermedio.

- **Raffinamento del Prompt per la Diminuzione dei Bias**: potrebbe essere di interesse indagare se tecniche di prompting avanzate come il *Few-Shot Prompting* [4] possano arginare il *leniency bias*. Questo permetterebbe di testare se sia possibile "forzare" i modelli ad aggirare i loro filtri di accondiscendenza, per migliorare l'assegnazione dei voti intermedi, addestrandoli sulla differenza coi voti massimi o minimi.

- **Evoluzione verso Valutazione Miste**: Anziché mirare ad una valutazione IA completamente autonoma, le ricerche future potrebbero esplorare l'efficacia di questi modelli come assistenti nella valutazione [17]. "Pre-processando" l'elaborato per evidenziare errori algebrici o algoritmici, l'IA lascerebbe all'umano l'esclusiva responsabilità di "pesare" l'errore, con possibili risparmi di tempo e spunti ulteriori di valutazione.

Bibliografia

[1] McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1955). *A proposal for the Dartmouth summer research project on artificial intelligence*. Archivi dell'Università di Stanford.

[2] Vaswani, A., et al. (2017). *Attention is all you need*. arXiv.org.

[3] Fedus, et al. (2022). *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. jmlr.org.

[4] Brown, T., et al. (2020). *Language models are few-shot learners*. arXiv.org.

[5] Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the knowledge in a neural network*. arXiv.org.

[6] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*. OpenAI.

[7] Ouyang, L., et al. (2022). *Training language models to follow instructions with human feedback*. arXiv.org.

[8] Kaplan, J., et al. (2020). *Scaling laws for neural language models*. arXiv.org.

[9] Kahneman, D. (2011). *Thinking, fast and slow*. Researchgate.com.

[10] Google DeepMind. (2025). *Gemini 3 Pro Model Card*. Report Ufficiale.

[11] OpenAI. (2025). *GPT-OSS-120B Model Card*. Report Ufficiale.

[12] DeepSeek-AI. (2025). *DeepSeek-V3.2 Model Card*. Report Ufficiale.

[13] xAI. (2026). *Grok 4.1 Model Card*. Report Ufficiale.

[14] Zheng, L., et al. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. arXiv.org.

[15] Grandini, et al. (2020). *Metrics for multi-class classification: an overview*. arXiv.org.

[16] Kartik, C., et al. (2025). *Judging the Judges: Evaluating Alignment and Vulnerabilities in LLMs-as-Judges*. aclanthology.org.

[17] Kasneci, E., et al. (2023). *ChatGPT for good? On opportunities and challenges of large language models for education*. sciencedirect.com.