Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Valutazione della capacità dei large language model di simulare vari livelli di abilità

Relatore:
Chiar.mo Prof.
Paolo Ciancarini

Presentata da:
Mauro Molari

III Sessione
Anno Accademico 2023/2024

Parole chiave

Large Language Model

Giochi

Tris

Prompting

Intelligenza Artificiale

Alla mia famiglia,
per avermi sempre supportato durante gli alti e bassi di questi anni.

Sommario

Negli ultimi anni abbiamo potuto assistere all'avvento dei Large Language Model (LLM) come *ChatGPT* di OpenAI o *Claude* di Anthropic. Questi modelli sono in grado di svolgere svariate attività ricevendo come input delle istruzioni in linguaggio naturale. In particolare, questi modelli sono anche in grado, seppur in maniera limitata, di giocare a vari giochi di scacchiera, come il tris o gli scacchi. In questa tesi, propongo un metodo per valutare se questi modelli sono in grado di variare la propria abilità in questo genere di giochi, creando istruzioni che specificano il livello di abilità desiderato e sottoponendo i modelli, con queste istruzioni, a una serie di posizioni del gioco del tris generate casualmente. I risultati ottenuti mostrano che i modelli sono in grado di variare la propria abilità in base alle istruzioni ricevute, anche se con risultati non sempre coerenti con le aspettative.

Indice

1 Introduzione **1**

2 Stato dell'arte **3**

2.1 Modelli AI specifici per giochi 3

2.1.1 AlphaGo 3

2.1.2 AlphaGo Zero 4

2.1.3 AlphaZero 5

2.1.4 MuZero 5

2.1.5 Stockfish 5

2.2 Large language model 6

2.2.1 Nozioni importanti 6

2.2.1.1 Tokenizzazione 6

2.2.1.2 Self-attention 7

2.2.1.3 Fine-tuning 8

2.2.2 Tecniche di prompting 8

2.2.2.1 Prompt engineering 9

2.2.2.2 Zero-shot learning, One-shot learning, Few-shot learning . 9

2.2.2.3 Chain of Thought 9

2.2.3 Modelli *reasoning* 9

3 Metodologia e progettazione **11**

3.1 Il gioco del tris 11

3.2 Struttura dell'esperimento 12

3.2.1 Scrittura dei prompt 13

3.2.2 Generazione delle posizioni 14

3.2.3 Simulazione delle posizioni 15

3.2.4 Raccolta dei dati 15

3.2.5 Esperimento sul modello reasoning 16

3.3 Tecnologie utilizzate 16

3.3.1 OpenRouter 16

3.3.2 Scelta dei modelli 17

3.3.3 Scelta modello *reasoning* 20

3.4 Struttura del codice sorgente . . . 21
3.4.1 La classe OpenRouter . . . 21
3.4.2 La classe TicTacToe . . . 22

4 Risultati **25**
4.1 Sommario esperimento . . . 25
4.2 Risultati modelli standard . . . 25
4.2.1 Analisi sommaria dei risultati . . . 26
4.3 RQ1: Analisi statistica . . . 27
4.3.1 Normalità . . . 28
4.3.2 Omogeneità della varianza . . . 29
4.3.3 Indipendenza dei dati . . . 29
4.3.4 Test ANOVA . . . 30
4.4 RQ2: Analisi dei prompt più interessanti . . . 30
4.4.1 Analisi del prompt beginner . . . 31
4.4.2 Analisi del prompt worse . . . 32
4.5 RQ3: Risultati modello reasoning . . . 32

5 Conclusioni e sviluppi futuri **35**
5.1 Conclusioni . . . 35
5.2 Research answer . . . 36
5.2.1 RQ1 - I large language model sono in grado di variare la propria abilità di gioco semplicemente cambiando le istruzioni che gli vengono date? . . . 36
5.2.2 RQ2 - Ci sono specifici prompt che danno risultati significativamente diversi rispetto ad altri? . . . 36
5.2.3 RQ3 - Le conclusioni ottenute nelle RQ1 e RQ2 vengono confermate anche utilizzando modelli con abilità di pensiero (*reasoning models*)? 36
5.3 Limitazioni . . . 37
5.4 Sviluppi futuri . . . 37
5.5 Costi . . . 37

Bibliografia **39**

1 Introduzione

Nel 2017, alcuni ricercatori di Google rilasciano un articolo, dal titolo *Attention is All You Need*[1], in cui propongono un modello di rete neurale chiamato *Transformer*. Questo modello si rivela essere molto più efficiente e preciso rispetto ai modelli precedenti, e qualche anno dopo diventa la base di un'intera nuova categoria di modelli di intelligenza artificiale chiamati *Large Language Model* (LLM), con il rilascio di *GPT* (*Generative Pre-trained Transformer*) [2] di OpenAI. I large language model hanno la caratteristica di essere estremamente abili nell'interpretare e generare testo in linguaggio naturale, e sono in grado di svolgere svariati compiti, come traduzione, risposta a domande, generazione di articoli, e molto altro. Tra i modelli più conosciuti si possono citare *ChatGPT* di OpenAI, *Claude* di Anthropic, *Gemini* di Google e *Llama* di Meta.
Le capacità di questi modelli non sono conosciute a priori, ma è necessario valutarle empiricamente. Questo perché il modo in cui i large language model sono addestrati e le loro dimensioni, che possono arrivare oltre i centinaia di miliardi di parametri, li rendono di una complessità tale da far risultare impossibile la loro analisi tramite metodi tradizionali. Ed è proprio tramite queste valutazioni che si è scoperta l'abilità dei large language model di comprendere e giocare a vari giochi, sia semplici come il tris, sia più complessi come gli scacchi. Nonostante ciò, il livello di esperienza dei large language model in questi giochi è limitato rispetto allo stato dell'arte. La caratteristica che contraddistingue questi modelli rispetto ad altri modelli specializzati è la capacità di interpretare istruzioni in linguaggio naturale (anche chiamate *prompt*), e sopratutto di dare all'utente la possibilità di modificarle a piacimento, senza aver bisogno di riallenare l'intero modello.
In questa tesi voglio sfruttare questa caratteristica per valutare se i large language model sono in grado di variare la propria abilità di gioco semplicemente cambiando le istruzioni che gli vengono date. Per fare ciò, propongo un *framework* che permette di valutare le abilità di un modello sottoponendolo a posizioni casuali del gioco del tris. In particolare, il framework permette di dare istruzioni diverse allo stesso modello, in modo da poter modificare il modo in cui esso si comporta. I risultati ottenuti sono poi raccolti e analizzati per valutare se i modelli testati riescono a giocare meglio o peggio a seconda delle istruzioni ricevute.

Ho individuato quindi le seguenti *research question*:

Research Question 1 (RQ1):
I large language model sono in grado di variare la propria abilità di gioco semplicemente cambiando le istruzioni che gli vengono date?

Research Question 2 (RQ2):
Ci sono specifici prompt che danno risultati significativamente diversi rispetto ad altri?

Research Question 3 (RQ3):
Le conclusioni ottenute nelle RQ1 e RQ2 vengono confermate anche utilizzando modelli con abilità di pensiero (*reasoning models*)?

Questa tesi è divisa in cinque capitoli:

- **1 - Introduzione**, In cui viene dato il contesto in cui si colloca il lavoro svolto e vengono presentati gli obiettivi e le motivazioni della tesi.
- **2 - Stato dell'arte e nozioni di base**, In cui vengono illustrati vari modelli di intelligenza artificiale e le loro capacità, sia in generale che in ambito ludico. Partirò da modelli AI specifici per il gioco, per poi arrivare ai large language model e alle loro capacità.
- **3 - Metodologia e implementazione**: In cui verranno spiegate le scelte effettuate per la realizzazione del framework, le tecnologie utilizzate e la metodologia di esecuzione dell'esperimento.
- **4 - Risultati**: In cui verranno presentati i risultati ottenuti durante la fase di sperimentazione. In particolare, verranno fatte varie ipotesi sulle prestazioni dei prompt e verranno confrontati i risultati ottenuti con quelli attesi. Infine, si cercherà di dare una spiegazione ai risultati ottenuti.
- **5 - Conclusioni e sviluppi futuri**: In cui verranno presentate le conclusioni ottenute e verranno proposti possibili sviluppi futuri del lavoro svolto.

2 Stato dell'arte

2.1 Modelli AI specifici per giochi

In questa sezione vengono presentati alcuni modelli di intelligenza artificiale che hanno ottenuto risultati significativi nel campo dei giochi. Questi modelli sono stati sviluppati per giocare a specifici giochi, e sono stati progettati per farlo nella maniera migliore possibile. Particolare attenzione viene data a modelli che sono stati allenati tramite *reinforcement rearning* (RL), una tecnica di apprendimento automatico che permette di insegnare ad un modello a svolgere compiti complessi tramite l'interazione con un ambiente di gioco.

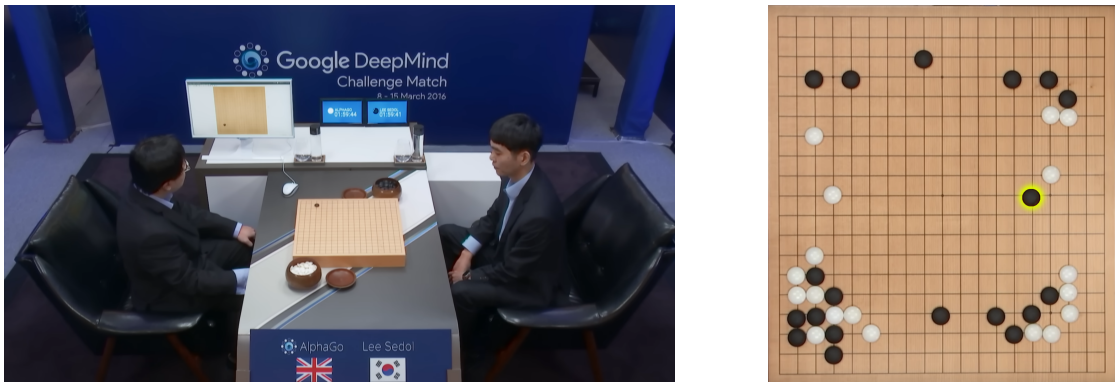

Figura 2.1: A sinistra: Aja Huang, capo ingegnere di DeepMind, replicando le mosse di Lee Sedol in un'interfaccia grafica collegata ad AlphaGo, ed eseguendo le mosse del modello nella tavola reale. A destra: Tavola di Go presa dal secondo incontro del DeepMind Challenge Match, alla mossa 37, giocata da AlphaGo. Questa mossa venne immediatamente considerata un errore da parte dei commentatori, salvo poi rivelarsi una mossa cruciale per la vittoria dell'AI. Fonte:[3]

2.1.1 AlphaGo

AlphaGo[4] è un programma di intelligenza artificiale capace di giocare al gioco da tavolo *Go*. È stato sviluppato da DeepMind, sussidiario di Google, nel 2015. Il gioco del Go è tutt'oggi considerato uno dei giochi da tavolo più complessi al mondo, data l'ampiezza notevole del suo spazio di ricerca e la difficoltà nel valutarne le posizioni. Il fatto che il Go sia un gioco ad informazione perfetta, ovvero che entrambi i giocatori conoscono lo stato del gioco in qualsiasi momento, fa sì che sia teoricamente possibile utilizzare

tecniche di ricerca ricorsiva per valutare ogni possibile mossa e trovare quella migliore. Tuttavia, l'enorme profondità dell'albero decisionale rende questa strategia impossibile in tempi ragionevoli. AlphaGo cerca di risolvere questo problema utilizzando due reti neurali separate: Una che ha il compito di valutare una posizione, e l'altra che ha il compito di selezionare la mossa ritenuta migliore. Queste due reti sono addestrate con una combinazione di *supervised learning*, che in questo caso consiste nel fornire al modello un set di dati preso da partite di giocatori umani professionisti, e *reinforcement learning*, che consiste nel far giocare il modello contro se stesso e premiare le mosse che portano alla vittoria. Questo approccio ha permesso ad AlphaGo di sconfiggere il campione del mondo coreano Lee Sedol nel 2016 vincendo 4 incontri su 5, in quello che è comunemente conosciuto come il *DeepMind Challenge Match*[5].

2.1.2 AlphaGo Zero

Successore di AlphaGo, AlphaGo Zero[6](2017) amplia il modello costruito da DeepMind, ma rimuove la parte di supervised learning. Questo significa che il modello non ha bisogno di partite di giocatori umani per imparare a giocare, ma riesce ad allenarsi giocando contro se stesso, conoscendo solamente le regole del gioco. Le due reti neurali del predecessore vengono combinate in una sola, molto più efficiente. AlphaGo Zero riesce quindi, in soli tre giorni, a raggiungere il livello di abilità di AlphaGo. Dopo 40 giorni diventa, presumibilmente, il miglior giocatore di Go al mondo. AlphaGo Zero è quindi un importante risultato nel mondo dell'intelligenza artificiale, in quanto dimostra come non ci sia bisogno di dati etichettati per allenare un modello, ma che questo possa imparare semplicemente interagendo con l'ambiente su cui deve agire.

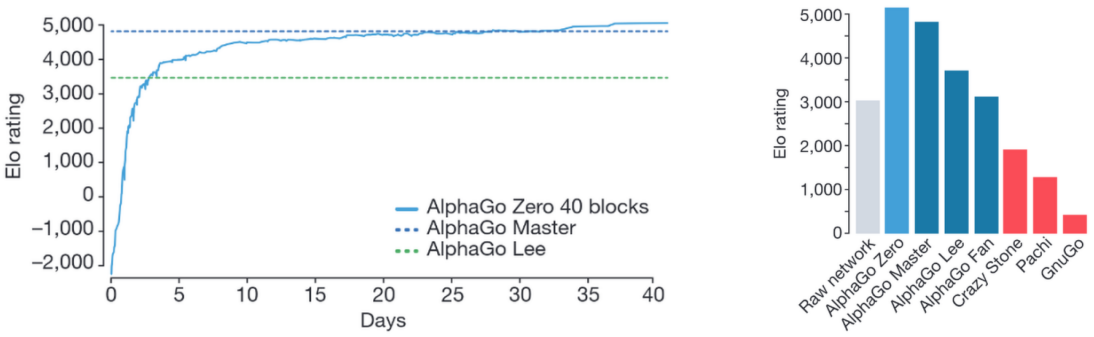

Figura 2.2: A sinistra: Grafico che mostra il tempo di allenamento di AlphaGo Zero. Si può vedere come il modello abbia raggiunto il livello di AlphaGo in soli tre giorni, e come abbia continuato a migliorare fino a diventare il migliore giocatore di Go al mondo. A destra: Grafico che mostra il confronto tra AlphaGo Zero e altre AI per Go. Fonte:[6]

2.1.3 AlphaZero

Continuando la serie di successi, DeepMind sviluppa, l'anno successivo, AlphaZero[7]. Questo modello continua a espandere sull'architettura di AlphaGo Zero, ma generalizzandola. Ciò ha permesso al modello di imparare a giocare, oltre al Go, anche a shogi e a scacchi. Infatti, AlphaZero riesce con successo a imparare a giocare a questi giochi, arrivando a battere dei campioni del mondo in tutti e tre. È importante far notare come AlphaZero utilizzi lo stesso algoritmo e la stessa architettura di rete neurale per tutti e tre i giochi, e l'unica cosa che gli viene data come input sono le regole e l'ambiente di gioco.

2.1.4 MuZero

Culmine della serie di modelli partita da AlphaGo, MuZero[8] viene sviluppato nel 2019, e riprende l'architettura del suo predecessore AlphaZero, ma con alcune importanti differenze che lo rendono molto più generale. Infatti, se tutti i modelli precedenti ad esso nella sua famiglia avevano bisogno di conoscere le regole del gioco per poter essere allenati, MuZero è in grado di apprendere anch'esse tramite l'interazione con l'ambiente fornito. Ciò è stato dimostrato infatti, oltre che allenando il modello sui giochi testati in precedenza senza fornirne le regole, anche su una collezione di 57 giochi per la console *Atari 2600*, riuscendo a raggiungere anche in questo caso lo stato dell'arte in termini di abilità. MuZero rappresenta quindi un enorme passo avanti per le tecniche di allenamento dette *unsupervised learning*, in cui una rete neurale è in grado di apprendere un compito senza avere bisogno di dati etichettati.

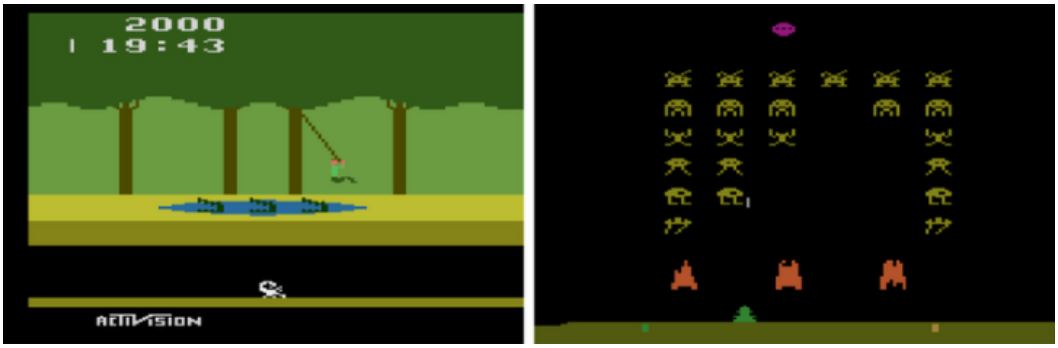

Figura 2.3: Esempio di giochi Atari 2600 su cui MuZero è stato allenato. In particolare, a sinistra *Pitfall!* e a destra *Space Invaders*. Fonte:[9]

2.1.5 Stockfish

Stockfish[10] è un motore di scacchi open-source, originariamente sviluppato da Tord Romstad, Marco Costalba, e Joona Kiiski. È stato rilasciato per la prima volta nel 2008, e da allora è diventato uno dei motori di scacchi più forti al mondo grazie alla sua natura aperta, che permette a chiunque di contribuire al suo sviluppo. Stockfish non utilizza un

approccio puramente neurale come i modelli di DeepMind, ma si basa principalmente su tecniche di ricerca euristica, ed utilizza una rete neurale solamente per l'analisi finale delle posizioni chiamato *NNUE*, ovvero *Neural Network Universal Evaluation*. In particolare, Stockfish è costituito da tre parti principali: la rappresentazione del tavolo di gioco, la ricerca di mosse tramite albero euristico, e la valutazione della posizione tramite NNUE. È uno dei motori di scacchi più forti al mondo, che è riuscito ad arrivare sempre nei primi posti della *Top Chess Engine Championship*[11][12], una competizione annuale tra i migliori motori di scacchi al mondo.

2.2 Large language model

I *large language model* (LLM) sono dei modelli di intelligenza artificiale che hanno come obiettivo principale la generazione di testo. Essi sono basati sul tipo di rete neurale chiamato *Transformer*, sviluppato da alcuni scienziati di Google nel 2017[1]. Questi modelli sono basati su un meccanismo chiamato *self-attention*, che permette di mettere in relazione ogni parola del testo dato in input con tutte le altre, e di dare un peso diverso a ciascuna di esse in base al contesto. Questo meccanismo, unito all'enorme quantità di dati su cui i large language model vengono allenati, hanno permesso loro di diventare uno degli strumenti più utilizzati in tutto il mondo, grazie alle loro molteplici applicazioni. Tra queste si può citare la traduzione tra vari linguaggi, la risposta a domande e il parsing di dati non strutturati, oltre che la generazione di vari tipi di testo. Nel testare le capacità dei large language model, si è scoperto che questi hanno anche una certa abilità nel giocare a vari giochi, sia semplici come il tris, sia più complessi come gli scacchi.

2.2.1 Nozioni importanti

2.2.1.1 Tokenizzazione

I large language model, al contrario di quanto si possa pensare, non lavorano direttamente con singoli caratteri o parole, ma con delle unità chiamate *token*. Questi token sono degli *n-grammi* di caratteri, e un vocabolario di essi viene generata prima di allenare il modello. Ciò viene fatto analizzando tutti i testi presenti nel dataset di allenamento, e prendendo gli n-grammi più frequenti. Questo permette al modello di avere una rappresentazione più compatta del tipo di testo su cui deve lavorare, e di aumentare l'efficienza computazionale. Il vocabolario generato verrà poi utilizzato dal *tokenizer* del modello, che ha il compito di convertire il testo dato in input in una sequenza di token. Questo processo è chiamato, appunto, *tokenizzazione*. Ciò comporta, però, anche degli svantaggi. Un esempio molto conosciuto è quello del conteggio delle lettere in una parola. Se, per esempio, si chiede ad un large language model di contare quante lettere R ci sono nella parola *carrier*, il modello non riuscirà sempre a rispondere correttamente (come mostrato nella Figura 2.4 a

fronte). Per capire il perché, è possibile analizzare come la parola carrier viene tokenizzata. Prendendo come esempio il modello GPT-4o di OpenAI, e andando sulla sezione del loro sito dedicata alla tokenizzazione, si può vedere come la parola carrier sia così comune da essere considerata un token a se stante.[13] Ciò significa che il modello vede questa parola come un singolo token, e non come una sequenza di lettere. Per questo motivo, il modello non riesce a contare accuratamente le lettere R all'interno di essa, in quanto non ha la capacità di suddividere i token in sotto-token. Questo è un problema che affligge tutti i large language model, e che può portare a risultati inaspettati se non si è a conoscenza di esso. Lo stesso problema ha rilevanza anche nel contesto dei giochi, in quanto molti di essi si basano su una certa relazione spaziale fra vari oggetti o simboli. Se si prende come esempio il Go, ci si accorge subito di come sia fondamentale riuscire a vedere ogni pietra singolarmente per poter valutare correttamente una posizione, cosa che i large language model faticano a fare. Recentemente, Meta ha pubblicato un articolo in cui viene presentata una nuova architettura per large language model, chiamata *Byte Latent Transformer*, che lavora direttamente sui byte invece che su token, e ha un'efficienza comparabile a quella dei transformer tradizionali, risolvendone però i problemi dati dalla tokenizzazione[14].

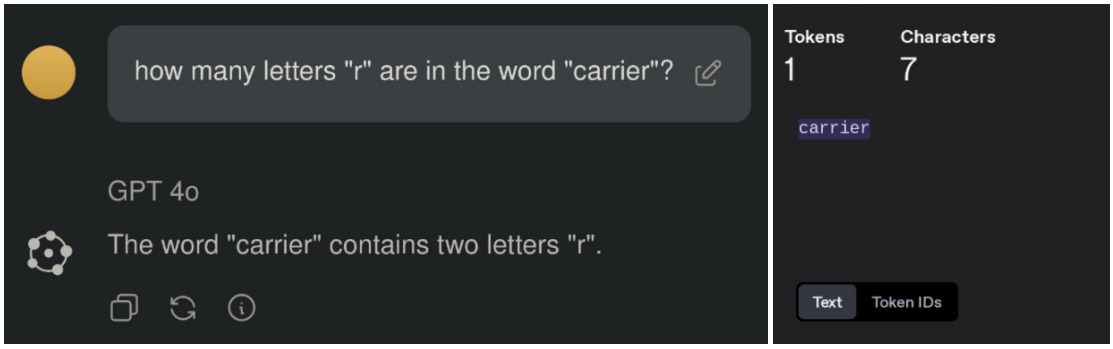

Figura 2.4: A sinistra: Esempio di chat con GPT-4o di OpenAI, in cui viene chiesto di contare le lettere R nella parola carrier. A destra: Esempio di tokenizzazione della parola carrier da parte del tokenizer di GPT-4o di OpenAI.[13]

2.2.1.2 Self-attention

Il meccanismo di *self-attention* è una tecnica di *machine learning* che determina l'importanza relativa di ciascun componente di una sequenza relativamente a tutti gli altri componenti. Nel caso dei large language model, questi componenti sono i token.

la self-attention è composta da tre parti principali:

- **Query**: Rappresenta la parola interrogante
- **Key**: Definisce come ogni parola può rispondere alle query
- **Value**: Contiene l'informazione effettiva da propagare

Per ogni token si calcola un **peso di attenzione** rispetto a tutti gli altri token utilizzando questi tre componenti, e si combinano, permettendo di dare più importanza a certi token rispetto ad altri e di creare una rappresentazione più complessa e contestualizzata del testo. Questo meccanismo è alla base dei modelli basati su transformer, e permette di creare modelli molto più espressivi e utili rispetto a quelli basati su architetture precedenti.

Nei modelli transformer questa operazione viene replicata in parallelo su più teste (*multi-head*), ognuna specializzata in diversi tipi di relazioni (sintattiche, semantiche, contestuali). Questo permette una maggiore parallelizzazione e una maggiore capacità di apprendimento.

2.2.1.3 Fine-tuning

I large language model hanno, di base, più o meno competenza in vari compiti di *natural language processing*. Per poter migliorare le capacità su un compito specifico, si può ricorrere a una tecnica chiamata *fine-tuning*. Questa tecnica consiste nel prendere un modello pre-allenato su un dataset molto grande, e allenarlo ulteriormente su un dataset più piccolo, preparato specificatamente con dati che riguardano il compito di cui si vogliono migliorare le capacità. Questo permette al modello di adattarsi meglio a una specifica situazione, e di ottenere risultati migliori rispetto a un modello grezzo. Il fine-tuning è un processo che può portare a risultati molto positivi, ma ha anche bisogno di un'elevata capacità di calcolo, il che lo rende spesso non accessibile a tutti. Per questo motivo, per molti compiti, spesso si preferisce utilizzare modelli pre-allenati senza fine-tuning, e concentrarsi invece sulle tecniche di *prompt engineering*, ovvero la scelta di prompt che permettano al modello di generare testo coerente e corretto. Uno dei modelli più usati dalla comunità open-source su cui effettuare fine-tuning è Llama di Meta, in quanto è disponibile in varie dimensioni e permette di ottenere risultati molto buoni con un costo computazionale accettabile. Molte aziende che offrono API per l'utilizzo di large language model permettono anche di fare fine-tuning utilizzando i loro modelli sui loro server. Questo processo comporta però spesso costi abbastanza elevati, e il modello ottenuto dopo il fine-tuning rimane utilizzabile solo tramite l'API stessa.

2.2.2 Tecniche di prompting

Una parte molto importante nell'utilizzo dei large language model è la scelta del *prompt*, ovvero delle istruzioni che vengono date al modello per generare il testo. Queste istruzioni sono fondamentali per ottenere risultati coerenti e corretti, e spesso la scelta di un buon prompt può fare la differenza tra un testo ben scritto e uno che non ha senso. Esistono varie tecniche per migliorare la qualità dei prompt.

2.2.2.1 Prompt engineering

Il prompt engineering è definito come l'insieme di tecniche che permettono di scrivere prompt che guidano meglio il modello, così da assicurarsi che esso sia in grado di generare testo in linea con le istruzioni che si stanno dando ad esso. Questo processo richiede una certa abilità nel capire come funziona il modello, e come interpreta le istruzioni che gli vengono date. Il processo di prompt engineering può essere fondamentale per ottenere risultati migliori, e spesso è preferibile al fine-tuning, in quanto non richiede alcun costo aggiuntivo.

2.2.2.2 Zero-shot learning, One-shot learning, Few-shot learning

Quando si sta scrivendo un prompt per risolvere un qualsiasi problema, ci sono vari modi di presentarlo al modello. Se si vuole che questo risolva un problema senza aver mai visto nessun esempio, si parla di *zero-shot learning*. Questo è un compito molto difficile, e spesso richiede un modello molto grande e ben allenato per ottenere prestazioni accettabili. Se si vuole invece che il modello risolva un problema dopo aver fornito un solo esempio, si parla di *one-shot learning*. Aggiungendo più esempi, si inizia a parlare invece di *few-shot learning*. Queste tecniche sono molto utili per risolvere problemi specifici, e spesso permettono di ottenere risultati molto buoni con un costo computazionale accettabile. Vari esperimenti effettuati in un articolo chiamato *Large Language Models are few-shot learners*[15] dimostrano come fornire anche solo pochi esempi al modello possa migliorare notevolmente le sue capacità e la sua intelligenza.

2.2.2.3 Chain of Thought

Parlando di *Chain of Thought* si intende una tecnica di prompting in cui, invece di chiedere semplicemente al modello di svolgere un determinato compito, si chiede inoltre ad esso di procedere allo svolgimento del compito passo a passo, e di pensare ad alta voce. Ciò è fatto fornendo al modello un esempio del metodo di ragionamento, che esso poi cercherà di seguire. sono state effettuate varie ricerche sull'efficacia di questa tecnica, come per esempio nell'articolo *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* di Wei Et al. [16], in cui vengono riportati notevoli miglioramenti sullo stesso modello dopo aver dato al large language model solo 8 esempi di ragionamento passo a passo.

2.2.3 Modelli *reasoning*

Verso la fine del 2024, OpenAI rilascia un nuovo modello, dal nome in codice *gpt-o1-preview* [17]. Questo modello riesce a superare molti dei benchmark dei large language model rilasciati in precedenza con punteggi molto alti, e riesce a diventare il nuovo stato dell'arte in termini di abilità. La particolarità di questo modello è infatti quella di essere in grado, prima di dare una risposta, di pensare e ragionare sul prompt dato in input. Il modo

in cui il modello sembra ragionare è molto simile a quello della tecnica del Chain of Thought, con la differenza che, nel caso dei modelli reasoning, l'abilità di ragionamento è integrata direttamente nel modello, senza aver bisogno di prompting aggiuntivo. Infatti, il large language model impara a ragionare direttamente dai dati su cui viene allenato, ed è in grado di imparare a farlo in modo completamente autonomo, come comportamento emergente dal reinforcement learning che guida l'allenamento del modello [18]. L'emergenza spontanea di questo comportamento fa sì che il modo di ragionare imparato dal modello sia molto più efficace e generalizzabile rispetto a quello che si potrebbe ottenere con il semplice prompting, il che porta ad un aumento delle capacità generali del modello.

3 Metodologia e progettazione

In questo capitolo vengono presentate le specifiche dell'esperimento che è stato condotto per questa tesi. In particolare, vengono descritte la varie fasi di progettazione, le scelte fatte e le motivazioni che hanno portato a tali decisioni. Inoltre, vengono descritte le fasi di implementazione del sistema, includendo le tecnologie utilizzate e il perché della loro scelta.

3.1 Il gioco del tris

Il gioco del tris (*tic-tac-toe* in inglese) è un gioco molto semplice, che viene giocato su una scacchiera 3x3. I giocatori, che si alternano, devono inserire il proprio simbolo (di solito una X o una O) in una casella vuota. Lo scopo del gioco è quello di allineare tre simboli dello stesso tipo in orizzontale, verticale o diagonale. Ho deciso di selezionare come gioco per l'esperimento il tris per diversi motivi:

- È un gioco risolto, ovvero è un gioco in cui è possibile determinare il risultato di una partita partendo da una qualsiasi posizione, assumendo che entrambi i giocatori giochino in modo ottimale. Questa caratteristica del tris rende possibile il poter calcolare, per una qualsiasi posizione, le mosse ottimali possibili. Ciò mi permette di valutare le risposte del modello in modo preciso, e di poter confrontare i risultati ottenuti con quelli di un algoritmo che calcola le mosse ottimali.

- È un gioco molto semplice, che richiede poche regole per essere giocato. Questo mi permette di poter implementare l'ambiente di gioco in modo abbastanza semplice, e di non dover mandare troppe informazioni al modello.

- È un gioco molto conosciuto. Ciò potrebbe significare che il modello ha già visto alcune partite di tris, e che quindi potrebbe essere in grado di giocare in modo migliore rispetto ad altri giochi, rafforzando la validità dell'esperimento.

È possibile calcolare, tramite una simulazione, il numero di posizioni legali del gioco del tris. Questo numero è pari a 5478 se si considerano tutte le posizioni possibili, e a 765 se si considerano solo le posizioni uniche, ovvero eliminando quelle che sono simmetrie di altre [19]. Questo numero è molto basso rispetto ad altri giochi, come per esempio gli scacchi. Prendendo una stima effettuata da John Tromp tramite una simulazione in *Haskell*, si trova

come il numero di posizioni legali degli scacchi sia circa pari a $(4.82 \pm 0.03) * 10^{44}$, con un intervallo di confidenza del 95%[20]. Questo numero eclissa completamente quello delle posizioni possibili nel tris, e dimostra come quest'ultimo sia un gioco piuttosto semplice. Tuttavia, la bassa complessità del gioco del tris mi permette di contenere la difficoltà della valutazione dei risultati, e di concentrarmi sulle prestazioni del modello in sé.

3.2 Struttura dell'esperimento

L'esperimento è stato strutturato in modo da poter testare il modello e il variare della sua abilità in un ambiente controllato. Come già detto in precedenza, con questo esperimento voglio prevalentemente testare l'effetto di vari prompt sulle capacità del modello, e non la sua abilità assoluta. L'esperimento è stato diviso in più parti:

- la **scrittura dei prompt**, in cui vengono scritte le varie istruzioni da passare al modello per farlo giocare a tris con un determinato livello di abilità.
- la **generazione delle posizioni**, in cui vengono generate le varie posizioni di gioco, e vengono scelte quelle che verranno utilizzate per testare il modello.
- la **simulazione delle posizioni**, in cui vengono mandate le posizioni al modello insieme alle istruzioni, e vengono raccolte le risposte.
- la **valutazione delle risposte**, in cui vengono valutate le risposte del modello, e vengono contate le mosse ottimali e non ottimali.
- la **valutazione statistica**, in cui vengono calcolate le statistiche relative ai vari prompt, e vengono confrontate e analizzate per trovare eventuali differenze significative.

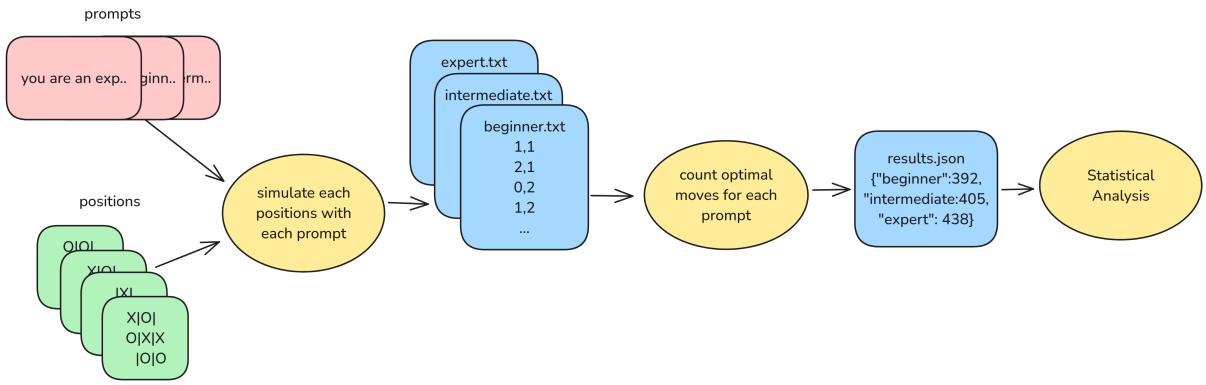

Figura 3.1: Diagramma di flusso dell'esperimento

3.2.1 Scrittura dei prompt

Per effettuare un qualsiasi compito specifico, i large language model hanno bisogno di ricevere delle istruzioni. Queste istruzioni vengono chiamate *prompt*, e sono fondamentali per il funzionamento del modello. Dato che l'obiettivo dell'esperimento era di analizzare l'abilità di un modello al variare del livello di abilità che gli viene detto di avere nel prompt stesso, ho dovuto preparare multipli prompt diversi. Per fare ciò, ho deciso di dividere le istruzioni in due parti: La prima parte in cui viene detto al modello che livello di abilità deve avere, la seconda in cui viene spiegato il gioco del tris e come deve essere strutturata la risposta del large language model. Questo mi ha permesso di riutilizzare la seconda parte, variando solamente la prima. Le istruzioni sono state scritte in inglese, essendo il linguaggio in cui i modelli hanno visto più dati, e che quindi dovrebbero comprendere meglio. Inoltre, ho cercato di scrivere le istruzioni in modo chiaro e diretto, in modo da cercare di non confondere il large language model. Per prima cosa, viene descritto al modello il suo ruolo. Ho deciso di testare cinque prompt diversi:

- **Default**: Non viene dato nessun ruolo specifico al modello. Questo permette di testare il modello senza dare alcuna informazione su come dovrebbe giocare.
- **Beginner**: Il modello è un giocatore alle prime armi, che non ha molta esperienza nel gioco del tris. Non è molto bravo a prevedere le mosse dell'avversario, e non conosce alcuna strategia.
- **Intermediate**: Il modello è un giocatore che ha una certa esperienza nel gioco del tris, e che conosce alcune strategie di base. È in grado di prevedere alcune mosse dell'avversario, e di evitare di perdere in modo evidente.
- **Expert**: Il modello è un giocatore molto esperto, che conosce tutte le strategie del gioco del tris, e che è in grado di prevedere tutte le mosse dell'avversario. È molto difficile da battere.
- **Worse**: Al modello viene specificatamente indicato che il suo obiettivo è quello di perdere la partita.

Ovviamente, la descrizione di questi prompt è quello che è stato detto al modello, ma non vuol dire che esso sia in grado di seguire queste istruzioni alla lettera. Infatti, il modello potrebbe non comprendere bene le istruzioni, o potrebbe non essere in grado di seguire del tutto le istruzioni date. Questo è uno dei motivi per cui ho deciso di testare più prompt, in modo da poter valutare l'effetto di queste istruzioni sulle capacità del modello. La prima parte delle istruzioni è stata scritta in modo da essere il più possibile chiara e diretta, e da non variare troppo come lunghezza. Questo per evitare che la lunghezza delle istruzioni influenzi i dati raccolti. La seconda parte delle istruzioni è invece sempre la

stessa, e spiega al modello tutto ciò che deve sapere per giocare a tris nell'ambiente. In particolare, il prompt contiene i seguenti elementi:

- Breve spiegazione del gioco del tris, e come si vince.
- L'obiettivo del modello, ovvero di determinare la mossa che dovrà giocare. È importante notare come si specifica "la mossa che deve giocare", e non "la mossa migliore", in quanto non si vuole influenzare il modello rispetto al ruolo dato nella prima parte.
- Due rappresentazioni della scacchiera di gioco, una in formato testuale in due dimensioni, mentre un'altra in formato json, con un array a una dimensione. Questo è stato fatto per minimizzare il rischio di errori dovuti alla rappresentazione della scacchiera.
- Un elenco di mosse valide, per minimizzare il rischio di errori dovuti a mosse non valide.
- Come deve essere strutturata la risposta del modello, ovvero con una stringa contenente la mossa da giocare, in formato "Move: row, col".
- Il simbolo con cui sta giocando il modello, X oppure O.

Queste due parti di istruzioni vengono poi concatenate, vengono inserite le informazioni dinamiche (come il simbolo con cui gioca il modello e la posizione della scacchiera), e vengono passate al modello. Questo permette di variare il livello di abilità del modello in modo controllato, e di testare l'effetto di queste istruzioni sulle sue capacità.

3.2.2 Generazione delle posizioni

Il metodo che ho utilizzato per valutare l'abilità del modello è quello di fargli credere di star giocando una partita già in corso, e di farlo rispondere con una singola mossa. Questo mi permette di non dovermi preoccupare di come il modello sia arrivato a quella posizione. Infatti, generando abbastanza scacchiere casuali, posso valutare il comportamento del large language model in una vasta gamma di situazioni, rimuovendo quelle più banali, come per esempio una scacchiera vuota o con una sola mossa. Per fare ciò, ho scritto un metodo che genera un numero n di scacchiere casuali del gioco del tris, e le salva in formato json in un file chiamato *boards.json*. Ciò mi permette di poter riutilizzare le stesse posizioni per ogni prompt, e di rimuovere eventuali differenze date da posizioni diverse. Le scacchiere vengono generate in modo molto semplice:

1. Si inizializza una scacchiera vuota
2. Viene scelto un giocatore casuale tra X e O che inizia la partita

3. Si sceglie un numero casuale m di mosse da fare tra due estremi forniti. Nel mio caso, ho scelto di fare tra le 2 e le 7 mosse, in modo da evitare posizioni troppo semplici o con troppe mosse ottimali.

4. per m volte, si sceglie una mossa casuale tra le mosse valide, e si effettua la mossa.

Ho deciso di generare, per ogni esecuzione dell'esperimento, 1000 posizioni casuali, in modo da poter minimizzare gli effetti aleatori, e di poter avere una buona varietà di posizioni. Questo mi permette di valutare con più precisione le differenze di abilità tra i vari prompt, se presenti.

3.2.3 Simulazione delle posizioni

Una volta generate le posizioni da utilizzare, è possibile procedere all'esecuzione dell'esperimento vero e proprio. Come prima cosa si caricano le posizioni da utilizzare dal file *boards.json*, le parti variabili dei prompt, e la parte comune a tutti i prompt. Per ogni posizione, vengono sostituite le informazioni specifiche della posizione nel prompt, e viene inviato al modello. Tutto questo viene fatto per ogni prompt, e per ogni posizione. Le risposte vengono poi raccolte e salvate in vari file di risultati. Questo processo è molto lungo, e richiede parecchio tempo per essere completato. Fortunatamente, OpenRouter mi permette di effettuare numerose richieste in parallelo, e di poter quindi ridurre il tempo di esecuzione dell'esperimento. Ho deciso di utilizzare il maggior numero di richieste parallele possibili permesse dal modello utilizzato, in modo da poter completare l'esperimento nel minor tempo possibile. Questo mi permette di poter eseguire più iterazioni nello stesso periodo di tempo, e di poter ottenere risultati più precisi.

3.2.4 Raccolta dei dati

Dopo aver ricevuto tutte le risposte dei large language model, Ho dovuto prendere la mossa effettuata dal modello per ogni posizione, e confrontarla con le mosse ottimali calcolate da un algoritmo apposito. È importante notare come le mosse ottimali possono essere più di una per una determinata posizione di una scacchiera del gioco del tris. Per esempio, si può pensare ad una partita al secondo turno, dove il primo giocatore ha giocato un simbolo al centro della scacchiera. In questo caso le mosse disponibili reali sono due, giocare in un angolo o giocare in un lato, in quanto l'angolo o il lato scelti non cambiano il fatto che la posizione risultante sarà in ogni caso una rotazione di uno degli altri quattro casi.

Per ogni mossa effettuata, se questa è presente nelle mosse ottimali calcolate, si da un punto al prompt, potendo così calcolare una percentuale di mosse ottimali attraverso la formula:

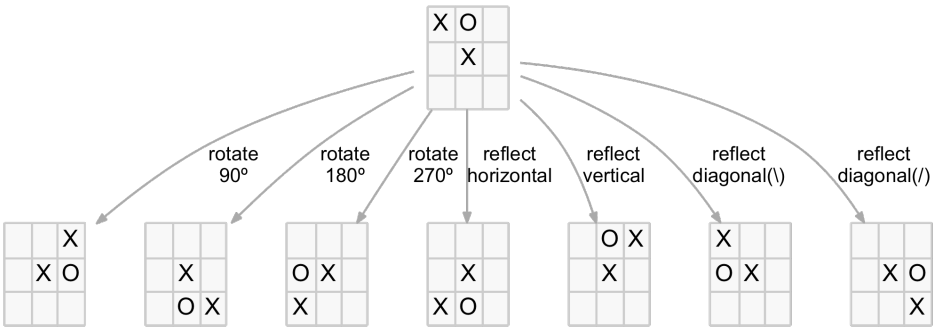

Figura 3.2: Esempio di simmetrie della scacchiera del tris. Fonte: [21]

$$\text{Percentuale} = \frac{\text{Mosse ottimali}}{\text{Mosse totali}} \tag{3.1}$$

3.2.5 Esperimento sul modello reasoning

Per il modello di tipo reasoning, l'esperimento viene eseguito in maniera analoga, ma con alcune differenze:

- Il numero di esecuzioni è ridotto a 3;
- Il numero di posizioni valutate per esecuzione è ridotto a 50.

Questa scelta è stata necessaria per permettere l'inclusione di un modello di questo tipo nell'esperimento, in quanto il loro costo di utilizzo è di molto superiore rispetto ai large language model tradizionali, sia per il loro maggiore consumo computazionale, sia per la lunghezza delle loro risposte.

3.3 Tecnologie utilizzate

3.3.1 OpenRouter

OpenRouter[22] è un sito web che offre un servizio per utilizzare vari modelli di intelligenza artificiale utilizzando un'unica *API* (*Application Programming Interface*). Ho deciso di utilizzare questo servizio per l'esperimento in quanto mi permette di non dover utilizzare multiple API per utilizzare modelli di aziende diverse, ma di poterli invece selezionare semplicemente inserendo il loro nome nel codice. OpenRouter offre un'interfaccia molto semplice e ben documentata, che mi ha permesso di iniziare a utilizzare i modelli in poco tempo. Il servizio adotta il sistema di pagamento *pay-as-you-go*, cioè che mi permette di pagare solo per l'utilizzo effettivo dei modelli, senza dover sottoscrivere abbonamenti ricorrenti. Questo mi permette di mantenere i costi dell'esperimento a livelli accettabili, e di non dover pagare per modelli che non utilizzo. Un ulteriore vantaggio di OpenRouter è quello di offrire automaticamente il bilanciamento di carico tra vari fornitori di un modello,

quando possibile. Il bilanciamento viene effettuato in base al tempo di risposta per il primo token, in base alla velocità in token al secondo, e in base al prezzo offerto dal fornitore. Questo mi permette di non dovermi preoccupare di selezionare manualmente il fornitore migliore, e di poter utilizzare il modello in modo più efficiente, evitando di dovermi preoccupare di eventuali problemi di sovraccarico di un fornitore.

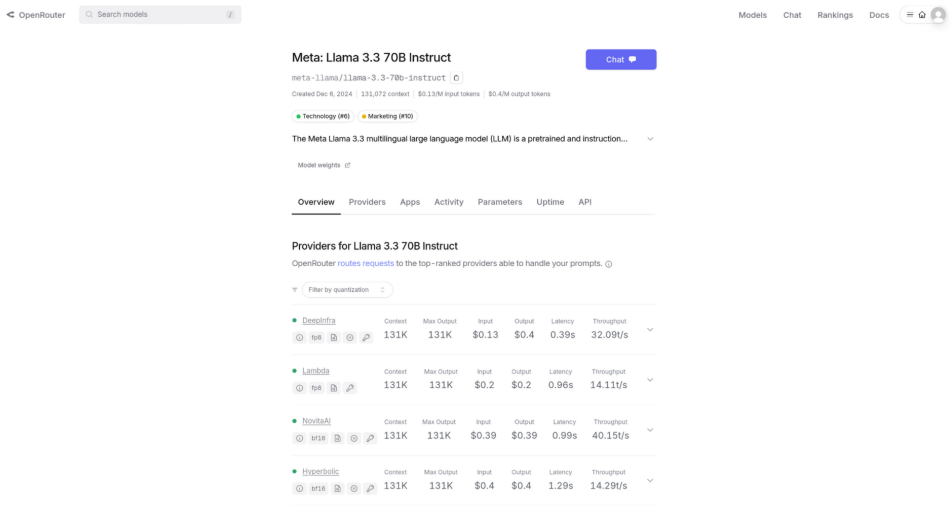

Figura 3.3: Screenshot della pagina di un modello (qui Llama di Meta) su OpenRouter. Si notino le informazioni sul modello in alto e i vari fornitori in basso.

3.3.2 Scelta dei modelli

Dato che l'obiettivo è quello di testare un large language model e valutarne l'abilità relativa nei giochi, è importante che la capacità del modello scelto non pregiudichi la validità dei risultati ottenuti. Per questo motivo, è stato necessario scegliere un modello che fosse in grado di seguire le istruzioni nel miglior modo possibile, e che dimostri una capacità di comprensione adeguata per il compito. Per aiutarmi in questa scelta, mi sono servito di un sito web chiamato Chatbot Arena [23]. Questo servizio dispone infatti di strumenti creati appositamente per comparare le capacità di vari large language model in determinate categorie, e dispone di una classifica che tiene conto di queste valutazioni. A differenza di molti dei benchmark esistenti, come per esempio LogiQA[24] e il più recente MastermindEval[25], in cui la classifica dei modelli è redatta solo sul dataset di problemi del benchmark oppure contiene un numero minore di modelli testati, Chatbot Arena utilizza un approccio più dinamico, in cui la classifica viene costantemente aggiornata e vengono aggiunti nuovi modelli continuamente, grazie ai contributi degli utenti. Questa classifica utilizza infatti un metodo di valutazione che tiene conto di varie battaglie sostenute dai modelli. Ognuna di queste battaglie è composta da una particolare attività data come istruzione a due modelli diversi. Dopo che i due modelli hanno dato la propria risposta, l'utente può decidere quale dei due modelli si è comportato meglio nel seguire le istruzioni date. La valutazione del singolo utente viene poi presa e consolidata con le altre valutazioni

già schedate nel database del servizio, aggiornando il punteggio e la posizione in classifica dei modelli testati. In particolare, tra le molteplici categorie valutate su Chatbot Arena, ho scelto di concentrarmi su tre di esse: *Math*, *Instruction Following* e *Multi-turn*. Sono state scelte queste tre categorie perché ritengo che siano le più rilevanti per ottenere dei buoni risultati nei giochi. In particolare:

- *Math* è una categoria che valuta la capacità di un large language model di risolvere problemi matematici. Questa categoria è stata scelta perché questa abilità è molto legata sia alla capacità di ragionamento che alla capacità del modello di effettuare calcoli, entrambe caratteristiche fondamentali per giocare.
- *Instruction Following* è una categoria che valuta la capacità di un large language model di seguire istruzioni. Questa categoria è stata scelta perché è molto importante che il modello sia in grado di seguire le istruzioni date dal gioco, e che abbia una bassa probabilità di negare la richiesta dell'utente o di effettuare azioni non richieste. Questo torna molto utile per il *parsing* delle azioni del large language model, che è un passaggio fondamentale per il funzionamento del sistema.
- *Multi-turn* è una categoria che valuta la capacità di un large language model di mantenere una conversazione complessa. Questa categoria è stata scelta perché un gioco è solitamente formato da più turni, che possono venire rappresentati come messaggi separati inviati al modello. È quindi importante che esso sia in grado di mantenere una conversazione complessa, e che sia in grado di ricordare le informazioni date in precedenza.

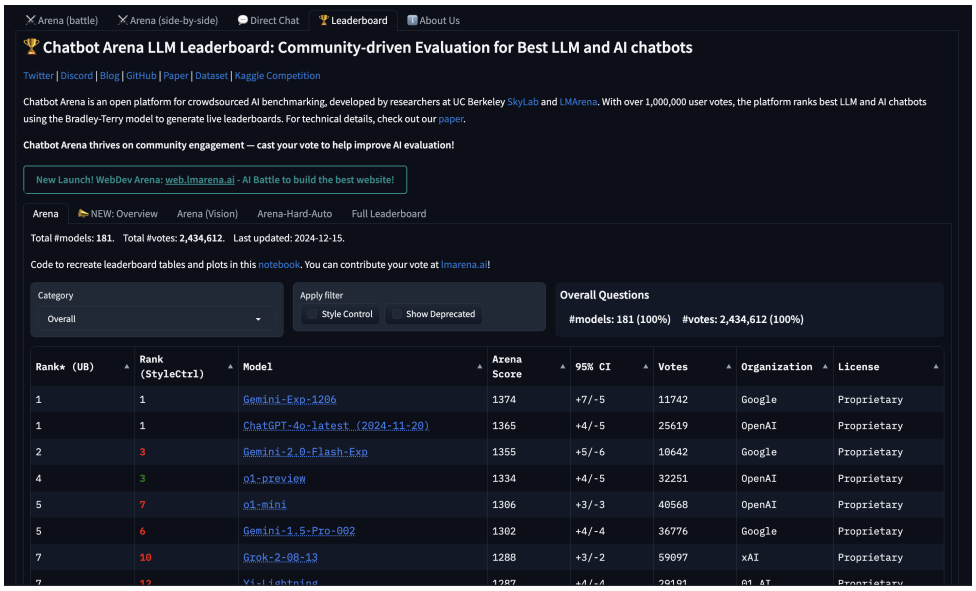

Figura 3.4: Screenshot della pagina classifica di Chatbot Arena

Un altro parametro importante nella scelta del modello è il suo costo. Infatti, più un modello è grande, e quindi spesso potente, più aumenterà la capacità computazionale

richiesta per eseguirlo. L'ideale sarebbe di poter utilizzare un large language model open-source localmente, in modo da poterlo eseguire senza dover pagare per l'utilizzo di API varie, ma questo mi limiterebbe all'utilizzo di large language model molto più piccoli, e pregiudicherebbe l'utilità dei risultati ottenuti. Ho deciso quindi di cercare di bilanciare entrambi gli aspetti, cercando di selezionare un modello che fosse abbastanza potente da poter essere utilizzato per il mio scopo, ma che non fosse troppo costoso da utilizzare. Per la valutazione del prezzo dei modelli, mi sono servito del servizio offerto da OpenRouter. Esso contiene infatti nel suo catalogo un gran numero di large language model diversi, ognuno con il proprio prezzo di utilizzo.

Come prima cosa, ho iniziato a raccogliere dati sui modelli più promettenti. Per fare ciò, sono andato sulla classifica di Chatbot Arena, ho selezionato la categoria Math e ho preso i primi 8 modelli, annotandone il nome. Ho poi raccolto, per quei modelli, i tre punteggi relativi alle categorie scelte, e ne ho calcolato la media aritmetica. Ho poi cercato i modelli su OpenRouter, e ne ho annotato il prezzo (solitamente espresso in dollari per milioni di token). Durante questo processo, ho dovuto rimuovere uno dei modelli, non essendo disponibile su OpenRouter. Questo non ha portato molta differenza nei risultati, in quanto il modello in questione era già ultimo tra i modelli scelti come capacità. Inoltre, due modelli tra quelli raccolti, o1 e o1-mini, sono modelli di tipo reasoning. Come già detto in precedenza, ho deciso di effettuare l'esperimento su un modello di questo tipo separatamente, ed ho quindi escluso anch'essi dalla scelta iniziale.

model name	Math	Instruction following	Multi-turn	avg
o1-preview	1321	1377	1405	1368
o1-mini	1340	1346	1367	1351
Gemini-Exp-1206	1286	1355	1404	1348
Gemini-2.0-Flash-Exp	1310	1347	1378	1345
ChatGPT-4o-latest (2024-11-20)	1328	1314	1324	1322
Gemini-1.5-Pro-002	1278	1301	1326	1302
Claude 3.5 Sonnet (20241022)	1281	1302	1300	1294

Figura 3.5: Confronto delle capacità dei modelli scelti

Come si può vedere dalla Figura 3.5, I due modelli che hanno ottenuto i punteggi più alti sono, infatti, o1 e o1-mini, data la loro capacità di ragionamento più elevata grazie alla tecnica di *reasoning* illustrata nel capitolo precedente. Dato che questi due modelli sono stati scartati, si trovano quindi due modelli di Google in testa, chiamati Gemini-Exp-1206 e Gemini-2.0-Flash-Exp. Seguono poi ChatGPT-4o di OpenAI, Gemini-1.5-Pro-002 sempre di Google, e Claude 3.5 Sonnet di Anthropic in fondo. È molto interessante notare come nessuno di questi modelli sia open-source. Ciò è dimostrazione dell'enorme mole di dati e computazione richiesta per allenare questi modelli, cosa che li rende spesso proibitivamente costosi da creare da zero.

Dalla Figura 3.6 nella pagina seguente, invece, si può notare qualcosa che all'inizio sembra essere un errore. Infatti, i modelli di Google più in alto in classifica sono attualmente utilizzabili gratuitamente. Si nota anche che i due modelli hanno una sigla *Exp* che segue il nome del modello. Questo indica che il modello è attualmente in fase sperimentale. Google

model name	$/1M input	$/1M output	$/1M Average
o1-preview	15	60	37.5
o1-mini	3	12	7.5
Gemini-Exp-1206	0	0	0
Gemini-2.0-Flash-Exp	0	0	0
ChatGPT-4o-latest (2024-11-20)	5	15	10
Gemini-1.5-Pro-002	1.25	5	3.125
Claude 3.5 Sonnet (20241022)	3	15	9

Figura 3.6: Confronto dei prezzi dei modelli scelti

ha infatti deciso di offrire gratuitamente l'utilizzo di questi large language model per un periodo di tempo limitato, in modo da raccogliere feedback e migliorarli. Ciò sarebbe stato molto utile per il mio esperimento, in quanto mi avrebbe permesso di utilizzare questi modelli senza dover preoccuparmi del costo di utilizzo, e con una qualità dei dati raccolti presumibilmente ottima, dato che sono tra i migliori in classifica. Questa soluzione sembrava inizialmente la migliore, ma tramite sperimentazione ho potuto constatare che i limiti di utilizzo di questi modelli sono molto stringenti. Infatti, dopo solo quattro richieste in 30 secondi, il servizio inizia a rispondere con un messaggio che indica il raggiungimento dei limiti di utilizzo. L'esperimento richiede di far interagire il modello con migliaia di posizioni diverse, e non è fattibile che queste vengano fatte limitando le richieste a 4 ogni 30 secondi, se non meno. Facendo qualche test sulla velocità di risposta di vari modelli, ho potuto constatare che i modelli di google tendono ad essere molto più veloci degli altri testati, soprattutto il modello chiamato *Gemini-1.5-Flash*, non presente nelle classifiche redatte da me. Nonostante ciò, si posiziona comunque al dodicesimo posto nella classifica generale di Chatbot Arena, e ha un prezzo estremamente basso: solo $0.075 per milione di token in input e $0.3 per milione di token in output. Ho quindi deciso di scegliere questo modello per l'esperimento, in quanto offre un buon compromesso tra prezzo e capacità, e permette di eseguire l'esperimento in modo più efficiente, riuscendo a testare centinaia di migliaia di posizioni senza dovermi preoccupare dei limiti di utilizzo. Oltre a questo modello, durante la stesura della tesi, è stato rilasciato un altro modello con prestazioni molto interessanti e dal costo contenuto, ovvero *DeepSeek V3*. Ho quindi deciso di includere anch'esso nell'esperimento per ottenere più dati e migliorare l'efficacia delle conclusioni.

3.3.3 Scelta modello *reasoning*

Visto il grande salto di qualità che molti nuovi modelli rilasciati utilizzanti la tecnica di reasoning hanno dimostrato, ho deciso di includere anche un modello di questo tipo nell'esperimento. Tuttavia, il costo di utilizzo di questi modelli è estremamente elevato, e ciò rende infattibile eseguire l'esperimento su di essi con il numero di iterazioni richiesto. Invece di rimuovere completamente questa categoria di modelli, ho deciso di includere un solo modello di questo tipo, e di eseguire l'esperimento su di esso con un numero di iterazioni ridotto. Questo mi permette di avere un'idea generale delle capacità di questi modelli, e di poterli confrontare con i modelli più tradizionali. Il modello scelto è *DeepSeek*

R1, un modello di DeepSeek che si avvicina molto alla qualità dei modelli reasoning di OpenAI, ma che ha un costo di utilizzo molto più basso, anche se comunque non trascurabile.

In conclusione, i modelli scelti per l'esperimento sono:

- **Gemini-1.5-Flash** di Google,
- **DeepSeek V3** di DeepSeek,
- **DeepSeek R1** di DeepSeek.

3.4 Struttura del codice sorgente

Ho deciso di scrivere il codice sorgente dell'esperimento in Python. Questa decisione è stata presa per molteplici motivi:

- È un linguaggio molto diffuso e ben supportato, con una vasta gamma di librerie e strumenti a disposizione.
- Permette la prototipazione molto veloce di nuove idee, grazie alla sua sintassi chiara e alla sua facilità di utilizzo.
- È uno dei linguaggi più utilizzati nel campo del *Machine Learning* e dei large language model, con librerie come *PyTorch*, e più recentemente *Transformers* di HuggingFace, che permettono di utilizzare facilmente modelli di intelligenza artificiale.
- Ultimo ma non meno importante, è un linguaggio di cui ho una buona padronanza, e che mi permette di scrivere codice in modo rapido.

In particolare, voglio concentrarmi sulle due classi principali utilizzate per rendere molti processi dell'esperimento più semplici.

3.4.1 La classe OpenRouter

La classe OpenRouter è stata creata per semplificare l'utilizzo dei large language model tramite OpenRouter. Essa gestisce l'invio di richieste al servizio, e ritorna le risposte ricevute. Per fare ciò, bisogna fornire alla classe, al momento della creazione, il nome del modello che si vuole utilizzare e la chiave API generata sul sito di OpenRouter. La classe utilizza la libreria *openai*. Questo è dovuto al fatto che, visto che OpenAI fu la prima azienda a fornire un large language model tramite API, molti dei servizi che offrono large language model hanno adottato lo stesso schema di richieste e risposte. Questo mi permette di utilizzare il modulo openai cambiando la proprietà *base_url* della classe, inserendo l'indirizzo dell'API di OpenRouter. La classe offre un singolo metodo, *generate_simple*, che prende

come parametri una stringa contenente le istruzioni, e un altro, opzionale, contenente il nome del modello da utilizzare. Il metodo ritorna una stringa contenente la risposta del modello. La classe è stata scritta in modo da essere facilmente estendibile, e da permettere di aggiungere nuovi metodi per utilizzare le funzionalità offerte da OpenRouter in modo semplice.

```python
from openrouter import OpenRouter

# Creazione dell'oggetto OpenRouter
client = OpenRouter("API_KEY")

prompt = "What is the capital of Italy?"
model = "Google/Gemini-1.5-Flash"

# Utilizzo del metodo generate_simple
response = client.generate_simple(prompt, model)

print(response)
# The capital of Italy is Rome.
```

Listing 1: Esempio di utilizzo della classe OpenRouter

3.4.2 La classe TicTacToe

La classe TicTacToe è stata creata per semplificare la creazione e la gestione di una partita a tris. Essa gestisce tutte le fasi di una partita, dalla creazione della griglia di gioco, alla gestione dei turni dei giocatori, fino ad arrivare al controllo della vittoria. La classe, inoltre, offre vari metodi complementari atti a facilitare lo svolgimento dell'esperimento, come un metodo per generare una posizione casuale valida. Per la valutazione delle mosse effettuate dai modelli ho deciso di implementare, all'interno di questa classe, un metodo che utilizza la tecnica di *minimax* per calcolare quali sono le mosse ottimali che è possibile effettuare in un certo momento di una partita. La tecnica di minimax è una tecnica di ricerca che permette di trovare la mossa migliore da effettuare in un gioco a due giocatori, in cui si cerca di massimizzare il proprio punteggio e minimizzare quello dell'avversario. È una tecnica relativamente semplice, ma data la bassa profondità dell'albero di gioco del tris, è sufficiente per trovare le mosse migliori in ogni situazione.

```python
from tictactoe import TicTacToe
from random import choice

# Creazione dell'oggetto TicTacToe
game = TicTacToe()

# continuiamo a giocare finché la partita non è finita
while not game.is_game_over():
  print(game)

  if game.current_player == "X":
      # se tocca a X, chiediamo all'utente di fare una mossa
      move_str : str = input("Enter your move (format: row,col): ")
      move : int = tuple(map(int, move_str.split(",")))
  else:
      # se tocca a O, utilizziamo il metodo minimax per trovare le mosse migliori
      # e ne scegliamo una casuale
      move = choice(game.get_best_moves())

  # effettuiamo la mossa
  game.make_move(move)

print("Game over!")
print((game.get_winner() + " wins!") if game.winner else "It's a draw!")

```

Listing 2: Esempio di una partita a tris utilizzando la classe TicTacToe

4 Risultati

In questo capitolo vengono presentati i risultati ottenuti durante la fase di sperimentazione. In particolare, vengono fatte varie ipotesi sulle prestazioni dei prompt e vengono confrontati i risultati ottenuti con quelli attesi. Infine, si cerca di dare una spiegazione ai risultati ottenuti, tenendo conto delle research question formulate nell'introduzione della tesi.

4.1 Sommario esperimento

Per ottenere i dati che verranno esposti in questo capitolo, sono stati utilizzati tre large language model diversi. Questi sono Gemini 1.5 Flash di Google, DeepSeek v3 di DeepSeek e DeepSeek R1, sempre di DeepSeek. È importante notare come, nella prima fase dell'analisi dei risultati, verranno analizzati solamente i modelli standard (senza capacità di ragionamento). i modelli standard sono stati testati con 5 esecuzioni dell'esperimento sui cinque prompt definiti in precedenza. Ogni esecuzione è stata effettuata con un numero di iterazioni pari a 1000. Il modello reasoning è stato invece testato con 3 esecuzioni, con 50 iterazioni per esecuzione. Per ogni esecuzione è stata inoltre calcolata una baseline casuale, ovvero il risultato di un modello che restituisce una mossa scelta a caso tra quelle possibili. Ciò è stato fatto per poter avere un riferimento per valutare le prestazioni dei prompt e dei modelli.

4.2 Risultati modelli standard

Le seguenti tabelle espongono i risultati degli esperimenti sui primi due modelli scelti, ovvero quelli senza capacità di ragionamento. In particolare, per ogni esecuzione e per ogni prompt si può vedere il numero di mosse ottimali effettuate dal modello su 1000 mosse totali. Inoltre, si può vedere la percentuale di mosse ottimali rispetto al totale delle mosse effettuate.

Gemini 1.5 Flash

Rank	AI name	run 1	run 2	run 3	run 4	run 5	sum	percentage
1	beginner	590	551	579	568	572	2860	57.20%
2	random (baseline)	560	525	559	539	565	2748	54.96%
3	intermediate	552	519	548	553	554	2726	54.52%
4	worse	560	519	545	530	538	2692	53.84%
5	expert	532	507	561	520	524	2644	52.88%
6	default	521	483	515	517	508	2544	50.88%

Figura 4.1: Risultati ottenuti con Gemini 1.5 Flash

DeepSeek v3

Rank	AI name	run 1	run 2	run 3	run 4	run 5	sum	percentage
1	beginner	687	680	681	710	658	3416	68.32%
2	default	669	655	663	689	655	3331	66.62%
3	intermediate	649	658	664	675	648	3294	65.88%
4	expert	655	658	662	653	633	3261	65.22%
5	worse	587	577	585	566	560	2875	57.50%
6	random (baseline)	555	585	555	564	533	2792	55.84%

Figura 4.2: Risultati ottenuti con DeepSeek v3

4.2.1 Analisi sommaria dei risultati

Come si può vedere dalle tabelle riportate, i risultati ottenuti con i primi due modelli sono molto diversi su alcuni punti, ma hanno anche delle similitudini non trascurabili. La prima cosa che è possibile notare è come DeepSeek v3 abbia ottenuto risultati nettamente migliori rispetto alla baseline casuale. Lo stesso non si può dire per Gemini 1.5 Flash, dove la baseline casuale è arrivata in seconda posizione per percentuale totale di mosse ottimali. Questo fa capire come DeepSeek v3 sia un modello molto più performante rispetto a Gemini 1.5 Flash, ed è una cosa che ci si poteva aspettare, visto che DeepSeek v3 è un modello uscito molto più di recente, ed è composto da una quantità di parametri maggiore rispetto al modello di Google. Una cosa che invece è risultata molto sorprendente è come, in entrambi i modelli e per ogni esecuzione dell'esperimento, il prompt beginner sia sempre arrivato primo. Come detto in precedenza, il prompt beginner è stato creato con l'idea di giocare ai livelli di un principiante. È quindi abbastanza inusuale il fatto che un prompt creato per giocare ai livelli di un principiante sia sempre arrivato primo, anche rispetto ai prompt intermediate e expert. Un altro risultato molto interessante è come il prompt worse, che è stato ideato per giocare nella maniera peggiore possibile, sembra riuscire a seguire il suo ruolo meglio nel modello DeepSeek v3, relativamente alle performance generali del modello. Lo stesso, però, non sembra avere il medesimo effetto nel modello di Google.

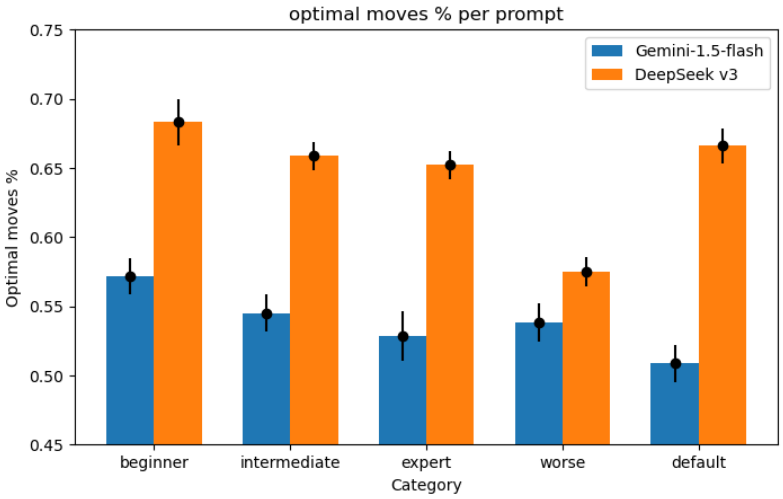

Figura 4.3: Grafico di confronto dei due modelli standard

4.3 RQ1: Analisi statistica

Con i dati sopra riportati, cercherò ora di analizzare i dati tenendo conto della prima delle tre research question che sono state formulate nell'introduzione della tesi:

> **Research Question 1 (RQ1):**
> I large language model sono in grado di variare la propria abilità di gioco semplicemente cambiando le istruzioni che gli vengono date?

Per rispondere a questa domanda, è necessario analizzare i risultati ottenuti e cercare di capire se i prompt hanno effettivamente influenzato le prestazioni dei modelli. Per fare ciò, è necessario fare un'analisi statistica dei dati ottenuti. È possibile utilizzare un test statistico chiamato ANOVA (*analysis of variances*) per confrontare i risultati ottenuti con i vari prompt. il test ANOVA è un test statistico utilizzato per determinare se ci sono differenze significative tra tre o più gruppi. In particolare, esso confronta la varianza tra i gruppi con la varianza all'interno dei gruppi. il test ANOVA richiede tre assunzioni iniziali:

- **Normalità**: gli scarti dei dati devono essere distribuiti normalmente;
- **Omogeneità della varianza**: le varianze dei gruppi devono essere uguali;
- **Indipendenza**: i dati devono essere indipendenti.

Per scarti si intende la differenza, per ogni esecuzione di ogni prompt, tra il punteggio ottenuto in quella esecuzione e la media dei punteggi ottenuti in tutte le esecuzioni di quel prompt. Per verificare queste assunzioni, ho utilizzato il test di Shapiro-Wilk per la normalità e il test di Levene per l'omogeneità della varianza, mentre ho verificato l'indipendenza dei dati tramite analisi della metodologia di raccolta degli stessi.

4.3.1 Normalità

Per verificare l'assunzione di normalità degli scarti, ho utilizzato il test di *Shapiro-Wilk* [26]. Questo test statistico viene utilizzato per determinare se un campione di dati proviene da una popolazione con distribuzione normale. Il test di Shapiro-Wilk è uno dei test più potenti per la normalità, specialmente per campioni di piccole dimensioni. Il test restituisce un valore p (detto *p-value*), che rappresenta la probabilità che i dati provengano da una distribuzione normale. Se il p-value è superiore ad un certo valore di soglia (solitamente $p = 0.05$), non si può rigettare l'ipotesi nulla che i dati provengano da una distribuzione normale (ovvero, i dati sono normalmente distribuiti). La formula del test di Shapiro-Wilk è la seguente:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.1}$$

Dove:

- $x_{(i)}$ è l'i-esimo valore più piccolo;
- $\bar{x}$ è la media dei dati;
- a_i sono i coefficienti calcolati in base alla media e alla varianza dei dati;
- n è il numero di dati;
- x_i sono i dati effettivi, nel mio caso gli scarti ottenuti dalle mosse ottimali.

Per effettuare il test, ho utilizzato la libreria python *scipy*. Il metodo *shapiro* della libreria scipy restituisce due valori: il primo è il valore W calcolato dal test, mentre il secondo è il valore p, che è il valore che mi interessa. Ho eseguito il test nel seguente modo: [H]

```
# ...experiment data definition...
from scipy.stats import shapiro

# calculate residuals
gemini_residuals = []
for category, scores in gemini_data.items():
    category_mean = np.mean(scores)
    category_residuals = [score - category_mean for score in scores]
    gemini_residuals.extend(category_residuals)

deepseek_residuals = []
for category, scores in deepseek_data.items():
    category_mean = np.mean(scores)
    category_residuals = [score - category_mean for score in scores]
    deepseek_residuals.extend(category_residuals)

# execute shapiro test
```

```
shapiro_gemini = shapiro(gemini_residuals)
shapiro_deepseek = shapiro(deepseek_residuals)
```

Eseguendo il test, si ottiene $p \approx 0.33$ per Gemini 1.5 Flash e $p \approx 0.97$ per DeepSeek v3. Entrambi i valori sono molto più alti del valore di soglia di $p = 0.05$, ed è possibile quindi assumere che i dati provengano da una distribuzione normale.

4.3.2 Omogeneità della varianza

Per verificare l'assunzione di omogeneità della varianza, ho utilizzato il test di *Levene* [27]. Il test di Levene è un test statistico che viene utilizzato per determinare se le varianze dei dati provenienti da due o più gruppi sono uguali. Ho deciso di omettere i dettagli matematici del test, e di passare direttamente ai risultati ottenuti. Anche qui, per effettuare il test, ho utilizzato scipy. In particolare, ho utilizzato il metodo *levene* nel seguente modo:

```
# ...experiment data definition...

from scipy import levene

# execute levene test
gemini_levene = levene(*gemini_data.values())
deepseek_levene = levene(*deepseek_data.values())
```

Per Gemini 1.5 Flash, il test di Levene ha restituito un valore $p \approx 0.97$, mentre per DeepSeek v3 $p \approx 0.94$. Dato che i valori sono superiori alla soglia di $p = 0.05$, si può assumere che le varianze dei dati siano uguali.

4.3.3 Indipendenza dei dati

Per la verifica dell'indipendenza, invece, non c'è bisogno di nessun test statistico. Infatti, l'indipendenza dei dati dipende esclusivamente dalla metodologia di raccolta dei dati dell'esperimento. Nel mio caso, ho effettuato cinque esecuzioni del mio esperimento, ed ogni esecuzione era formata da 1000 iterazioni, ovvero 1000 mosse effettuate dal modello su una posizione casuale del gioco del tris. Questo ci permette di affermare che i dati sono indipendenti, in quanto per ogni iterazione sono state generate posizioni casuali completamente diverse dalle altre.

4.3.4 Test ANOVA

Dopo aver verificato le assunzioni necessarie per effettuare un test ANOVA, ho proceduto con il test vero e proprio. La formula del test ANOVA è la seguente:

$$F = \frac{MS_{between}}{MS_{within}} \tag{4.2}$$

Dove:

- $MS_{between}$ è la varianza tra i gruppi;
- MS_{within} è la varianza all'interno dei gruppi.
- F è il rapporto tra le due varianze.

Sempre utilizzando la libreria scipy, ho eseguito il test ANOVA nel seguente modo:

```
# ...experiment data definition...

from scipy.stats import f_oneway

gemini_anova = f_oneway(*gemini_data.values())
deepseek_anova = f_oneway(*deepseek_data.values())
```

il metodo utilizzato calcola automaticamente anche il valore p del test. Nel caso del test ANOVA, si vuole che p sia minore della soglia di significatività $p = 0.05$, in modo da poter rigettare l'ipotesi nulla che i modelli si comportino allo stesso modo indipendentemente dal prompt.

I risultati ottenuti sono i seguenti:

- Per Gemini 1.5 Flash, si hanno $F \approx 10.28$ e $p \approx 0.0001$;
- Per DeepSeek v3, si hanno $F \approx 47.04$ e $p \approx 0.0000$.

Entrambi i valori di p sono estremamente bassi, e si può quindi affermare che i prompt hanno effettivamente influenzato le prestazioni dei modelli. In particolare, si può affermare che i prompt hanno avuto un effetto significativo e molto marcato su DeepSeek v3, mentre l'effetto è stato minore su Gemini 1.5 Flash, ma comunque significativo.

4.4 RQ2: Analisi dei prompt più interessanti

In questa sezione, terrò conto della seconda research question formulata nell'introduzione della tesi:

Research Question 2 (RQ2):
Ci sono specifici prompt che danno risultati significativamente diversi rispetto ad altri?

Procederò ad analizzare più nel dettaglio i dati raccolti, concentrandomi su prompt specifici. Verranno analizzati i risultati ottenuti con i prompt che hanno dato i risultati più interessanti, e verranno fatte delle ipotesi sul motivo delle prestazioni ottenute.

4.4.1 Analisi del prompt beginner

Il risultato più interessante ottenuto dall'esperimento è sicuramente il fatto che il prompt beginner sia sempre arrivato primo in ogni esecuzione. Per cercare di capire se questo risultato è statisticamente significativo, ho effettuato un'analisi statistica. In particolare, ho effettuato un semplice test binomiale confrontando la probabilità che il prompt beginner sia arrivato primo per caso con la probabilità che sia arrivato primo per effettiva superiorità. Un test binomiale è un test statistico utilizzato per determinare se la proporzione di successi in un esperimento è significativamente diversa dalla proporzione attesa in una distribuzione binomiale. La formula di probabilità utilizzata è la seguente:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{4.3}$$

Che è definita come la probabilità di ottenere k successi in n tentativi, con una probabilità di successo p. Nel mio caso, ho:

- $n = 5$ (numero di esecuzioni dell'esperimento);
- $k = 5$ (numero di volte che il prompt beginner è arrivato primo);
- $p = \frac{1}{5}$ (probabilità che il prompt beginner sia arrivato primo per caso, rispetto agli altri prompt).

Sostituendo i valori nella formula, ottengo:

$$\begin{aligned} & P(X = 5) \\ & = \tbinom{5}{5} \left(\tfrac{1}{5}\right)^5 \left(\tfrac{4}{5}\right)^0 \\ & = 1 \cdot \left(\tfrac{1}{5}\right)^5 \cdot 1 \\ & = \tfrac{1}{3125} \\ & = 0.00032 \end{aligned} \tag{4.4}$$

il p-value ottenuto indica quindi che la probabilità che il prompt beginner sia arrivato primo ogni volta per puro caso è dello 0.032%. Questo valore è ampiamente al di sotto di $p = 0.05$, ed è possibile quindi affermare che il prompt beginner è arrivato primo per

effettiva superiorità rispetto agli altri prompt. Si possono fare varie ipotesi sul perché di questo risultato. Una di queste è che il modello, messo davanti a istruzioni che gli indicano di giocare come un esperto, cerchi di pensare troppo alla mossa da dare e a come seguire il proprio ruolo, finendo per scegliere una mossa subottimale in più casi rispetto all'utilizzo di un prompt che invece indica di giocare come un principiante.

4.4.2 Analisi del prompt worse

Un altro risultato molto interessante ottenuto dall'esperimento è relativo al prompt worse. Questo prompt è stato creato con l'idea di giocare sempre nella maniera peggiore possibile. Analizzando i punteggi ottenuti con entrambi i large language model si può vedere come, mentre con Gemini 1.5 Flash non si notano molte differenze tra il prompt worse e i prompt vicini ad esso in classifica, le cose cambiano parecchio analizzando i punteggi di DeepSeek v3. Con quest'ultimo modello, infatti, il prompt worse ha una differenza di quasi 8 punti percentuali rispetto al prompt appena sopra di esso in classifica (expert), mentre gli altri prompt hanno delle differenze minori, di al massimo 2-3 punti percentuali. È importante ricordare come il modello di DeepSeek è più recente e più performante di quello di Google, cosa che potrebbe spiegare il perché di questa discrepanza tra i due risultati. Si può presupporre che DeepSeek v3, essendo un modello più capace in generale, sia riuscito a seguire meglio le istruzioni del prompt di giocare nella maniera peggiore possibile. Tuttavia, i dati raccolti con gli altri prompt su DeepSeek v3 non evidenziano la stessa differenza marcata che si può trovare nel prompt worse. Si può quindi presumere che il modello sia sì stato in grado di migliorare nell'aderenza alle istruzioni fornite, ma solo quando queste istruzioni sono estreme abbastanza da portare ad un cambio radicale di approccio, e comunque mai sopra le abilità assolute del large language model.

4.5 RQ3: Risultati modello reasoning

Procederò ora ad analizzare i risultati ottenuti con il modello di tipo reasoning, ovvero DeepSeek R1. In particolare, terrò conto della terza research question formulata nell'introduzione della tesi:

> **Research Question 3 (RQ3):**
> Le conclusioni ottenute nelle RQ1 e RQ2 vengono confermate anche utilizzando modelli con abilità di pensiero (*reasoning models*)?

È quindi necessario confrontare i risultati ottenuti con questo modello con quelli ottenuti con i modelli standard, e valutare quali sono le differenze e le similitudini tra di essi.

La prima cosa di cui ci si accorge guardando i risultati ottenuti con il modello di tipo reasoning è come esso abbia ottenuto risultati molto migliori rispetto ai modelli standard.

Rank	AI name	run 1	run 2	run 3	sum	percentage
1	beginner	47	44	46	137	91.33%
2	default	47	43	44	134	89.33%
3	expert	47	41	44	132	88.00%
4	intermediate	47	42	46	135	90.00%
5	random (baseline)	33	31	29	93	62.00%
6	worse	22	27	23	72	48.00%

Figura 4.4: Risultati ottenuti con il modello di tipo reasoning

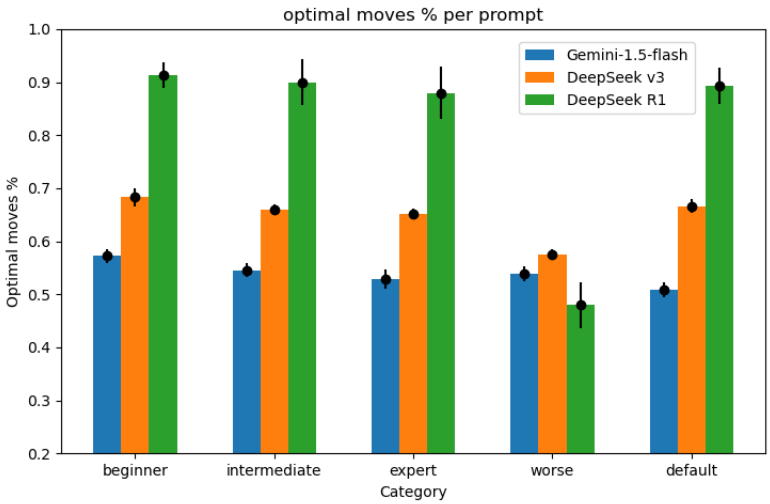

Figura 4.5: Confronto tra tutti e tre i modelli

In particolare, il modello DeepSeek R1 sembra riuscire a giocare a tris in maniera ottimale in quasi tutte le situazioni. Questo risultato dimostra il grande passo avanti che sono stati i modelli di tipo reasoning rispetto ai modelli standard. Dato il minor numero di dati raccolti con il modello di tipo reasoning, è difficile fare delle analisi statistiche sui risultati ottenuti. Tuttavia, leggendo ed interpretando i dati, si possono avanzare teorie interessanti. La prima cosa che salta all'occhio è come tutti tranne un prompt siano riusciti a trovare la mossa giusta circa il 90% delle volte. Il prompt che invece è arrivato ultimo, cioè worse, ha ottenuto il risultato più basso tra tutti i prompt testati con tutti e tre i modelli, arrivando molto al di sotto della baseline casuale. Questa osservazione conferma quanto detto in precedenza riguardo al prompt worse, dato che DeepSeek R1 sembra comportarsi allo stesso modo di DeepSeek v3, ma in maniera molto più marcata. Dai dati raccolti, però, sembra emergere anche una sorta di appiattimento delle prestazioni tra i vari prompt. Infatti, mentre con i modelli standard si potevano notare delle differenze significative tra i vari prompt, con il modello di tipo reasoning queste differenze sembrano essere molto minori. Una possibile ipotesi per questo fenomeno potrebbe essere legata al modo in cui un large language model di tipo reasoning ragiona ad alta voce. Infatti, questi modelli più avanzati sono allenati per pensare per un periodo molto maggiore rispetto a quello che si poteva ottenere con i modelli standard usando la tecnica del Chain of Thought. Ciò potrebbe portare il modello a considerare i token più vecchi in maniera meno significativa

rispetto ai token appena generati, portando durante il ragionamento a considerare sempre meno il ruolo assegnato all'inizio dell'iterazione.

5 Conclusioni e sviluppi futuri

5.1 Conclusioni

Grazie all'esperimento che ho condotto all'interno di questa tesi, sono riuscito ad analizzare il comportamento dei large language model testandone la capacità di variare il proprio livello di abilità tramite istruzioni sul livello effettivo che dovrebbero avere. I risultati ottenuti dimostrano come, entro un certo limite, questi modelli riescano a seguire il loro ruolo, ma non sempre nel modo previsto. È infatti emerso che il prompt beginner, che costituiva l'insieme di istruzioni che doveva far giocare il modello come un principiante, ha costantemente dimostrato prestazioni superiori rispetto ad altri prompt, inclusi quelli progettati per un livello di gioco più avanzato. Questo risultato suggerisce che i large language model potrebbero favorire uno stile di prompting più modesto per ottenere prestazioni migliori. L'analisi del prompt worse ha inoltre mostrato come, utilizzando un modello più capace, si può ottenere una migliore aderenza al ruolo dato come istruzioni. Tuttavia, questa migliore aderenza sembra comunque essere più evidente in negativo, come evidenziato dalla maggiore differenza tra il prompt worse e tutti gli altri nei dati raccolti con il modello DeepSeek v3. Questo potrebbe suggerire che, sebbene i modelli siano in grado di adattarsi a istruzioni di diverso livello, essi potrebbero avere una tendenza a seguire meglio le istruzioni negative rispetto a quelle positive. Infine, l'analisi dei dati raccolti con il modello DeepSeek R1 ha mostrato come i modelli di tipo reasoning sembrano essere meno sensibili alle istruzioni date, e tendono a ragionare in modo più prevedibile rispetto ai modelli standard. Analizzando i dati raccolti con questo modello, si è potuto notare come tutti i prompt tranne il prompt worse abbiano ottenuto risultati molto simili tra loro. È stato proprio quest'ultimo prompt, però, a dimostrare una grossa differenza rispetto agli altri, con un punteggio medio di vittorie molto più basso di tutti gli altri prompt su tutti i modelli testati.

5.2 Research answer

5.2.1 RQ1 - I large language model sono in grado di variare la propria abilità di gioco semplicemente cambiando le istruzioni che gli vengono date?

Sì, i large language model sono in grado di variare il proprio livello di abilità in base alle istruzioni date, ma non sempre nel modo previsto. Si è infatti notato come prompt che chiedevano al modello un livello di abilità più basso abbiano ottenuto risultati migliori rispetto a prompt che chiedevano un livello di abilità più alto. Questo suggerisce che i large language model potrebbero favorire uno stile di prompting più modesto per ottenere prestazioni migliori.

5.2.2 RQ2 - Ci sono specifici prompt che danno risultati significativamente diversi rispetto ad altri?

I risultati ottenuti mostrano come il prompt beginner abbia ottenuto risultati migliori rispetto a tutti gli altri prompt, arrivando in prima posizione in tutte e 5 le esecuzioni effettuate con entrambi i modelli standard. Inoltre, il prompt worse, che chiedeva di giocare nella maniera peggiore possibile, si è rivelato più efficace con il modello DeepSeek v3 rispetto al modello Gemini 1.5 Flash, dimostrando come i modelli più capaci possano adattarsi meglio alle istruzioni date. Tuttavia, questo effetto sembra essere più marcato in negativo, data la minore differenza tra gli altri prompt nel modello DeepSeek v3.

5.2.3 RQ3 - Le conclusioni ottenute nelle RQ1 e RQ2 vengono confermate anche utilizzando modelli con abilità di pensiero (*reasoning models*)?

Analizzando i dati raccolti dal modello DeepSeek R1, si può vedere come i modelli di tipo reasoning sembrano essere meno sensibili al livello di abilità richiesto, e tendono a ragionare in modo più prevedibile rispetto ai modelli standard. Infatti, tutti i prompt tranne il prompt worse hanno ottenuto risultati molto simili tra loro. L'unico prompt che ha dimostrato una differenza significativa rispetto agli altri è stato proprio il prompt worse, che ha ottenuto un punteggio medio di vittorie molto più basso rispetto a tutti gli altri prompt su tutti i modelli testati.

5.3 Limitazioni

È importante riconoscere alcuni limiti della metodologia utilizzata. In primo luogo, l'esperimento è stato condotto su un gioco relativamente semplice come il tris, e i risultati potrebbero non essere generalizzabili a compiti più complessi. Inoltre, sono stati testati solo due modelli tradizionali e un modello di tipo reasoning, che rappresentano solo una piccola parte dell'ecosistema attuale dei large language model. Bisogna anche far presente come i large language model possono essere molto suscettibili a come le istruzioni vengono scritte. Il modo in cui ho formulato i prompt nella mia tesi potrebbe non essere quello migliore, e altri formati di istruzioni potrebbero dare risultati diversi. Ciò nonostante, ho cercato di essere il più rigoroso possibile nella metodologia e nell'analisi dei dati.

5.4 Sviluppi futuri

Questo studio apre diverse direzioni per ricerche future. Sarebbe interessante estendere l'analisi a giochi più complessi, come per esempio gli scacchi, per capire se i risultati ottenuti vengono mantenuti. Si potrebbe inoltre pensare di testare i modelli su altri tipi di compiti, come la traduzione o la generazione di testo, per capire se i risultati ottenuti sono specifici del gioco del tris o se possono essere generalizzati ad altri compiti, e se i modelli di tipo reasoning si comportano in modo simile. Infine, sarebbe interessante esplorare nuovi formati di istruzioni per capire se i risultati ottenuti sono sensibili al modo in cui le istruzioni vengono scritte. Un'ultima direzione più pratica potrebbe essere quella di testare un numero maggiore di modelli, per capire se i risultati ottenuti sono specifici dei modelli testati o se possono essere generalizzati a un numero maggiore di modelli.

5.5 Costi

L'esecuzione dei vari esperimenti ha comportato dei costi, che ho dovuto sostenere. In particolare, per l'utilizzo dei vari modelli testati attraverso OpenRouter, ho utilizzato un numero di crediti pari al valore di circa €20. È interessante notare come, pur avendo eseguito solamente 750 chiamate circa a DeepSeek R1 contro le decine di migliaia effettuate per gli altri LLM, esso ha utilizzato circa la metà dei crediti totali. Ciò dimostra come i modelli di tipo *reasoning* siano molto più costosi e quindi meno accessibili rispetto ai modelli standard. Questo è un aspetto che rende difficile l'utilizzo di questi modelli in ambito di ricerca, e che va tenuto in considerazione per eventuali sviluppi futuri.

Bibliografia

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser e I. Polosukhin, «Attention Is All You Need», *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. indirizzo: http://arxiv.org/abs/1706.03762.

[2] A. Radford, K. Narasimhan, T. Salimans e I. Sutskever, «Improving language understanding by generative pre-training», 2018.

[3] G. Kohs, *AlphaGo*, Documentary film, Visionabile su YouTube. Accessed: 2024-12-14, 2017. indirizzo: https://www.youtube.com/watch?v=WXuK6gekU1Y.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., «Mastering the game of Go with deep neural networks and tree search», *nature*, vol. 529, n. 7587, pp. 484–489, 2016.

[5] S. Byford, «Google's AlphaGo AI beats Lee Se-dol again to win Go series 4-1», *The Verge*, mar. 2016, Accessed: 2024-12-12. indirizzo: https://www.theverge.com/2016/3/15/11213518/alphago-deepmind-go-match-5-result.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., *Mastering the game of go without human knowledge*, 2017.

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan e D. Hassabis, «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play», *Science*, vol. 362, n. 6419, pp. 1140–1144, 2018. doi: 10.1126/science.aar6404. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404. indirizzo: https://www.science.org/doi/abs/10.1126/science.aar6404.

[8] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel et al., «Mastering atari, go, chess and shogi by planning with a learned model», *Nature*, vol. 588, n. 7839, pp. 604–609, 2020.

[9] M. G. Bellemare, Y. Naddaf, J. Veness e M. Bowling, «The arcade learning environment: An evaluation platform for general agents», *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[10] Stockfish Team, *Stockfish: Strong open-source chess engine*, Accessed: 2024-12-13, 2024. indirizzo: `https://stockfishchess.org/`.

[11] *TCEC - Top Chess Engine Championship*, Accessed: 2024-12-14, 2024. indirizzo: `https://tcec-chess.com`.

[12] *TCEC Season 26 Superfinal Results*, Accessed: 2024-12-14, 2024. indirizzo: `https://www.chess.com/events/2024-tcec-26-superfinal/results`.

[13] OpenAI, *OpenAI models tokenizer*, Utilizzato tokenizer per GPT-4o, Accessed: 2024-12-14, 2024. indirizzo: `https://platform.openai.com/tokenizer`.

[14] A. Pagnoni, R. Pasunuru, P. Rodriguez, J. Nguyen, B. Muller, M. Li, C. Zhou, L. Yu, J. Weston, L. Zettlemoyer, G. Ghosh, M. Lewis, A. Holtzman† e S. Iyer, «Byte Latent Transformer: Patches Scale Better Than Tokens», 2024. indirizzo: `https://github.com/facebookresearch/blt`.

[15] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever e D. Amodei, *Language Models are Few-Shot Learners*, 2020. arXiv: `2005.14165 [cs.CL]`. indirizzo: `https://arxiv.org/abs/2005.14165`.

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le e D. Zhou, *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, 2023. arXiv: `2201.11903 [cs.CL]`. indirizzo: `https://arxiv.org/abs/2201.11903`.

[17] OpenAI Team, *Introducing OpenAI o1-preview*, Accessed: 2025-03-02, 2024. indirizzo: `https://openai.com/index/introducing-openai-o1-preview/`.

[18] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan et al., *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*, 2025. arXiv: `2501.12948 [cs.CL]`. indirizzo: `https://arxiv.org/abs/2501.12948`.

[19] S. Schaefer, «Tic-Tac-Toe (Naughts and Crosses, Cheese and Crackers, etc.) (January 2002)», 2002, Accessed: 2025-03-09, Link to archived page on The Wayback Machine. indirizzo: https://web.archive.org/web/20130628112339/http://www.mathrec.org/old/2002jan/solutions.html.

[20] J. Tromp, «Chess Position Ranking», 2020, Accessed: 2025-03-09. indirizzo: https://github.com/tromp/ChessPositionRanking.

[21] O. Enthusiast, *Tic-Tac-Toe Analysis using Clojure - Part 1*, Accessed: 2025-02-04, 2015. indirizzo: https://www.occasionalenthusiast.com/tic-tac-toe-analysis-using-clojure-part-1/.

[22] OpenRouter Team, *OpenRouter*, Accessed: 2024-12-18, 2024. indirizzo: https://openrouter.ai/.

[23] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. Jordan, J. E. Gonzalez e I. Stoica, *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*, 2024. arXiv: 2403.04132 [cs.AI]. indirizzo: https://arxiv.org/abs/2403.04132.

[24] J. Liu, L. Cui, H. Liu, D. Huang, Y. Wang e Y. Zhang, «Logiqa: A challenge dataset for machine reading comprehension with logical reasoning», *arXiv preprint arXiv:2007.08124*, 2020.

[25] J. Golde, P. Haller, F. Barth e A. Akbik, «MastermindEval: A Simple But Scalable Reasoning Benchmark», in *Workshop on Reasoning and Planning for Large Language Models*, 2025. indirizzo: https://openreview.net/forum?id=H4donosutm.

[26] S. S. Shapiro e M. B. Wilk, «An analysis of variance test for normality (complete samples)», *Biometrika*, vol. 52, n. 3-4, pp. 591–611, 1965.

[27] H. Levene, «Robust tests for equality of variances», *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, pp. 278–292, 1960.

[28] PyTorch Team, *PyTorch*, Accessed: 2024-12-18, 2024. indirizzo: https://pytorch.org/.

[29] Hugging Face Team, *Transformers*, Accessed: 2024-12-18, 2024. indirizzo: https://huggingface.co/transformers/.

[30] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo e Y. Iwasawa, *Large Language Models are Zero-Shot Reasoners*, 2023. arXiv: 2205.11916 [cs.CL]. indirizzo: https://arxiv.org/abs/2205.11916.

Ringraziamenti

Vorrei ringraziare prima di tutto il mio relatore, il Professor Paolo Ciancarini, per avermi seguito nella stesura di questa tesi, e per avermi spronato a migliorarla sempre di più.

Ringrazio anche i miei genitori e tutta la mia famiglia, per avermi sempre sostenuto e per avermi dato la possibilità di studiare ciò che più mi piace.

Un ringraziamento speciale va a tutti i miei amici, sia online che non, che mi hanno sempre sostenuto e aiutato nei momenti di difficoltà.

Infine, ringrazio tutti i professori e i colleghi che ho incontrato durante il mio percorso di studi. Il ricordo di questi anni rimarrà sempre con me.

Senza tutte queste persone, non sarei riuscito a portare a termine questo percorso di studi.