VIP Cheatsheet: Trasformatori e Modelli linguistici di grandi dimensioni

Afshine Amidi e Shervine Amidi
Tradotto da Gianluca Guzzetta

26 aprile 2025

Questo VIP cheatsheet fornisce una panoramica di ciò che è contenuto nel libro "Super Study Guide: Trasformatori e Modelli linguistici di grandi dimensioni", che contiene ∼600 illustrazioni in 250 pagine e approfondisce i seguenti concetti.
Ulteriori dettagli sono disponibili sul sito `https://superstudy.guide`.

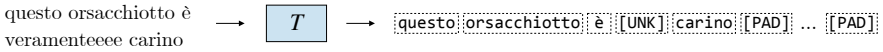
1 Le basi

1.1 I Token

❒ **Definizione** – Un *token* è un'unità di testo indivisibile, ovvero una parola, una sotto-parola o un carattere, e fa parte di un vocabolario predefinito.

Osservazione: Il token ignoto o sconosciuto (unknown, `[UNK]`*) rappresenta componenti ignote di testo, mentre il token di riempimento (padding,* `[PAD]`*) è utilizzato per riempire le posizioni vuote per garantire la coerenza delle lunghezze delle sequenze in input o in ingresso.*

❒ **Tokenizzatore** – Un *tokenizzatore* T suddivide il testo in token con un arbitrario livello di granularità.



Qui sotto sono riportati i principali tipi di tokenizzatori:

Tipologia	Vantaggio	Svantaggio	Figura
Parola	• Facile da interpretare • Sequenza breve	• Vocabolario di grandi dimensioni • Le variazioni delle parole non sono gestite	`orsacchiotto`
Sotto-parola	• Influenzato dalle radici delle parole • Embeddings intuitivi	• Lunghezza della sequenza • La tokenizzazione è più complessa	`orsac` `##chiotto`
Carattere Byte	• Nessun problema di elementi al di fuori del dizionario • Vocabolario di piccole dimensioni	• Una lunghezza della sequenza molto più vasta • Modelli difficili da interpretare perché di livello troppo basso	`o r s a c c h i o t t o`

Osservazione: Byte-Pair Encoding (BPE) e Unigram sono tokenizzatori a livello di sotto-parola comunemente utilizzati.

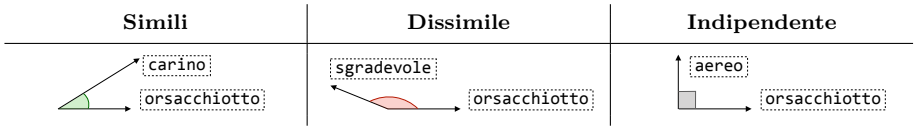
1.2 Embeddings

❒ **Definizione** – Un *embedding* è una rappresentazione numerica di un elemento (e.s. token, frase) ed è caratterizzata da un vettore $x \in \mathbb{R}^n$.

❒ **Similarità** – La *similitudine del coseno* o similarità del coseno tra due token t_1, t_2 è quantificata da:

$$\text{similarità}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||} = \cos(\theta) \in [-1, 1]$$

L'angolo θ caratterizza la similarità tra i due token:

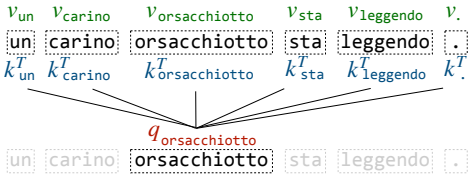


Osservazione: Approximate Nearest Neighbors (ANN) e Locality Sensitive Hashing (LSH) sono metodi che approssimano l'operazione di similarità in modo efficiente su database di grandi dimensioni.

2 Trasformatori

2.1 Attenzione

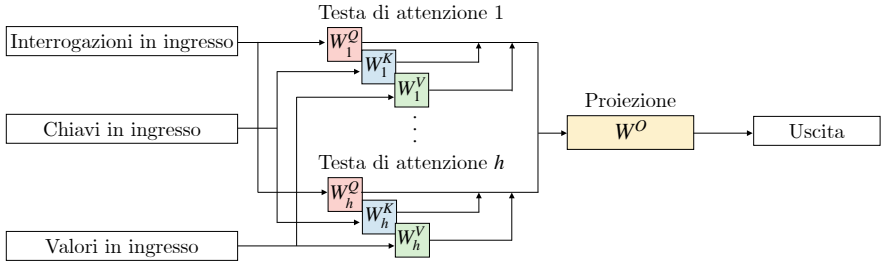
❒ **Formula** – Data un'*interrogazione* q (*query*), vogliamo conoscere quale *chiave* k (*key*) l'interrogazione q dovrebbe prestare attenzione rispetto al *valore* associato v (*value*).



L'attenzione può essere calcolata in modo efficace usando le matrici Q, K, V che contengono rispettivamente le interrogazioni q, le chiavi k ed i valori v, lungo la dimensione d_k delle chiavi:

$$\text{attenzione} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

❒ **MHA** – Uno *strato di attenzione multi-testa* (***M**ulti-**H**ead **A**ttention*, MHA) esegue i calcoli computazionali dell'attenzione attraverso molteplici "teste", successivamente proietta il risultato nello spazio dell'output o d'uscita.

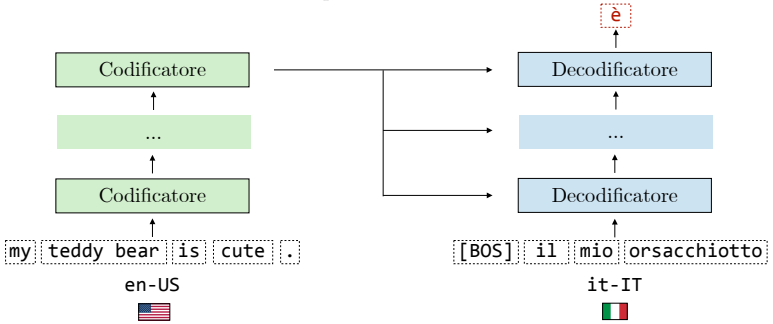


È composta da teste di attenzione h, così come le matrici W^Q, W^K, W^V, che proiettano l'input per ottenere le interrogazioni Q, le chiavi K ed i valoi V. La proiezione è trovata usando la matrice W^O.

Osservazione: Grouped-Query Attention (GQA) e Multi-Query Attention (MQA) sono varianti di MHA che riducono l'overhead computazionale condividendo chiavi e valori tra le teste di attenzione.

2.2 Architettura

❒ **Panoramica** – Il *trasformatore* (*Transformer*) è un modello di riferimento che si basa sul meccanismo dell'auto-attenzione ed è composto da codificatori e decodificatori. I codificatori calcolano le incorporazioni significative dell'input, che vengono poi utilizzate dai decodificatori per prevedere il token successivo nella sequenza.



Osservazione: Sebbene il trasformatore fu inizialmente proposto come modello per compiti di traduzione, è ad oggi molto utilizzato in numerose applicazioni.

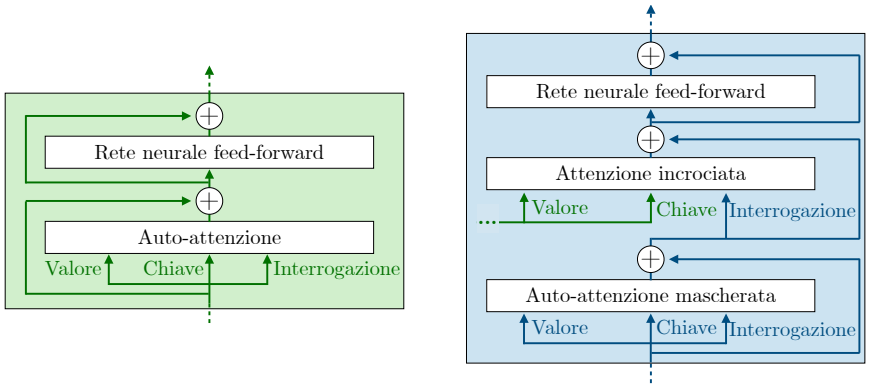
❒ **Componenti** – Il *codificatore* ed il *decodificatore* sono due componenti fondamentali dei trasformatore e hanno dei diversi ruoli:

Codificatore

Gli embeddings codificati racchiudono il significato dell'input.

Decodificatore

Gli embeddings decodificati contengono il significato sia dell'input sia dell'output predetto sino a qual momento.

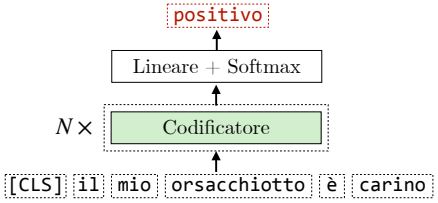


❒ **Embeddings di posizione** – Gli *embeddings di posizione* informano sulla posizione del token nella frase e hanno la stessa dimensione degli embeddings di token. Possono essere definiti arbitrariamente o appresi dai dati.

Osservazione: I RoPE (Rotary Position Embeddings) sono una variante popolare ed efficiente che ruota i vettori delle query e delle chiavi per incorporare informazioni sulla posizione relativa.

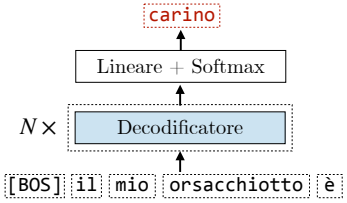
2.3 Varianti

❒ **Solo codificatore** – ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* (BERT) è un modello basato su trasformatori, composto da una pila di codificatori che prende in input un testo e produce embeddings significativi, che possono essere utilizzati in seguito in compiti di classificazione a valle.



Un token `[CLS]` viene aggiunto all'inizio della sequenza per catturare il significato della frase. Il suo embedding codificato viene spesso utilizzato in attività successive, come l'estrazione del sentiment.

❒ **Solo decodificatore** – Il ***G**enerative **P**re-trained **T**ransformer* (GPT) è un modello basato su un trasformatore autoregressivo composto da una pila di decodificatori. A differenza di BERT e dei suoi derivati, GPT tratta tutti i problemi come problemi testo-testo.



La maggior parte degli attuali LLM si basa su un'architettura di solo decodifica, come la serie GPT, LLaMA, Mistral, Gemma, DeepSeek, etc.

Osservazione: I modelli codificatore-decodificatore, come il T5, sono anch'essi autoregressivi e condividono molte caratteristiche con i modelli di solo decodifica.

2.4 Ottimizzazioni

❒ **Approssimazione dell'attenzione** – I calcoli dell'attenzione sono in $\mathcal{O}(n^2)$, il che può essere costoso all'aumentare della lunghezza della sequenza n. Esistono due metodi principali per approssimare i calcoli:

- *Sparsità*: L'auto-attenzione non avviene per tutta la sequenza, ma solo tra i token più rilevanti.

- *Low-rank*: La formula dell'attenzione viene semplificata come prodotto di matrici di basso rango, riducendo così il carico di calcolo.

❒ **Attenzione flash** – L'*attenzione flash* (*Flash attention*) è un metodo esatto che ottimizza i calcoli dell'attenzione sfruttando abilmente l'hardware delle GPU, utilizzando la veloce memoria statica ad accesso casuale (***S**tatic **R**andom-**A**ccess **M**emory*, SRAM) per le operazioni sulle matrici prima di scrivere i risultati sulla più lenta memoria ad alta larghezza di banda (***H**igh **B**andwidth **M**emory*, HBM).

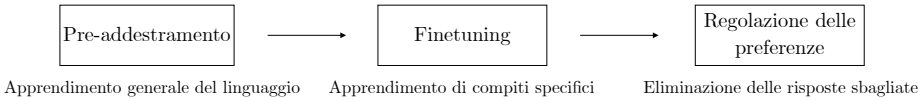
Osservazione: In pratica, questo riduce l'uso della memoria e velocizza i calcoli.

3 Modelli di linguaggio di grandi dimensioni

3.1 Panoramica

❒ **Definizione** – Un *modello di linguaggio di grandi dimensioni* (***L**arge **L**anguage **M**odel*, LLM) è un modello basato su un trasformatore con forti capacità NLP. È "grande" nel senso che contiene tipicamente miliardi di parametri.

❒ **Ciclo di vita** – Un LLM viene addestrato in 3 fasi: pre-addestramento, messa a punto e messa a punto delle preferenze.

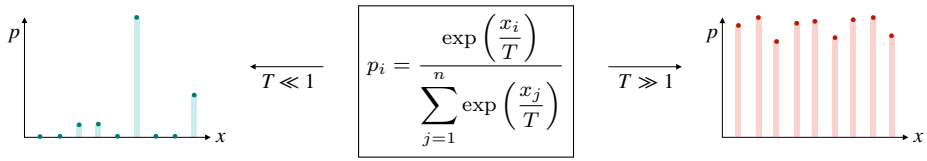


Il finetuning e il preference tuning sono approcci post-training che mirano ad *allineare il modello* per eseguire determinati compiti.

3.2 Prompting

❒ **Lunghezza di contesto** – La *lunghezza di contesto* di un modello è il numero massimo di tokens che possono essere inseriti all'input o all'ingresso. Tipicamente spazia da una decina di migliaia di token sino a milioni di tokens.

❒ **Campionamento della decodifica** – Le previsioni dei token sono campionati dalla distribuzione di probabilità predetta p_i, che è controllata dall'iperparametro di temperatura T.



Osservazione: Una alta temperatura porta ad output più 'creativi', mentre una bassa temperatura comporta risultati più deterministici.

❒ **Catena di pensiero** – La *catena di pensiero* (***C**hain-**o**f-**T**hought*, CoT) è un processo di ragionamento in cui il modello suddivide un problema più complesso in una serie di passi intermedi. Tutto ciò aiuta il modello a generare una risposta finale corretta. L'albero di pensieri (***T**ree **o**f **T**houghts*, ToT) è una versione più avanzata della CoT.

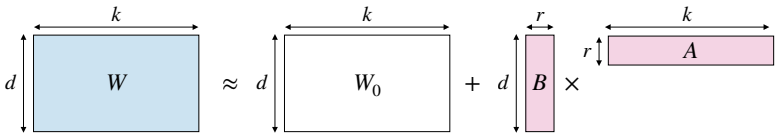
Osservazione: L'auto consistenza è un metodo che aggrega le risposte tra i vari percorsi di ragionamento della CoT.

3.3 Finetuning

❒ **SFT** – Il *finetuning supervisionato* (***S**upervised **F**ine**T**uning*, SFT) è un approccio post-addestramento che allinea il comportamento del modello al suo scopo finale. Si basa su coppie ingresso-uscita (input-output) di alta qualità allineate al compito.

Osservazione: Se i dati del SFT riguardano delle istruzioni, questo step viene definito come 'instruction tuning', o perfezionamento dell'istruzione.

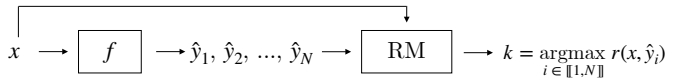
❒ **PEFT** – Il ***P**arameter-**E**fficient **F**ine**T**uning* (PEFT) è una categoria di metodi utilizzati per eseguire SFT in modo efficiente. In particolare, il ***Lo**w-**R**ank **A**daptation* (LoRA) approssima i pesi apprendibili W fissando W_0 e apprendendo invece le matrici di basso rango A e B:



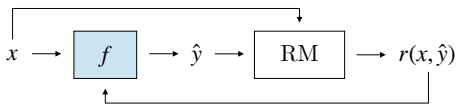
Osservazione: Altre tecniche PEFT includono la sintonizzazione del prefisso e l'inserimento del livello adattatore.

3.4 Tuning delle preferenze

❒ **Modello di ricompensa** – Un *modello di ricompensa* (***R**eward **M**odel*, RM) è un modello che prevede quanto bene un output $\hat{y}$ si allinei con il comportamento desiderato dato un certo input x. Il campionamento al meglio di N (***B**est-**o**f-**N***, BoN), definito anche *campionamento di rigetto* (*rejection sampling*) è un metodo che utilizza un modello di ricompensa per selezionare la risposta migliorate tra le N risposte generate.



❒ **Apprendimento per rinforzo** – L'*apprendimento per rinforzo* (***R**einforcement **L**earning*, RL) è un approccio che sfrutta il modello di ricompensa RM e aggiorna il modello f in base alle ricompense per gli output generati. Se il RM è basato sulle preferenze umane, questo processo è chiamato ***R**einforcement **L**earning from **H**uman **F**eedback* (RLHF).

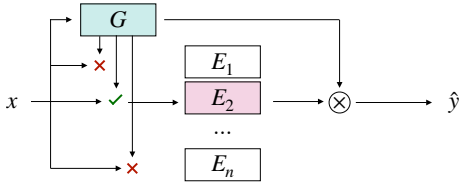


L'*ottimizzazione della politica prossimale* (***P**roximal **P**olicy **O**ptimization*, PPO) è un algoritmo di RL molto diffuso che incentiva ricompense più elevate, mantenendo il modello vicino a quello di base per evitare l'hacking delle ricompense mantenendo il modello vicino al modello di base per evitare l'hacking delle ricompense.

Osservazione: Esistono anche approcci supervisionati, come la Direct Preference Optimization (DPO), che combinano RM e RL in un'unica fase supervisionata.

3.5 Ottimizzazioni

❒ **Miscela di esperti** – Una *miscela di esperti* (***M**ixture **o**f **E**xperts*, MoE) è un modello che attiva solo una parte dei suoi neuroni al momento dell'inferenza. Si basa su un gate G e sugli esperti $E_1, ..., E_n$.



$$\hat{y} = \sum_{i=1}^{n} G(x)_i E_i(x)$$

I LLM basati su MoE utilizzano questo meccanismo di gating nelle loro FFNN.

Osservazione: L'addestramento di un LLM basato su MoE è notoriamente impegnativo, come indicato nell'articolo su LLaMA, i cui autori hanno scelto di non utilizzare questa architettura nonostante la sua efficienza in termini di tempo di inferenza.

❒ **Distillazione** – La *distillazione* è un processo in cui un modello studente (piccolo) S viene addestrato sui risultati di previsione di un modello insegnante (grande) T. Viene addestrato utilizzando la perdita di divergenza KL:

$$\text{KL}(\hat{y}_T || \hat{y}_S) = \sum_i \hat{y}_T^{(i)} \log\left(\frac{\hat{y}_T^{(i)}}{\hat{y}_S^{(i)}}\right)$$

Osservazione: Le etichette di addestramento sono considerate etichette "morbide", poiché rappresentano le probabilità delle classi.

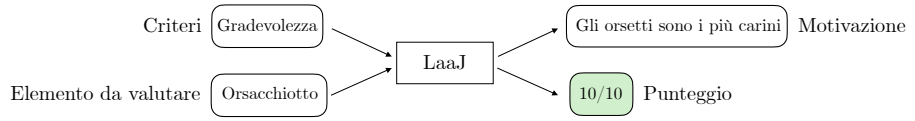
❒ **Quantizzazione** – La *quantizzazione del modello* è una categoria di tecniche che riduce la precisione dei pesi del modello limitandone l'impatto sulle prestazioni del modello stesso. Di conseguenza, riduce l'ingombro in memoria del modello e ne accelera l'inferenza.

Osservazione: QLoRA è una variante quantizzata di LoRA comunemente utilizzata.

4 Applicazioni

4.1 LLM come giudice

❒ **Definizione** – Il *LLM come giudice* (***LLM-as-a-Judge***, LaaJ) è un metodo che utilizza un LLM per assegnare un punteggio a determinati output in base ad alcuni criteri forniti. In particolare, è anche in grado di generare una logica per il suo punteggio, che aiuta l'interpretabilità.



Contrariamente alle metriche dell'era pre-LLM, come ***R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation* (ROUGE), LaaJ non ha bisogno di alcun testo di riferimento, il che la rende comoda da valutare su qualsiasi tipo di compito. In particolare, LaaJ mostra una forte correlazione con le valutazioni umane quando si basa su un modello grande e potente (ad esempio GPT-4), poiché richiede capacità di ragionamento per ottenere buone prestazioni.

Osservazione: LaaJ è utile per eseguire rapidi cicli di valutazione, ma è importante monitorare l'allineamento tra i risultati di LaaJ e le valutazioni umane per assicurarsi che non vi siano divergenze.

❒ **Distorsioni comuni** – I modelli LaaJ possono presentare le seguenti distorsioni o bias:

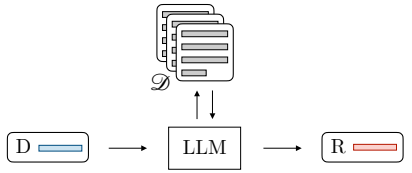
	Posizione distorta	**Verbosità distorta**	**Distorsione da auto-valorizzazione**
Problema	Favorisce la prima posizione nei confronti a coppie	Preferisce contenuti più verbosi	Favorisce gli output generati da se stessi
Soluzione	Metrica media su posizioni randomizzate	Aggiungere una penalità sulla lunghezza dell'output	Utilizzare un giudice costruito a partire da un modello di base diverso

Un rimedio a questi problemi può essere la messa a punto di un LaaJ personalizzato, ma ciò richiede un notevole sforzo.

Osservazione: L'elenco di pregiudizi sopra riportato non è esaustivo.

4.2 RAG

❒ **Definizione** – La ***R**etrieval-**A**ugmented **G**eneration* (RAG) è un metodo che consente al LLM di accedere a conoscenze esterne rilevanti per rispondere a una determinata domanda. Questo metodo è particolarmente utile se si vogliono incorporare informazioni che superano la data limite della conoscenza preaddestrata del LLM.



Data una base di conoscenza $\mathscr{D}$ e una domanda, un **R**etriever recupera i documenti più rilevanti, quindi **A**rricchisce la domanda con le informazioni pertinenti prima di **G**enerare l'output.

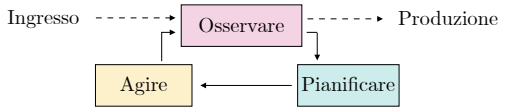
Osservazione: La fase di recupero si basa tipicamente sulle incorporazioni dei modelli di solo codificatore.

❒ **Iperparametri** – La base di conoscenza $\mathscr{D}$ viene inizializzata suddividendo i documenti in pezzi di dimensione n_c e incorporandoli in vettori di dimensione $\mathbb{R}^d$.

4.3 Agenti

❒ **Definizione** – Un *agente* è un sistema che persegue autonomamente degli obiettivi e completa dei compiti per conto di un utente. Per farlo può utilizzare diverse catene di chiamate LLM.

❒ **ReAct** – ***Reason*** + ***Act*** (ReAct) è un framework che consente catene multiple di chiamate LLM per completare compiti complessi:



Questo framework è composto dai passi seguenti:

- *Osservare*: Sintetizzare le azioni precedenti e dichiarare esplicitamente ciò che è attualmente noto.
- *Pianificare*: Dettagliare i compiti da svolgere e gli strumenti da utilizzare.
- *Agire*: Eseguire un'azione tramite un'API o cercare informazioni pertinenti in una base di conoscenza.

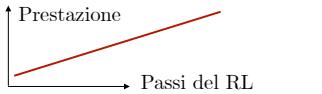
Osservazione: La valutazione di un sistema agenziale è impegnativa. Tuttavia, è possibile farlo sia a livello di componente, tramite gli input-output locali, sia a livello di sistema, tramite catene di chiamate.

4.4 Modelli di ragionamento

❒ **Definizione** – Un *modello di ragionamento* è un modello che si basa su tracce di ragionamento basate sulla CoT per risolvere compiti più complessi di matematica, codifica e logica. Esempi di modelli di ragionamento sono la serie o di OpenAI, DeepSeek-R1 e Gemini Flash Thinking di Google.

Osservazione: DeepSeek-R1 fornisce esplicitamente la sua traccia di ragionamento tra i tag `<think>`.

❒ **Ridimensionamento** – Per migliorare le capacità di ragionamento si utilizzano due tipi di metodi di scalatura:

	Descrizione	**Figura**
Ridimensionamento in tempo di addestramento	Eseguire l'RL più a lungo per consentire al modello di imparare a produrre tracce di ragionamento in stile CoT prima di dare una risposta.	
Ridimensionamento del tempo di test	Lasciate che il modello rifletta più a lungo prima di fornire una risposta con parole chiave di forzatura del budget come "`Aspetta`".