

# Large Language Models

CORSO DI NATURAL LANGUAGE PROCESSING (ELABORAZIONE DEL LINGUAGGIO NATURALE)

a.a. 2025/2026

Prof. Roberto Pirrone

# Sommario

- 📌 Large Language Models
- 📌 Next token prediction
- 📌 Autoregressione
- 📌 NLP Tasks con gli LLM
- 📌 Decoding
- 📌 Sampling
- 📌 Addestramento
- 📌 Algoritmo

- 📌 Dati di addestramento
- 📌 Finetuning
- 📌 Valutazione
- 📌 Scalabilità
- 📌 Scaling Laws
- 📌 KV Cache
- 📌 PEFT
- 📌 LoRA
- 📌 Quantizzazione



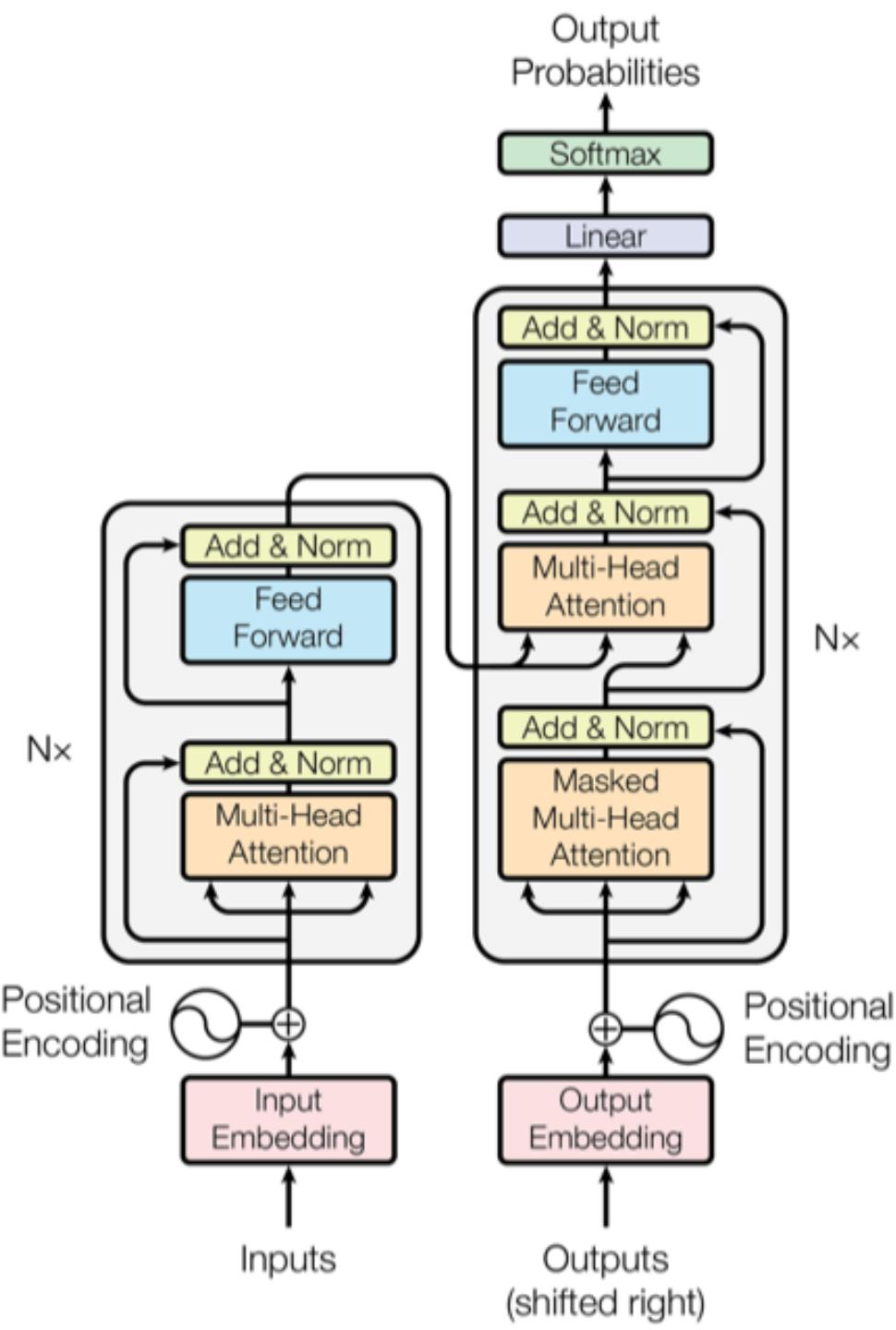
<https://web.stanford.edu/~jurafsky/slp3/>

# Large Language Models

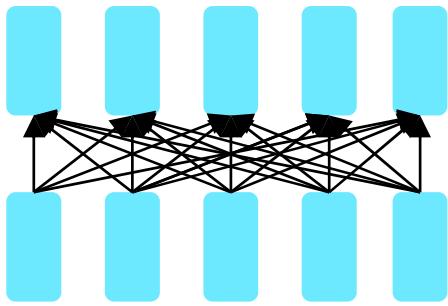
- Ricordiamo il semplice modello linguistico a N-grammi
  - ✓ Assegna probabilità a sequenze di parole
  - ✓ Genera testo campionando le possibili parole successive
  - ✗ È addestrato su conteggi calcolati da a partire da grandi quantità di testo
- I **Large Language Models (LLM)** sono simili e diversi:
  - ✓ Assegna probabilità a sequenze di parole
  - ✓ Genera testo campionando le possibili parole successive
  - 💡 Sono addestrati imparando a indovinare la parola successiva

# Large Language Models

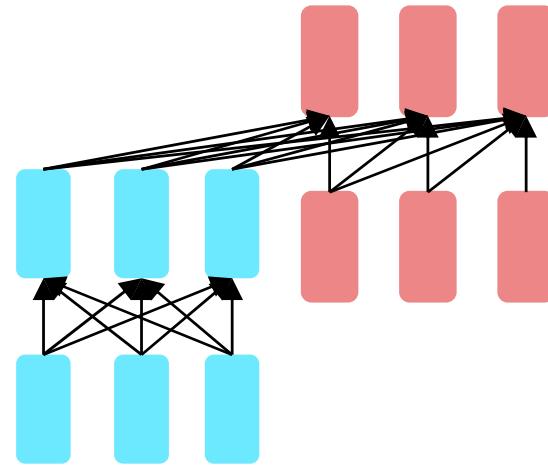
- L'architettura di base è sempre la stessa, il transformer è composto da un encoder e un decoder
- A seconda dei modelli, cambia come le componenti encoder-decoder interagiscono fra di loro e il metodo di attenzione utilizzato



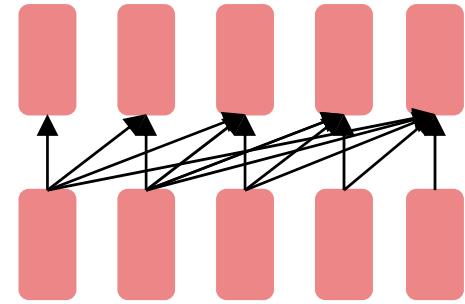
# Large Language Models



**Encoders**  
BERT family

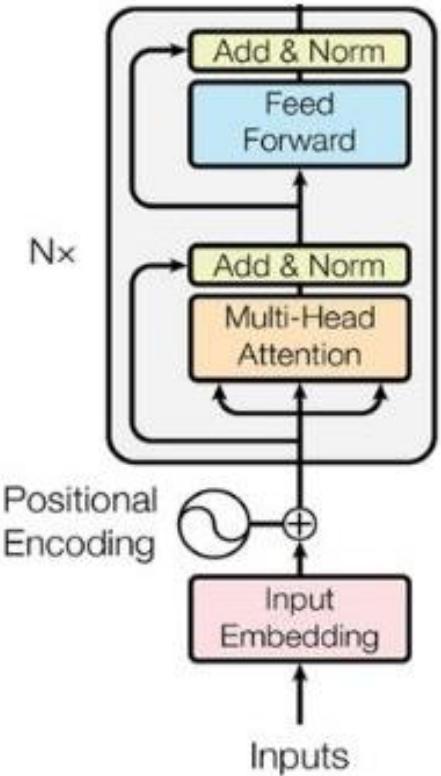


**Encoder-decoders**  
Flan-T5, Whisper

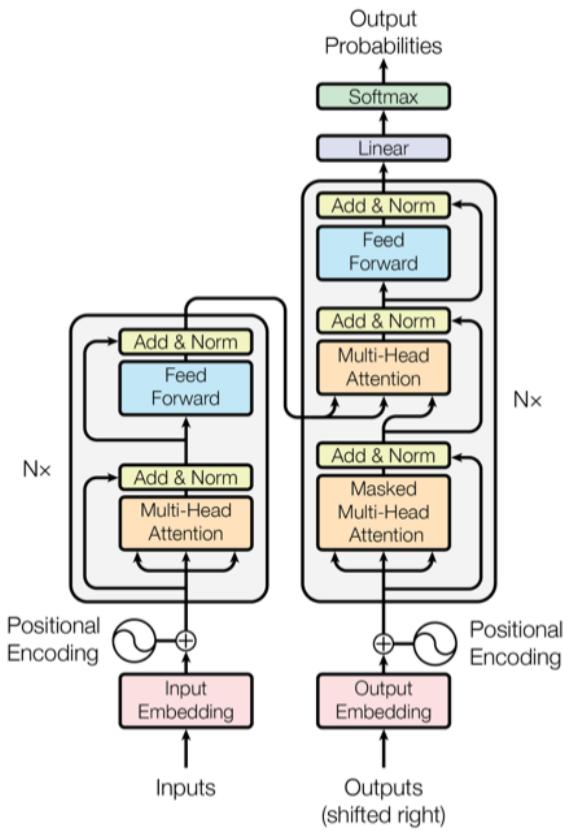


**Decoders**  
GPT, Claude,  
Llama, Mixtral

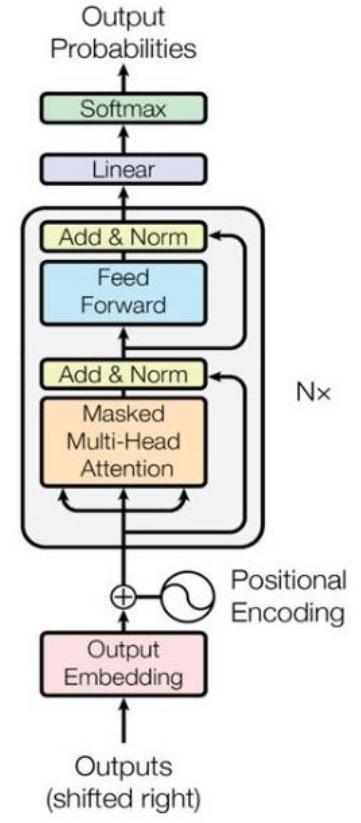
# Large Language Models



Encoder-only

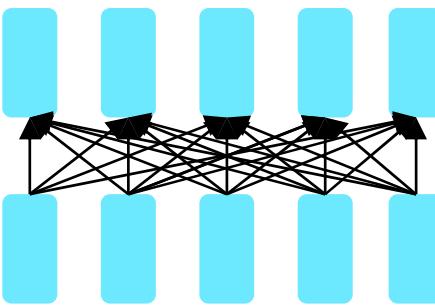


Encoder-decoder



Decoder-only

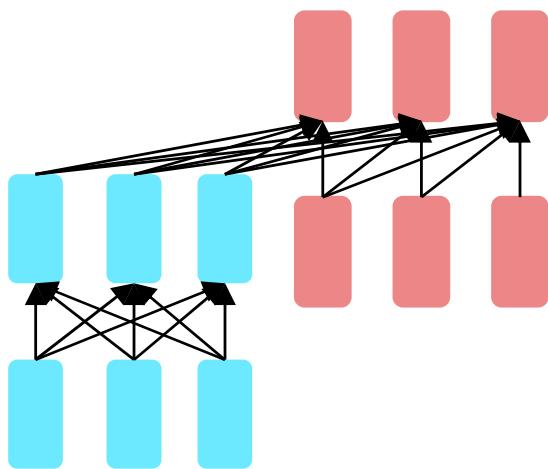
# Encoder-only



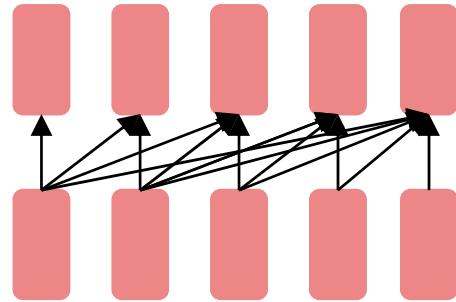
- Molte varietà!
  - Masked Language Models (MLMs)
  - Modelli BERT-like
- Addestrati a partire da una parola target e quelle nel suo intorno destro e sinistro
- Sono solitamente finetuned (addestrati su dati supervised) per compiti di classificazione

# Encoder-Decoders

- Addestrati a mappare da una sequenza a un'altra
- Molto popolari per:
  - machine translation (mappare da una lingua all'altra)
  - speech recognition (mappare dall'acustica alle parole)

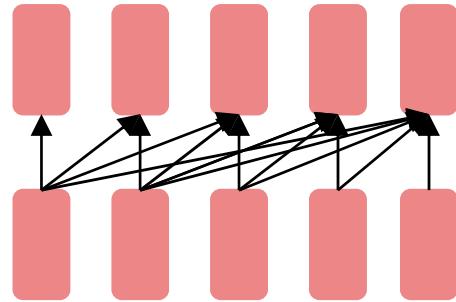


# Decoder-only



- Cosa possono fare questi modelli rispetto ai precedenti?
- Tutti i task precedenti possono essere riformulati in maniera generativa
  - Task di tipo *next-word prediction*

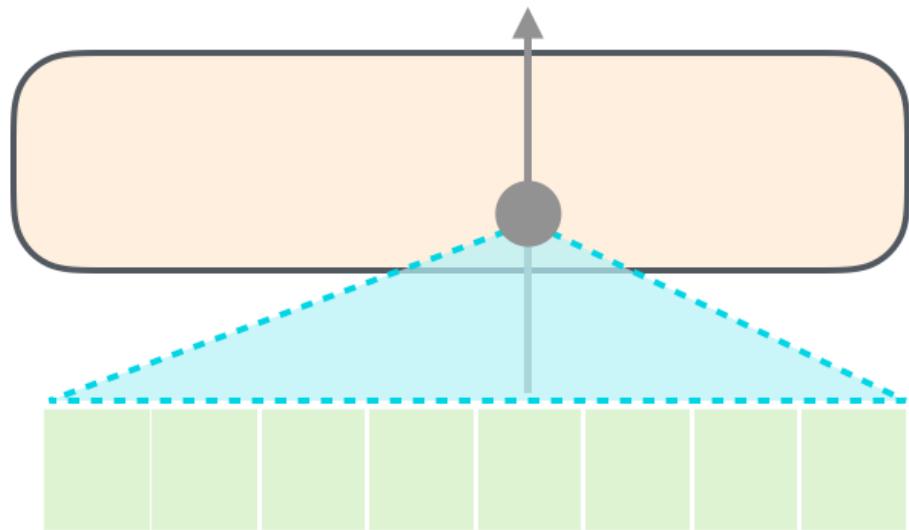
# Decoder-only



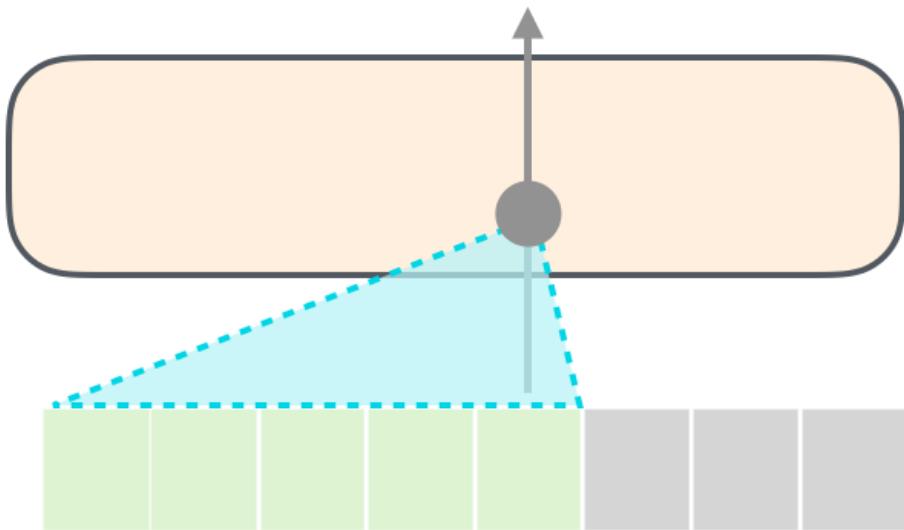
- Chiamati anche
  - Causal LLMs
  - Autoregressive LLMs
  - Left-to-right LLMs
- Predicono le parole da sinistra a destra

# Decoder-only

**Self-Attention**

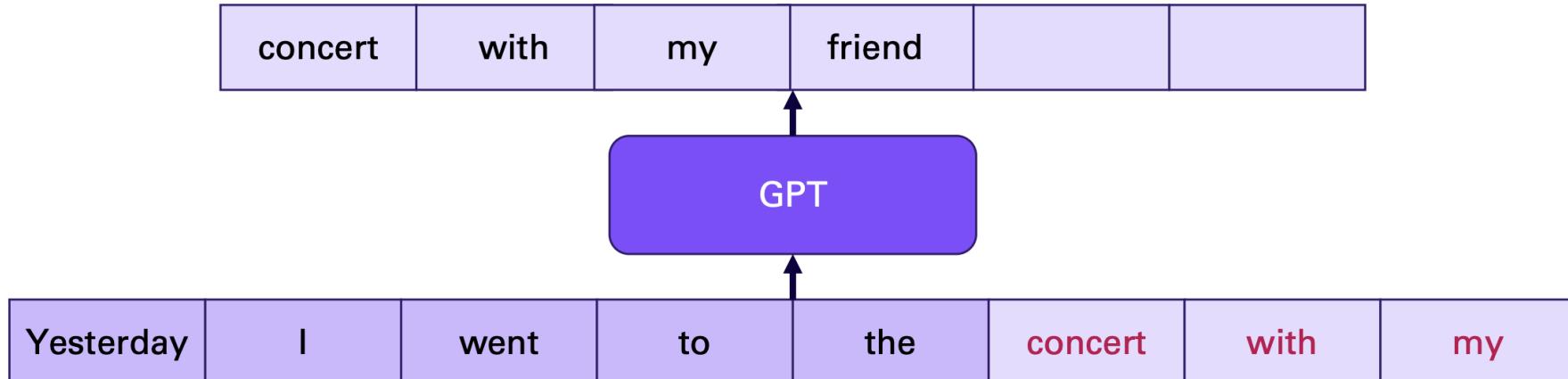


**Masked Self-Attention**



# Decoder-only – Next token prediction

- Masked self-attention + decoder-only
- Tecnica di addestramento per prevedere il token successivo in una sequenza, imparando a generare testo in base al contesto precedente.



# Decoder-only – Next token prediction

- I modelli decoder-only sono modelli linguistici generativi in grado di prevedere e generare nuove parole a partire dall'input fornito.
- Questo è generalmente formalizzato attraverso la chain rule.

$$P(w_{1:n}) = \prod_{i=1}^n P(w_i | w_{1:i-1})$$

# Decoder-only – Autoregressione

- In fase di training il token corretto viene iniettato nel modello a prescindere dalla predizione effettuata dallo stesso (teacher forcing)
- In fase di inferenza lo stesso token generato è ridato in input al modello
- Se l'output dipende dai token precedenti, posso specificare un'istruzione da dare al modello così da generare il testo desiderato

# Decoder-only – Autoregressione

- L'algoritmo di autoregressione è utilizzato in fase di inferenza, per condizionare la generazione dell'output con l'output precedentemente generato

---

## Algorithm 1 Overview of the autoregressive algorithm

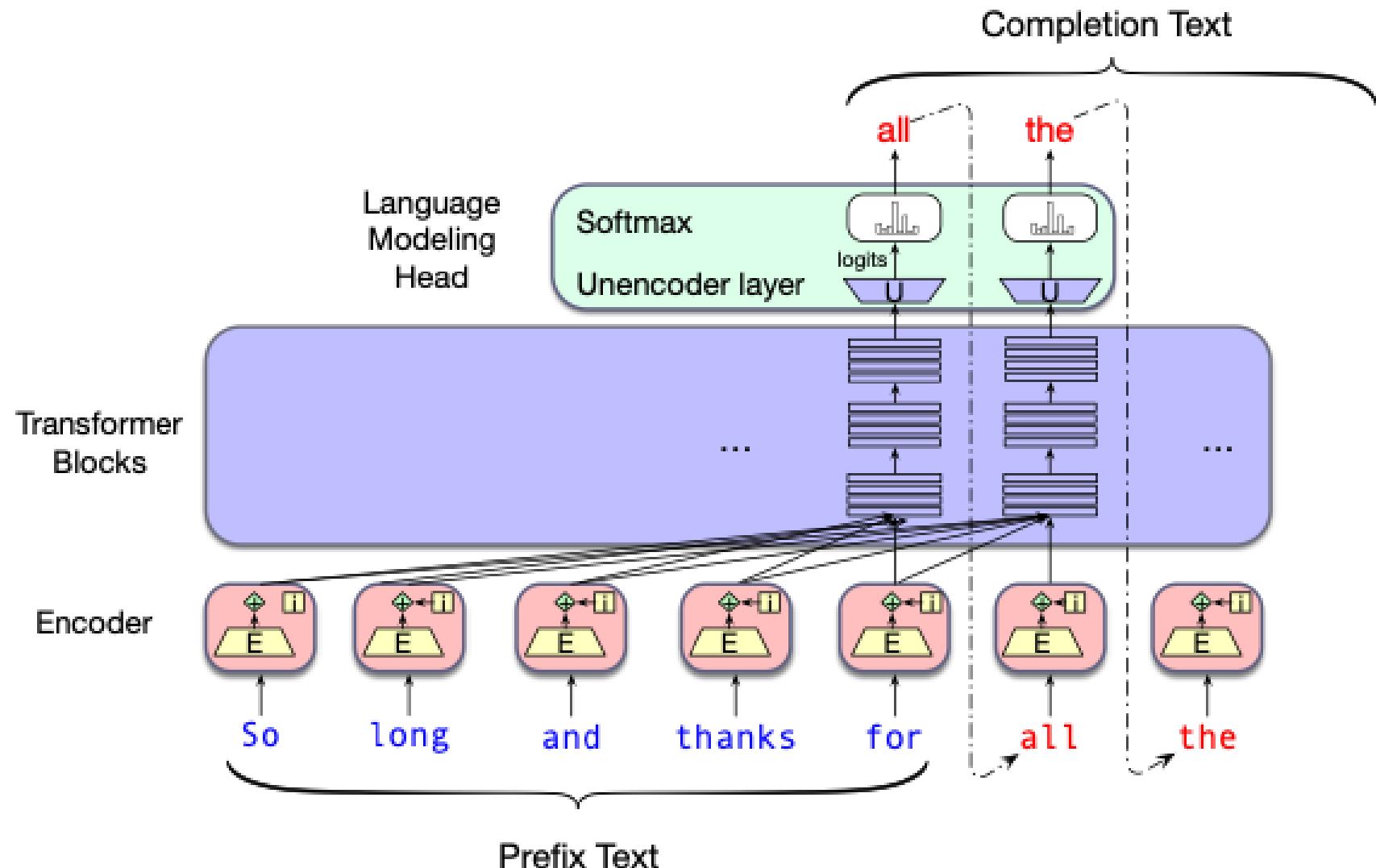
---

```
i ← 1  
wi ~ P(w)  
while wi ≠ EOS do  
    i ← i + 1  
    wi ~ P(wi|w<i)  
end while
```

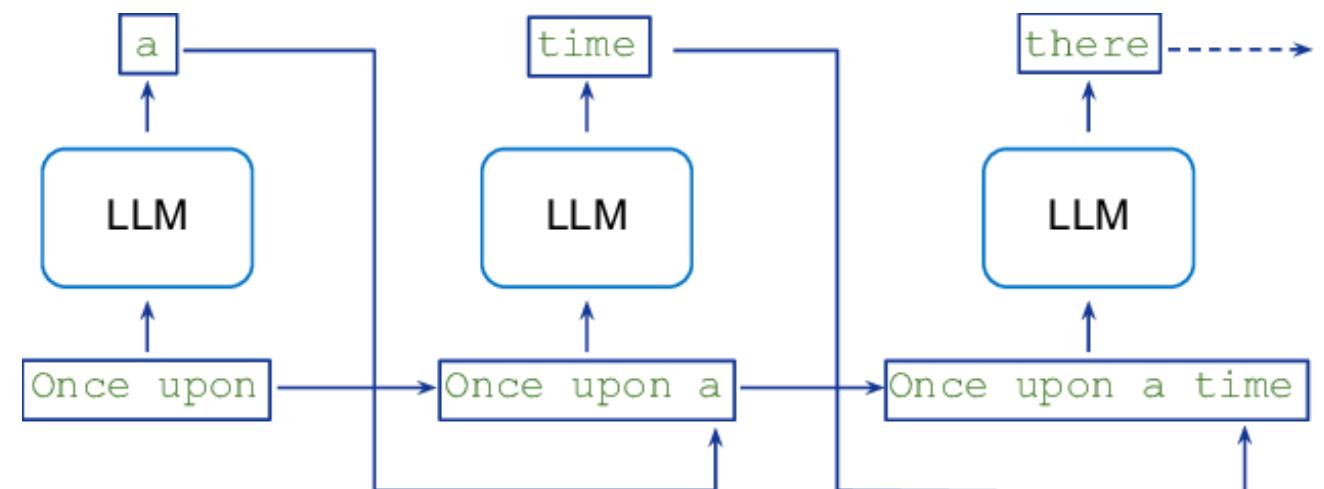
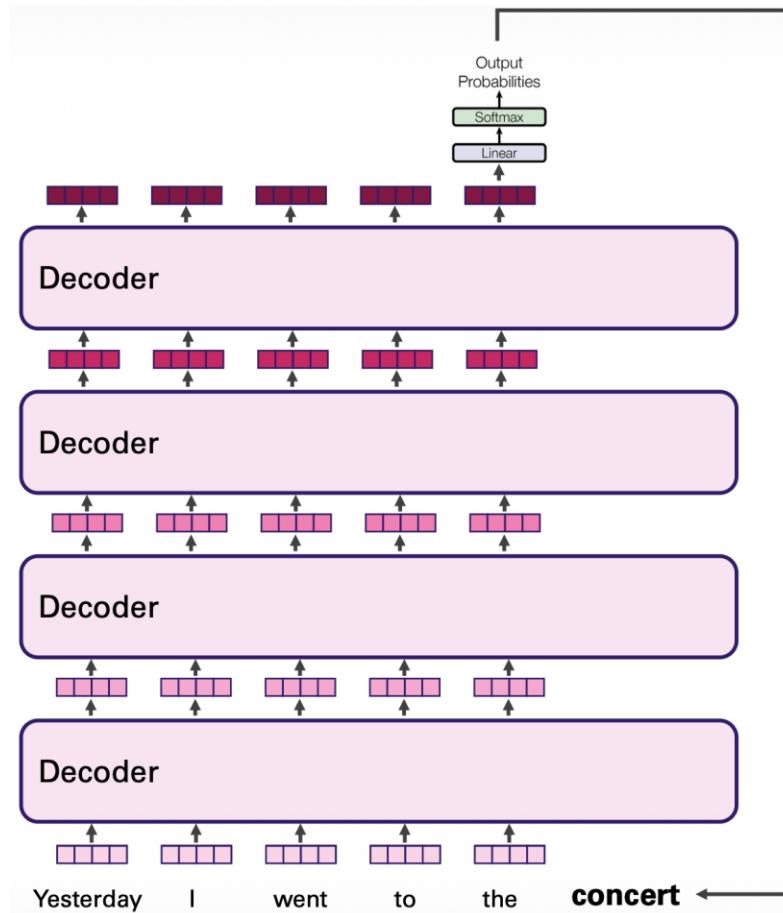
---

P(w) è la *strategia* con cui vengono *campionati* e quindi generati i token

# Decoder-only – Autoregression



# Decoder-only – Autoregression



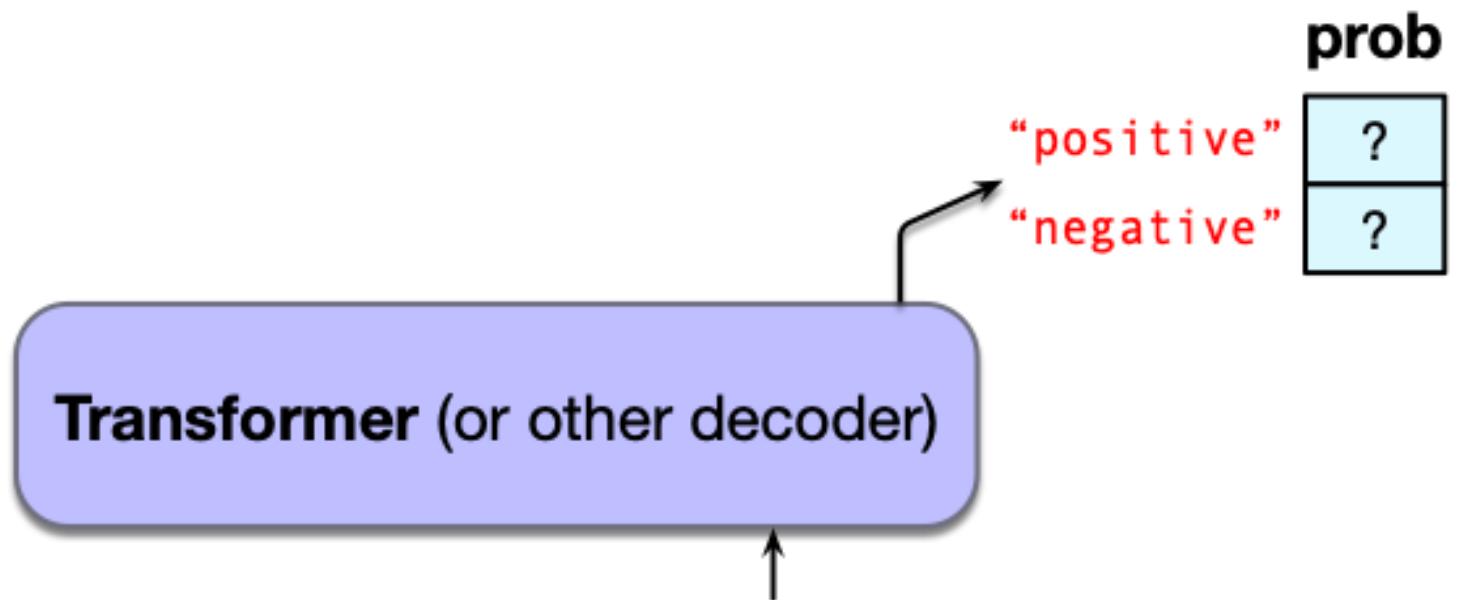
# Decoder-only – Autoregressione

- Se l'output dipende dai token precedenti, posso specificare un ***prompt*** (*istruzione*) da dare al modello così da generare l'output desiderato.

# Decoder-only – Autoregressione

- In quest'ottica i modelli generativi possono svolgere una serie di compiti tradizionali dei modelli encoder-only (*classificazione*) e encoder-decoder (*traduzione*) ma anche nuovi (*generazione*)

# Sentiment analysis



The sentiment of the sentence “I like Jackie Chan” is:

# Sentiment analysis

- Diamo al language model questa stringa:

The sentiment of the sentence "I like Jackie Chan" is:

- E vediamo quale parola pensa che venga dopo:

$P(\text{positive} | \text{The sentiment of the sentence } \text{"I like Jackie Chan"} \text{ is:})$

$P(\text{negative} | \text{The sentiment of the sentence } \text{"I like Jackie Chan"} \text{ is:})$

- Il modello genererà l'etichetta più probabile, che completa meglio il prompt, fornendo così la predizione

# Question-Answering

**QA:** "Who wrote The Origin of Species"

- Diamo al language model questa stringa:

Q: Who wrote the book "The Origin of Species"? A:

- E vediamo quale parola pensa che venga dopo:

$P(w|Q: \text{Who wrote the book "The Origin of Species"? A:})$

- E iteriamo:

$P(w|Q: \text{Who wrote the book "The Origin of Species"? A: Charles})$

# Summarization

## Testo originale

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For \$89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” says Waring’s website, ShipSnowYo.com. “We’re in the business of expunging snow!”

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

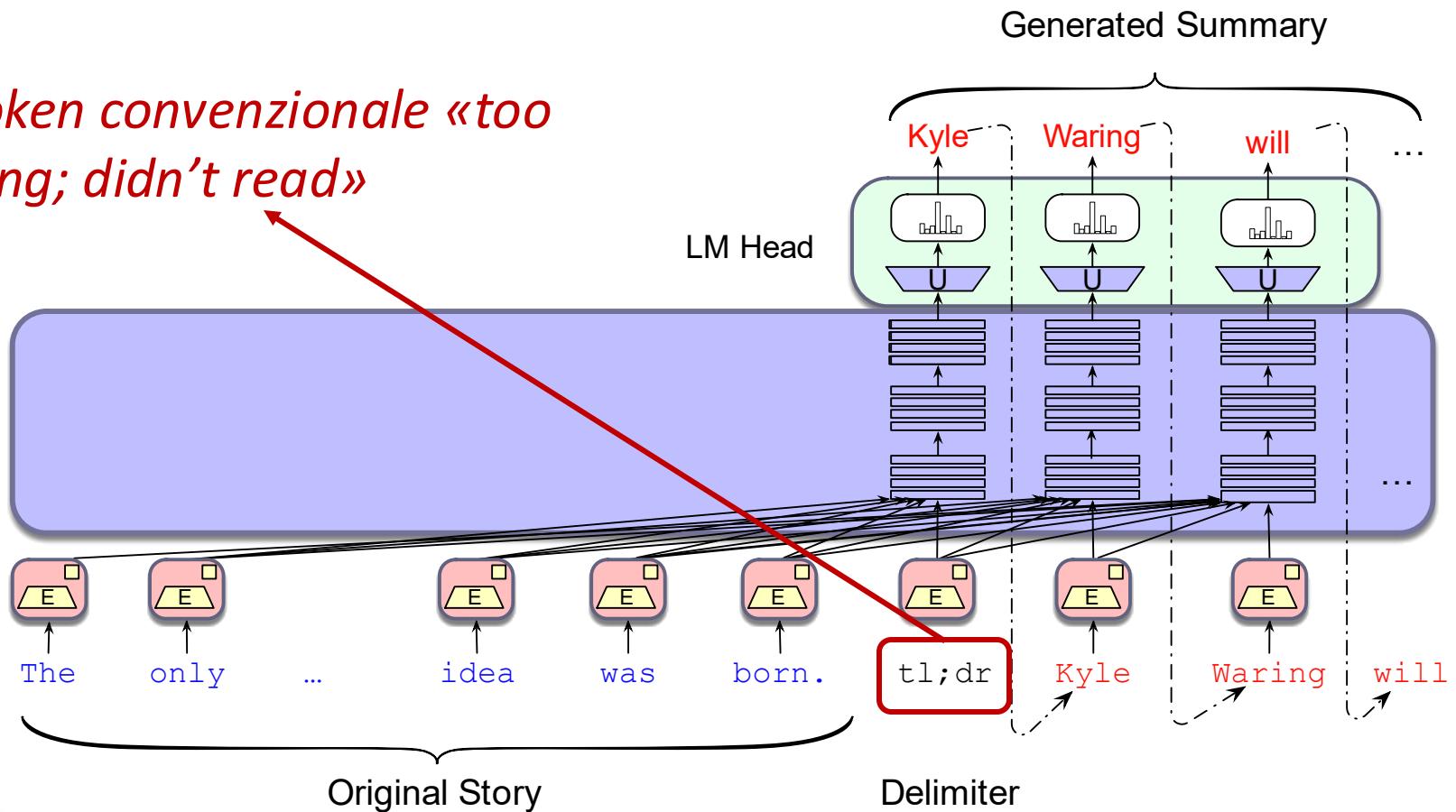
According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. [...]

## Riassunto

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

# Summarization

*Token convenzionale «too long; didn't read»*

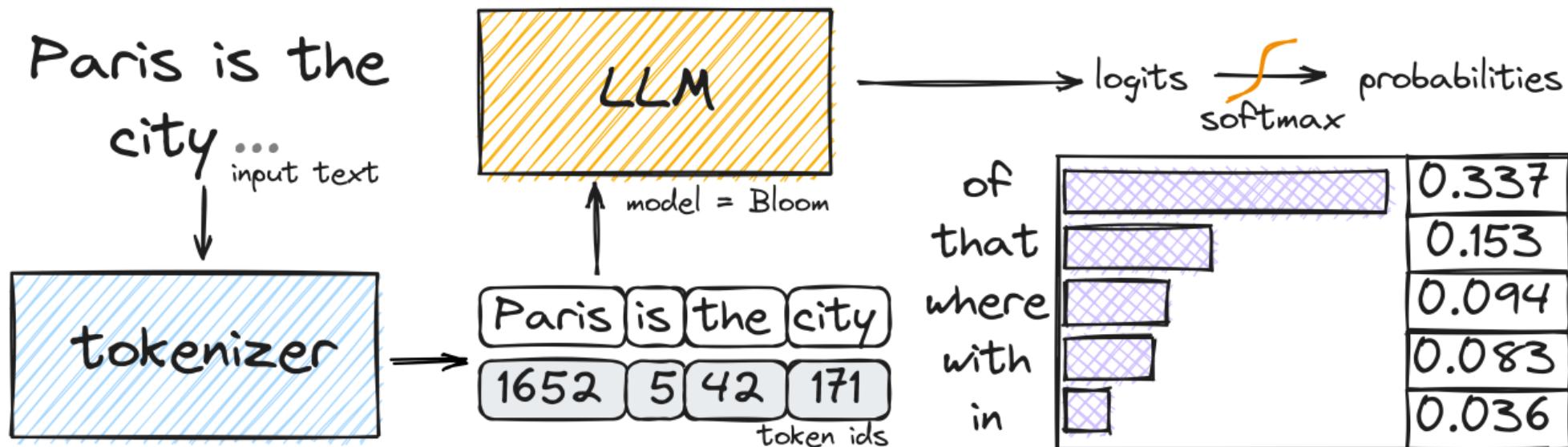


# Decoding

Testo → tokens → LLM → token predetto → testo predetto

# Decoding

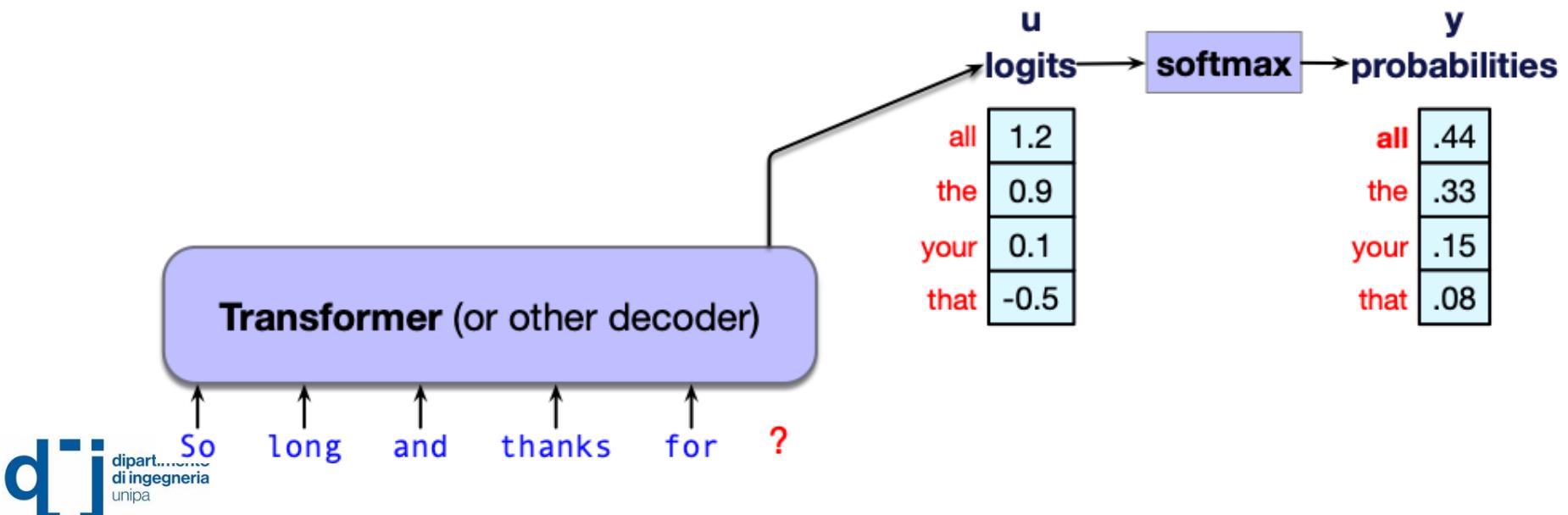
Testo → tokens → LLM → token predetto → testo predetto



Come seleziono i token che formano il testo in output?

# Decoding

- I layer neurali degli LLM generano i logit per ogni token nel vocabolario.
- Il vettore di punteggio  $\mathbf{u}$  con shape  $[1 \times |V|]$  viene trasformato in un vettore di probabilità  $\mathbf{y}$  tramite softmax



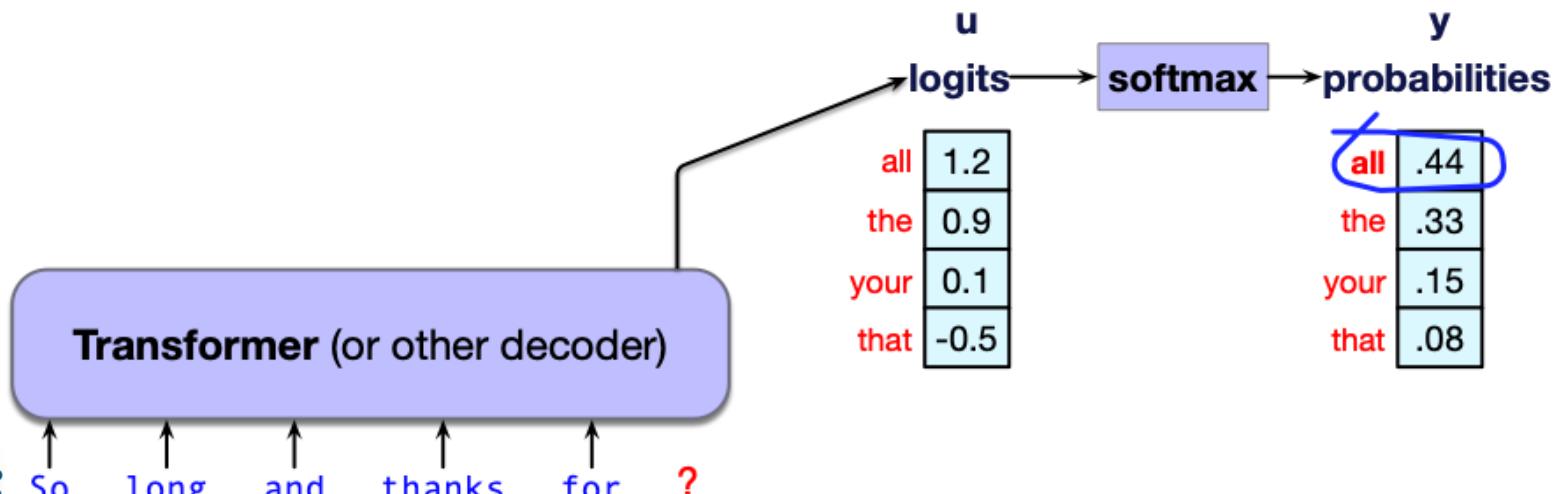
# Decoding and Sampling

- Il decoding è la scelta della parola da generare basandosi sulle su in criterio in base alle probabilità fornite in output dal modello
- Decoding deterministico
  - Greedy search, beam search
- Decoding stocastico (sampling)
  - Random sampling, Top-k sampling, Top-p sampling (nucleus sampling), Temperature sampling, contrastive seach

# Greedy Search

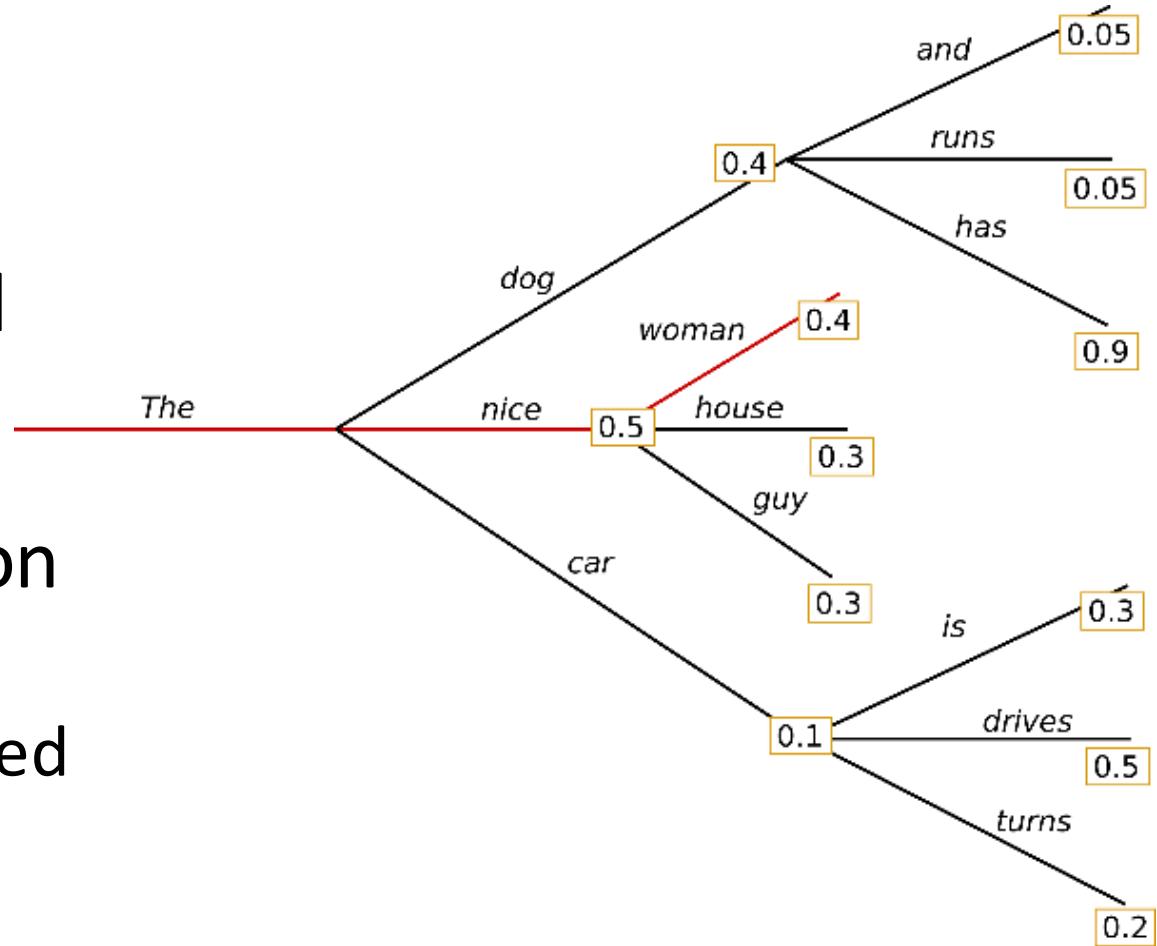
- La Greedy search è il più semplice metodo di decoding
- Seleziona la parola con maggiore probabilità di essere selezionata come parola successiva

$$w_t = \operatorname{argmax}_w P(w | w_{1:t-1})$$



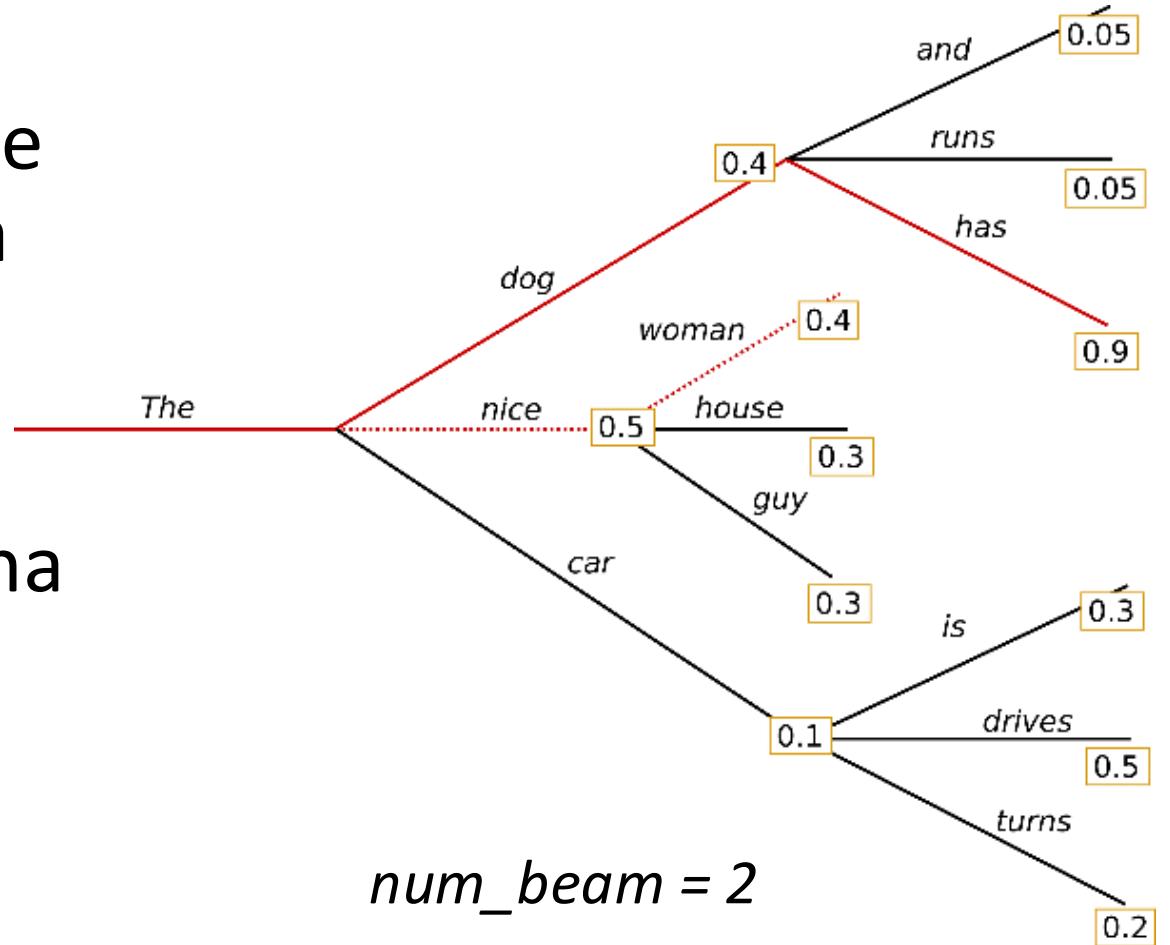
# Greedy Search

- Algoritmo che fornisce la scelta localmente migliore
- Le frasi generate tendono ad essere ripetitive
- Una ricerca di questo tipo non sempre è la migliore:
  - Has ha una probabilità di 0.9 ed è nascosta da dog



# Beam search

- Beam search riduce i rischi legati alla perdita di sequenze di parole con alta probabilità
- Mantiene in memoria  $num\_beam$  ipotesi e seleziona quella con la maggiore probabilità



# Beam search

- La Beam search con `num_beam = 1` coincide con la greedy search
- Entrambe le strategie sono prevedibili e posso generare risultati ripetitivi e poco interessanti
- Cosa succede se aggiungiamo un elemento randomico?

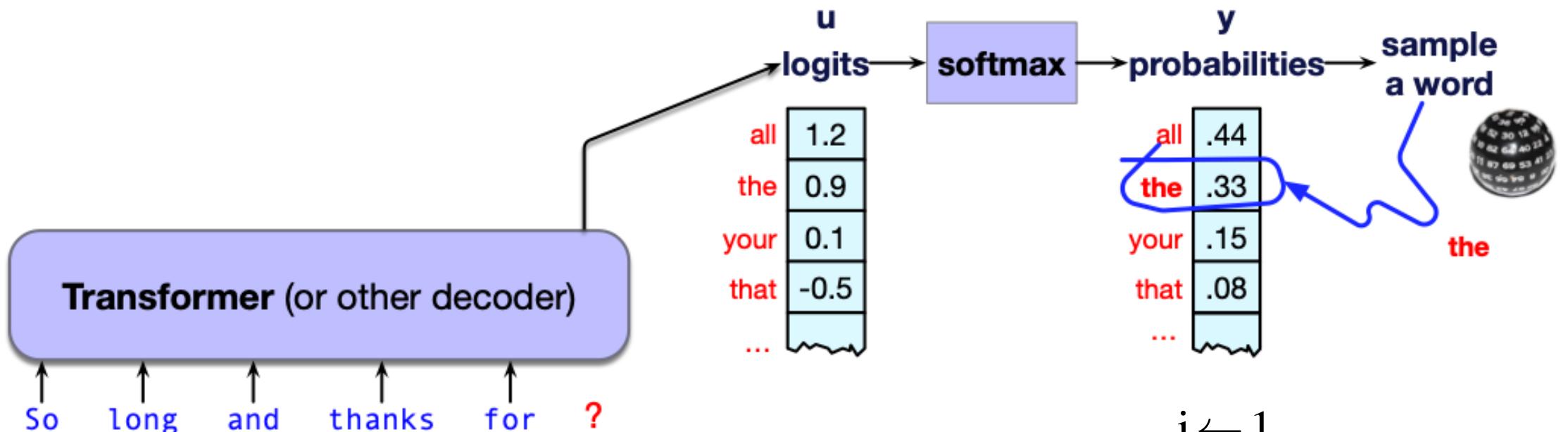
# Random sampling

- Sampling (campionamento) da una distribuzione significa scegliere punti casuali in base alla loro probabilità.
- Sampling da un LLM significa scegliere il token successivo da generare in base alla sua probabilità.

# Random sampling

- Random (multinomial) sampling: il token da generare è selezionato casualmente in base alla sua probabilità definita dal modello, condizionata dalle scelte precedenti, con un processo iterativo

# Random sampling



*Dopo ogni token campioneremo parole da generare in base alla loro probabilità condizionata dai token precedentemente predetti*

$$i \leftarrow 1$$

$$w_i \sim p(w)$$

**while**  $w_i \neq \text{EOS}$

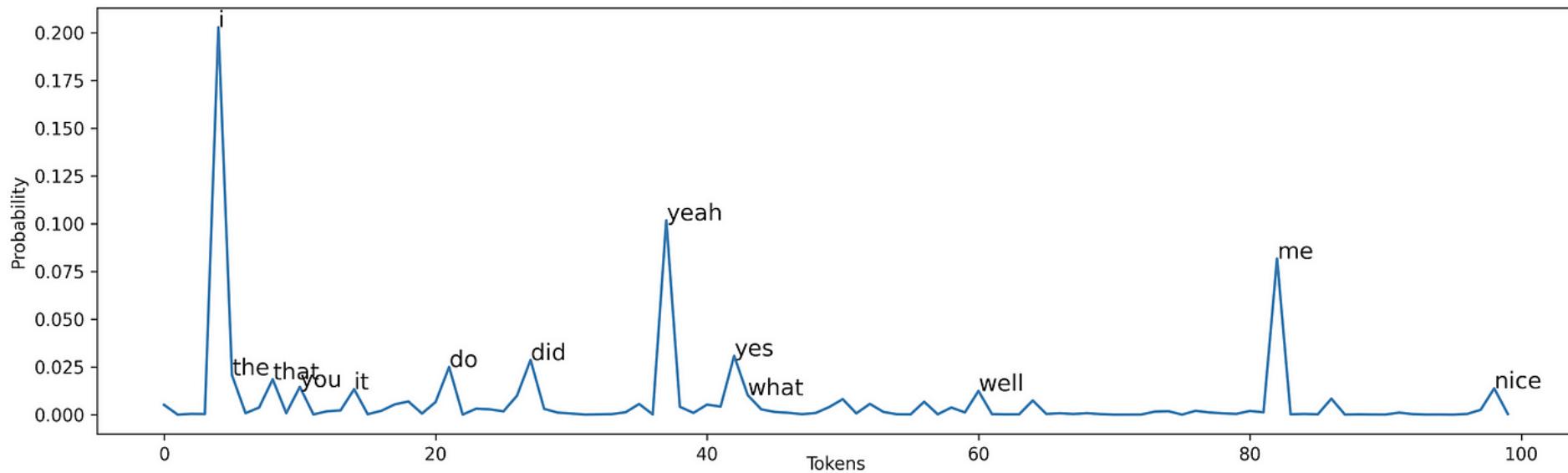
$$i \leftarrow i + 1$$

$$w_i \sim p(w_i \mid w_{<i})$$

# Random sampling

Considerando il seguente contesto:

I love watching movies



*I, yeah e me sono i token più probabili*

# Random sampling

- Anche se il random sampling genera per lo più parole sensate e ad alta probabilità, ci sono molte parole strane e a bassa probabilità nella distribuzione
- Ognuna ha bassa probabilità, ma sommate costituiscono una grande porzione della distribuzione
- Ma possono essere selezionate e portare alla generazione di frasi strane

# Random sampling – qualità e diversità

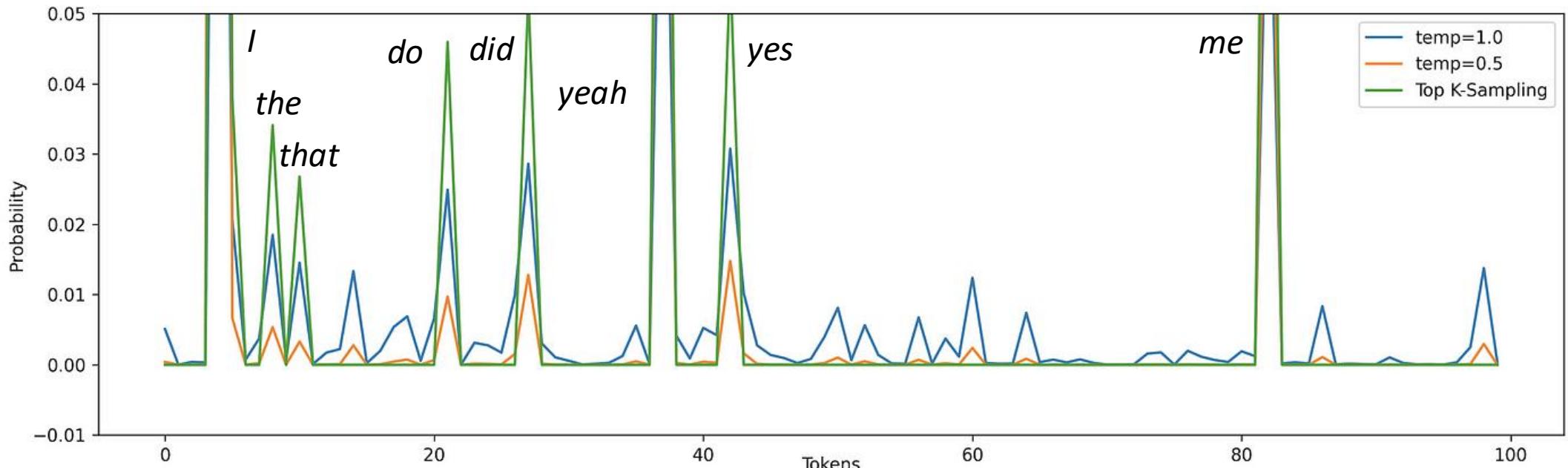
- Enfatizzare parole ad alta probabilità
  - + quality: più accurate, coerenti e fattuali
  - diversity: noiose, ripetitive
- Enfatizzare parole a media probabilità
  - + diversity: più creative, diverse
  - quality: meno fattuali, incoerenti

# Top- $k$ sampling

1. Scegli il numero di parole  $k$
2. Per ogni parola nel vocabolario  $V$ , usa il LLM per calcolare la verosimiglianza (likelihood) di questa parola dato il contesto  $p(w_t | \mathbf{w}_{<t})$
3. Ordina le parole per verosimiglianza, mantieni solo le  $k$  parole più probabili
4. Rinormalizza i punteggi delle  $k$  parole per ottenere una distribuzione di probabilità legittima
5. Campiona casualmente una parola all'interno di queste  $k$  parole più probabili rimanenti in base alla sua probabilità

# Top- $k$ sampling

- Nel nostro esempio in corso abbiamo campionato con  $k=8$
- Confrontiamo con la linea blu (random sampling)



# Top- $p$ sampling (Holtzman et al., 2020 )

- Detto anche nucleus sampling
- Nel caso di top-k sampling, il valore k è fisso quindi può coprire quantità molto diverse di massa di probabilità in situazioni diverse

# Top- $p$ sampling (Holtzman et al., 2020 )

💡 Nel top- $p$  sampling viene mantenuta fissa la massa di probabilità del top  $p$  percento

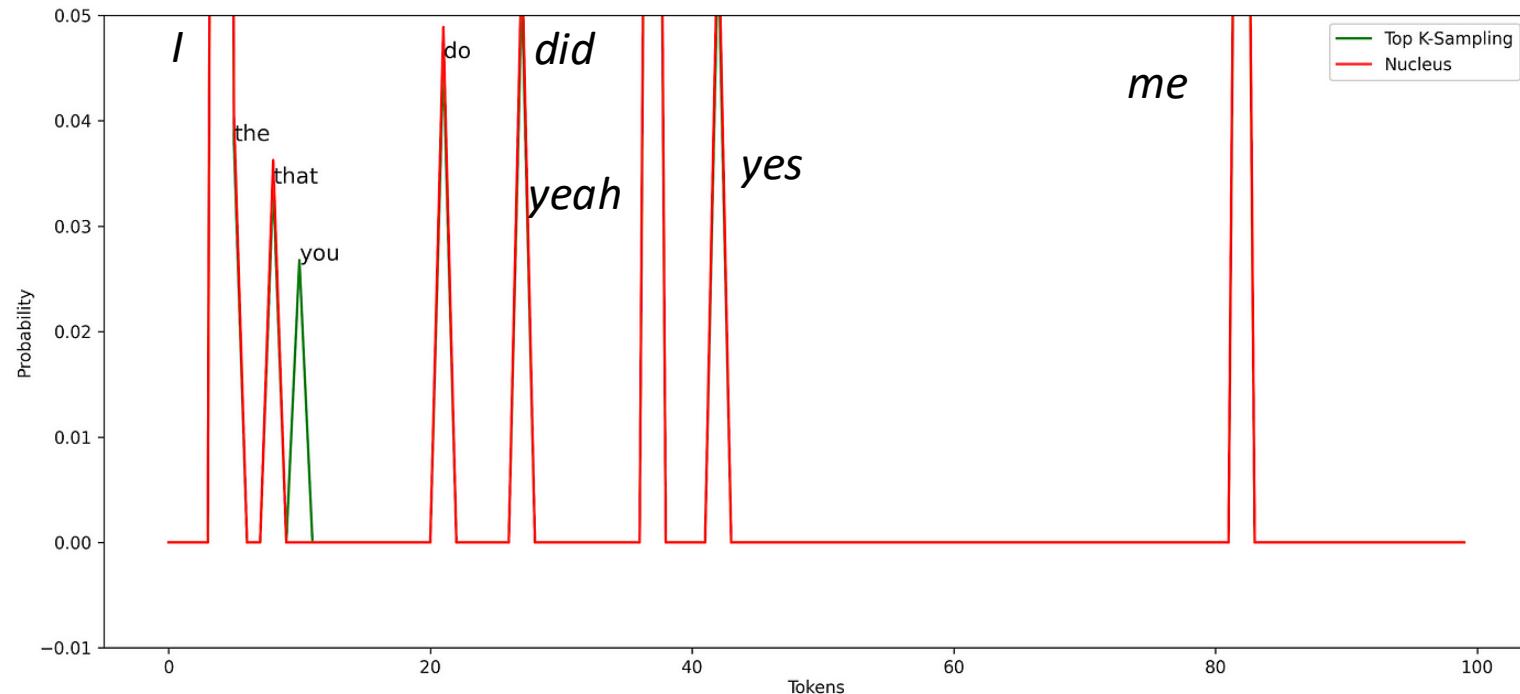
- Data una distribuzione  $P(w_t | \mathbf{w}_{<t})$ , il vocabolario top- $p$   $V(p)$  è l'insieme più piccolo di parole tale che

$$\sum_{w \in V^{(p)}} P(w | \mathbf{w}_{<t}) \geq p$$

*Implementa una sorta di selezione adattiva del valore di k*

# Top- $p$ sampling (Holtzman et al., 2020 )

- Nel nostro esempio in corso, «you» non è stato campionato a causa del valore della soglia  $p$



# Temperature sampling

- Rimodella la distribuzione invece di troncarla secondo un parametro di *temperatura*  $\tau$
- Intuizione dalla termodinamica
  - un sistema ad alta temperatura è flessibile e può esplorare molti possibili stati
  - un sistema a bassa temperatura è probabile che esplori un sottoinsieme di stati ad energia inferiore (migliore)

# Temperature sampling

- Nel *low temperature sampling*, ( $\tau \leq 1$ )
  - aumentiamo in modo uniforme la probabilità delle parole più probabili
  - diminuiamo la probabilità delle parole rare

# Temperature sampling

- Si divide il logit per il parametro di temperatura  $\tau$  prima di passarlo attraverso la softmax

Invece di

$$\text{y} = \text{softmax}(u)$$

calcolo

$$\text{y} = \text{softmax}(u/\tau)$$

# Temperature sampling

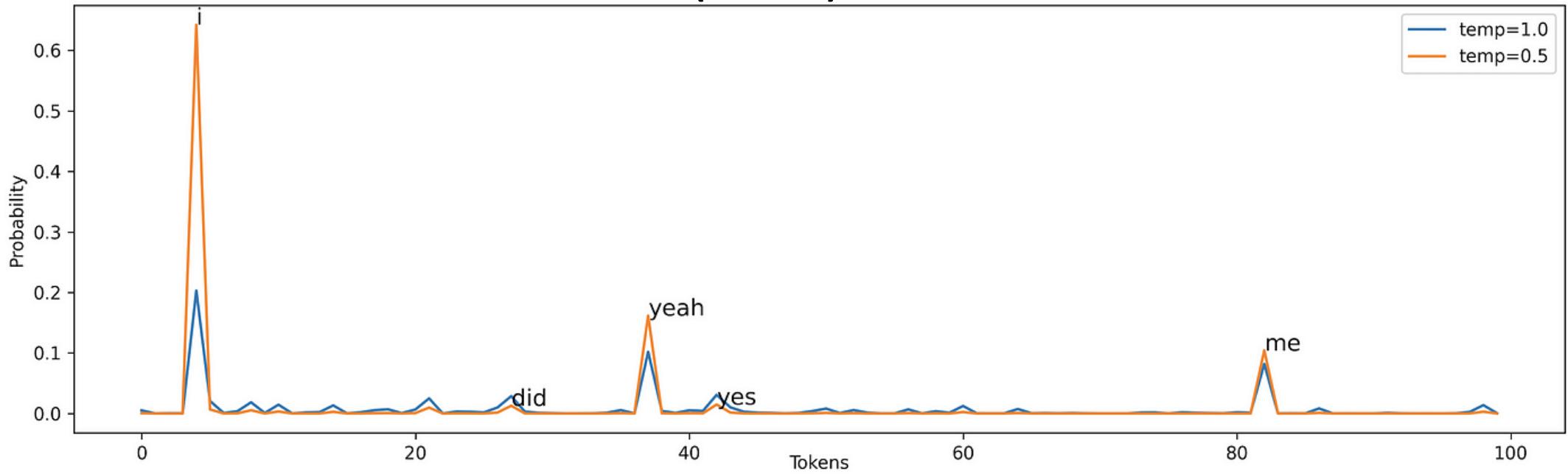
- Perché funziona?
  - Quando  $\tau$  è vicino a 1, la distribuzione non cambia molto
  - Più basso è  $\tau$ , più grandi sono i punteggi passati alla softmax
  - La softmax spinge i valori alti verso 1 e i valori bassi verso 0

# Temperature sampling

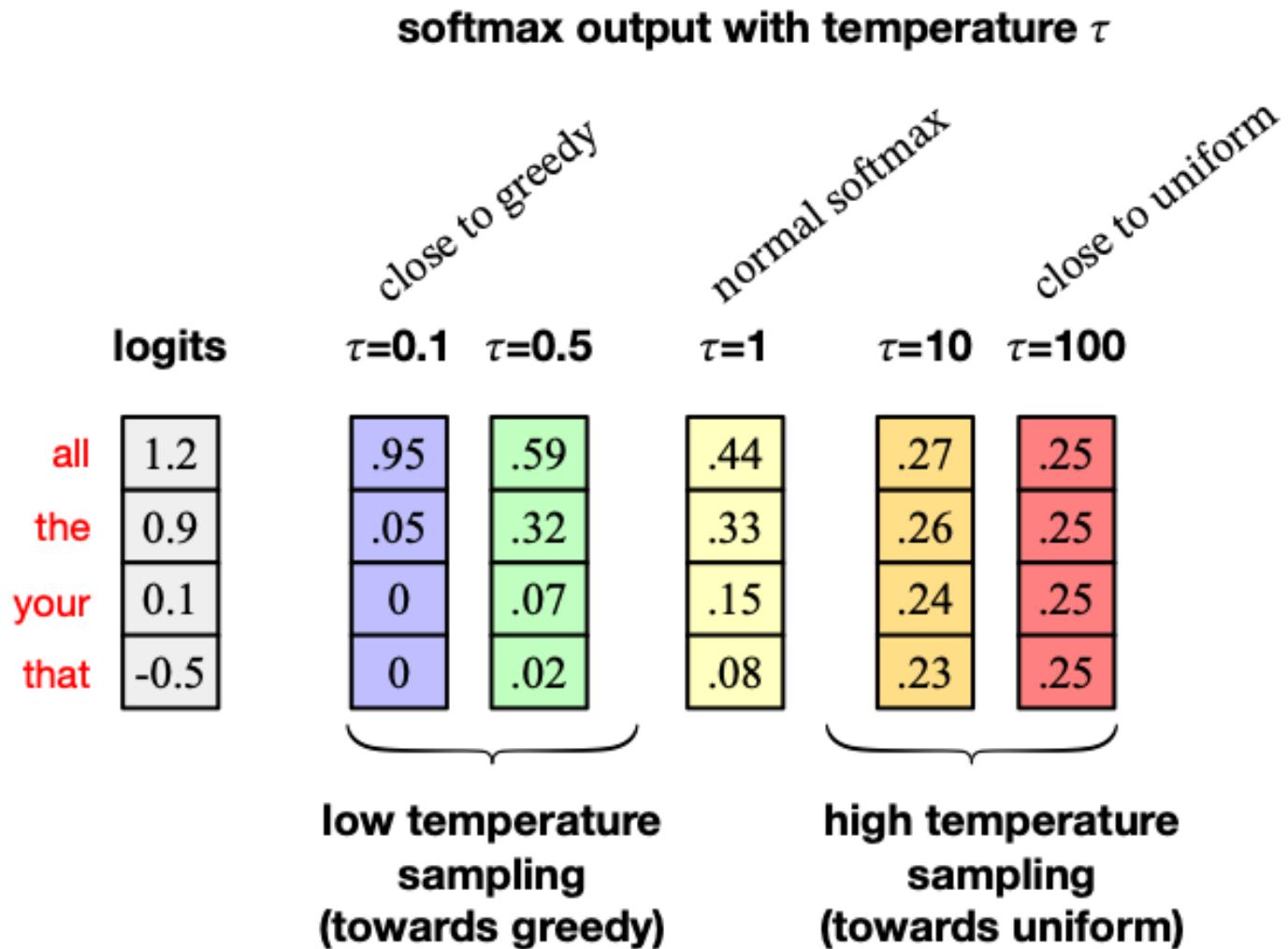
- Perché funziona?
  - Grandi input spingono le parole ad alta probabilità più in alto e le parole a bassa probabilità più in basso, rendendo la distribuzione più greedy
  - Mentre  $\tau$  si avvicina a 0, la probabilità della parola più probabile si avvicina a 1

# Temperature sampling

- Nel nostro esempio, *I*, *yeah*, e *me* hanno una maggiore probabilità di essere selezionati
- Confronta con la linea blu ( $\tau = 1$ )



# Temperature sampling



# Contrastive Search

- Dato il contesto  $\mathbf{x}_{<t}$ , il token successivo  $\mathbf{x}_t$  è generato come

$$x_t = \arg \max_{v \in V^{(k)}} \left\{ (1 - \alpha) \times \underbrace{p_\theta(v | \mathbf{x}_{<t})}_{\text{model confidence}} - \alpha \times \underbrace{\left( \max\{s(h_v, h_{x_j}) : 1 \leq j \leq t-1\} \right)}_{\text{degeneration penalty}} \right\}$$

- $V^{(k)}$  è l'insieme dei top- $k$  token predetti dal modello secondo la sua distribuzione di probabilità  $p_\theta(v | \mathbf{x}_{<t})$ , ovvero la ***model confidence*** cioè la probabilità che il modello generi  $v$

# Contrastive Search

- Dato il contesto  $\mathbf{x}_{<t}$ , il token successivo  $\mathbf{x}_t$  è generato come

$$x_t = \arg \max_{v \in V^{(k)}} \left\{ (1 - \alpha) \times \underbrace{p_\theta(v | \mathbf{x}_{<t})}_{\text{model confidence}} - \alpha \times \underbrace{\left( \max\{s(h_v, h_{x_j}) : 1 \leq j \leq t-1\} \right)}_{\text{degeneration penalty}} \right\}$$

- La *degeneration penalty* misura quanto è discriminativo  $v$ , rispetto al contesto precedente.
- La funzione  $s(\cdot, \cdot)$  calcola la massima similarità coseno tra gli embedding dei token.

# Contrastive Search

- Dato il contesto  $\mathbf{x}_{<t}$ , il token successivo  $\mathbf{x}_t$  è generato come

$$x_t = \arg \max_{v \in V^{(k)}} \left\{ (1 - \alpha) \times \underbrace{p_\theta(v | \mathbf{x}_{<t})}_{\text{model confidence}} - \alpha \times \underbrace{\left( \max\{s(h_v, h_{x_j}) : 1 \leq j \leq t-1\} \right)}_{\text{degeneration penalty}} \right\}$$

- Maggiore il valore della degeneration penalty, più il token generato è simile al contesto nello spazio degli embedding.
- Questo comporta la possibilità di riscontrare un peggioramento nel testo generato dal modello.

# Contrastive Search

- Dato il contesto  $\mathbf{x}_{<t}$ , il token successivo  $\mathbf{x}_t$  è generato come

$$x_t = \arg \max_{v \in V^{(k)}} \left\{ (1 - \alpha) \times \underbrace{p_\theta(v | \mathbf{x}_{<t})}_{\text{model confidence}} - \alpha \times \underbrace{\left( \max\{s(h_v, h_{x_j}) : 1 \leq j \leq t-1\} \right)}_{\text{degeneration penalty}} \right\}$$

- L'iperparametro  $\alpha$  regola l'importanza delle due componenti
- Se  $\alpha=0$ , la contrastive search degenera alla greedy search.

# Addestramento

- Cosa significa addestrare un LLM?
  - Foundation models, puri language model, *pretrained*
  - Instruction models, modelli in grado di generare del testo seguendo le istruzioni date in input
  - Finetuned models, modelli addestrati per lo svolgimento di un task specifico

# Pretraining

- L'idea fondamentale che è alla base delle incredibili performance dei language models è:
  - *Prima* eseguire il pretraining di un modello transformer su enormi quantità di testo.
  - *Poi* adattarlo per lo svolgimento di nuovi compiti.

# Algoritmo di training self-supervised

- Gli LLM sono addestrati per predire la parola successiva
- Partendo da un corpus di testo, ad ogni *time step t*,
  - Si chiede al modello di predire la parola successiva
  - Si addestra il modello usando la discesa del gradiente per minimizzare l'errore in questa predizione.
- Non serve un'annotazione esterna

# Cross-Entropy Loss

- Il task è multi-classe e quindi la loss è la Cross-Entropy (CE)
  - Differenza tra la distribuzione di probabilità **corretta** e **predetta**.

$$L_{CE} = - \sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

$$\mathbf{y}_t[w] = \begin{cases} 1, & w = w_{t+1} \\ 0, & \forall w \neq w_{t+1} \end{cases}$$

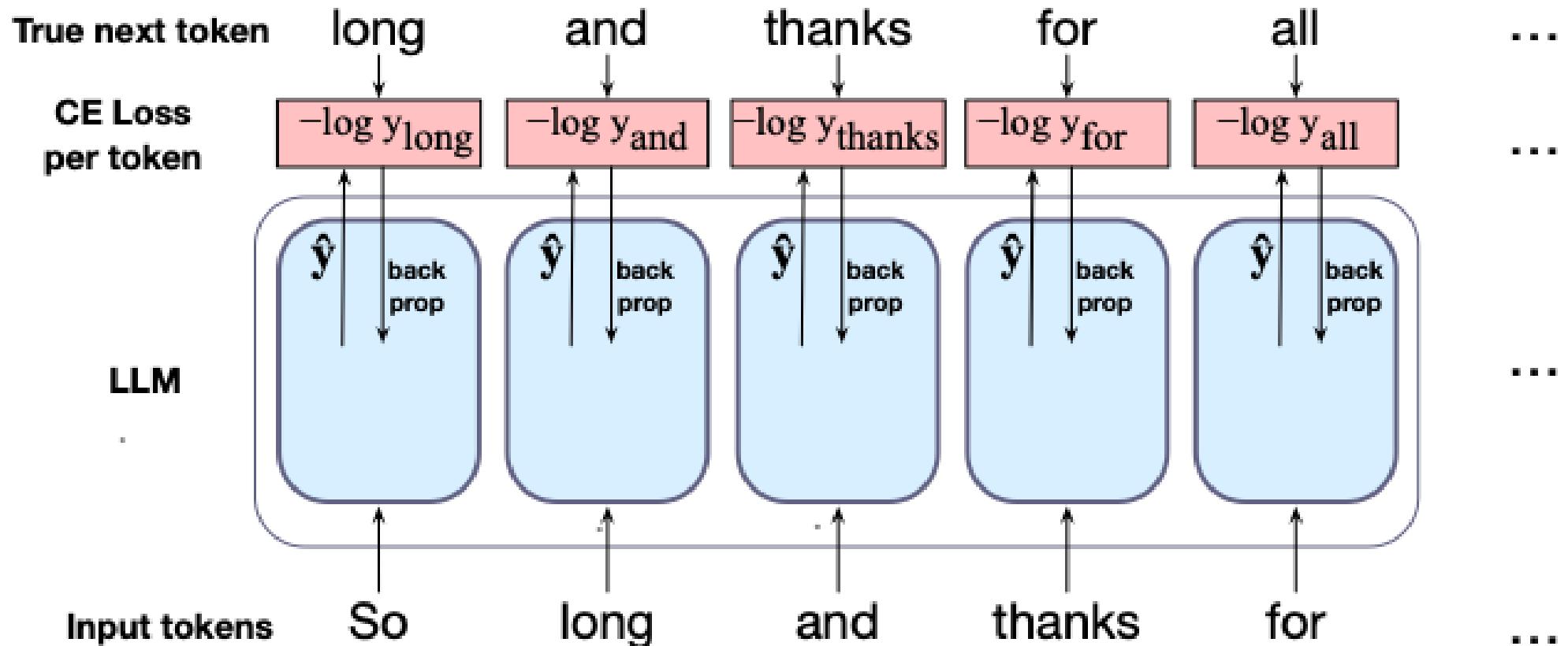
$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

*Negative Log Loss (NLL Loss)*

# Teacher Forcing

- La strategia di addestramento è il Teacher Forcing
- Ad ogni token in posizione  $t$ , il modello vede i *token* corretti  $w_{1:t}$  e calcola la *NLL Loss* per il *token* successivo  $w_{t+1}$
- Al token successivo in posizione  $t+1$  ignoriamo la parola  $\hat{w}_{t+1}$  predetta dal modello.
- Prendiamo la *parola corretta*  $w_{t+1}$ , la aggiungiamo al contesto e proseguiamo.

# Teacher Forcing



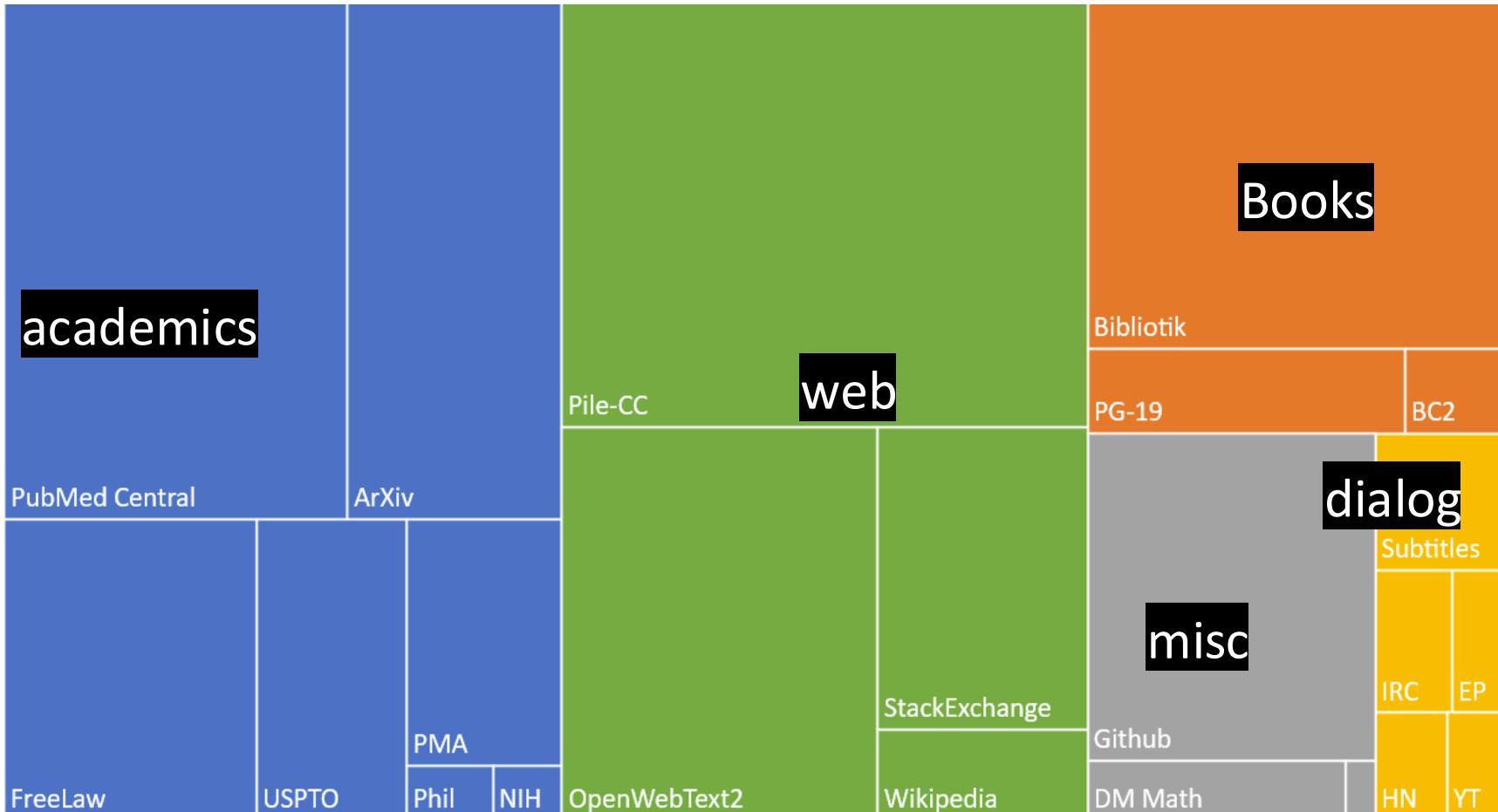
# Dati di addestramento

- Serve una grande quantità di dati per addestrare un LLM
- Da dove posso prendere dei dati?
- Con quale criterio?
- Cosa succede se addestro lo stesso modello con dei data set di addestramento diversi?

# Dati di addestramento

- Gli LLM sono principalmente addestrati sul dati web.
- Common Crawl, *scansione dell'intero web* prodotta dall'organizzazione non-profit Common Crawl con miliardi di pagine.
- Colossal Clean Crawled Corpus, detto anche C4, (Raffel et al. 2020) 156 miliardi di *tokens* di inglese, filtrato.
- Cosa contiene? Principalmente documenti di testo di brevetti, Wikipedia e siti di notizie.

# The Pile: un corpus di pretraining



# Filtraggio per qualità

- Il concetto di qualità è soggettivo e dipende dagli scopi
- Molti LLM tentano di replicare Wikipedia, libri, siti web particolari, ovvero i dati su cui sono stati addestrati

# Filtraggio per qualità

- Necessario rimuovere
  - Testi standard e/o ripetitivi
  - Contenuti per adulti
  - Dati sensibili e personali
- Deduplication (rimozione dei duplicati)
  - URL, documenti, righe

# Filtraggio per sicurezza

- La sicurezza è soggettiva e dipende dagli scopi
  - Le informazioni personali e sensibili
- La tossicità si riferisce alla generazione di contenuti dannosi, offensivi o inappropriati
  - Tali contenuti possono rafforzare bias, pregiudizi o stereotipi.
  - Testi così generati possono portare a risultati discriminatori e influire negativamente sugli individui e sulle comunità.
  - Viene rilevata eseguendo classificatori esterni ad hoc

# Web scraping – problemi

Authors Sue OpenAI Claiming Mass Copyright Infringement of Hundreds of Thousands of Novels



*The Times Sues OpenAI and Microsoft Over A.I. Use of Copyrighted Work*

Millions of articles from The New York Times were used to train chatbots that now compete with it, the lawsuit said.



# Web scraping – problemi

- Copyright
  - Gran parte del testo in questi data set è protetto da copyright.
  - Ambiguità nella legge per l'uso di questi dati (US, EU)
  - Questa rimane una questione legale aperta.
- Consenso sui dati
  - I proprietari dei siti web possono indicare di non volere che il loro sito venga scansionato.
- Privacy
  - I siti web possono contenere indirizzi IP privati e numeri di telefono.

# Web scraping – problemi

- Normative a confronto
- EU – Text and Data Mining (TDM): le tecniche TDM sono consentite per scopi di ricerca e anche commerciali posto che, in entrambi i casi, si abbia il *diritto legale* di accedervi
  - Istituzioni scientifiche: tramite licenze o abbonamenti
  - Uso commerciale: accesso legale e i titolari dei diritti non abbiano esplicitamente vietato l'uso nelle loro licenze

# Web scraping – problemi

- Normative a confronto
- USA – Fair Use of data: permette di utilizzare materiali protetti senza chiedere il permesso, a patto che l'uso sia limitato a scopi specifici (come l'istruzione o la critica).
- La legalità non è automatica ma si valuta bilanciando **quattro criteri**:
  1. Lo scopo (meglio se non commerciale).
  2. La natura dell'opera.
  3. La quantità usata.
  4. L'impatto economico sull'originale.

# Web scraping – problemi

- Questi problemi sui dati di addestramento dei modelli, hanno poi delle dirette conseguenze sul testo generato dai modelli che può contenere allucinazioni.
- Il modello può generare testo errato, privo di senso o contraddittorio, nonostante l'uso corretto della lingua e una capacità espositiva che mostra un apparente alto grado di affidabilità

# Cosa sa il modello dopo il pretraining?

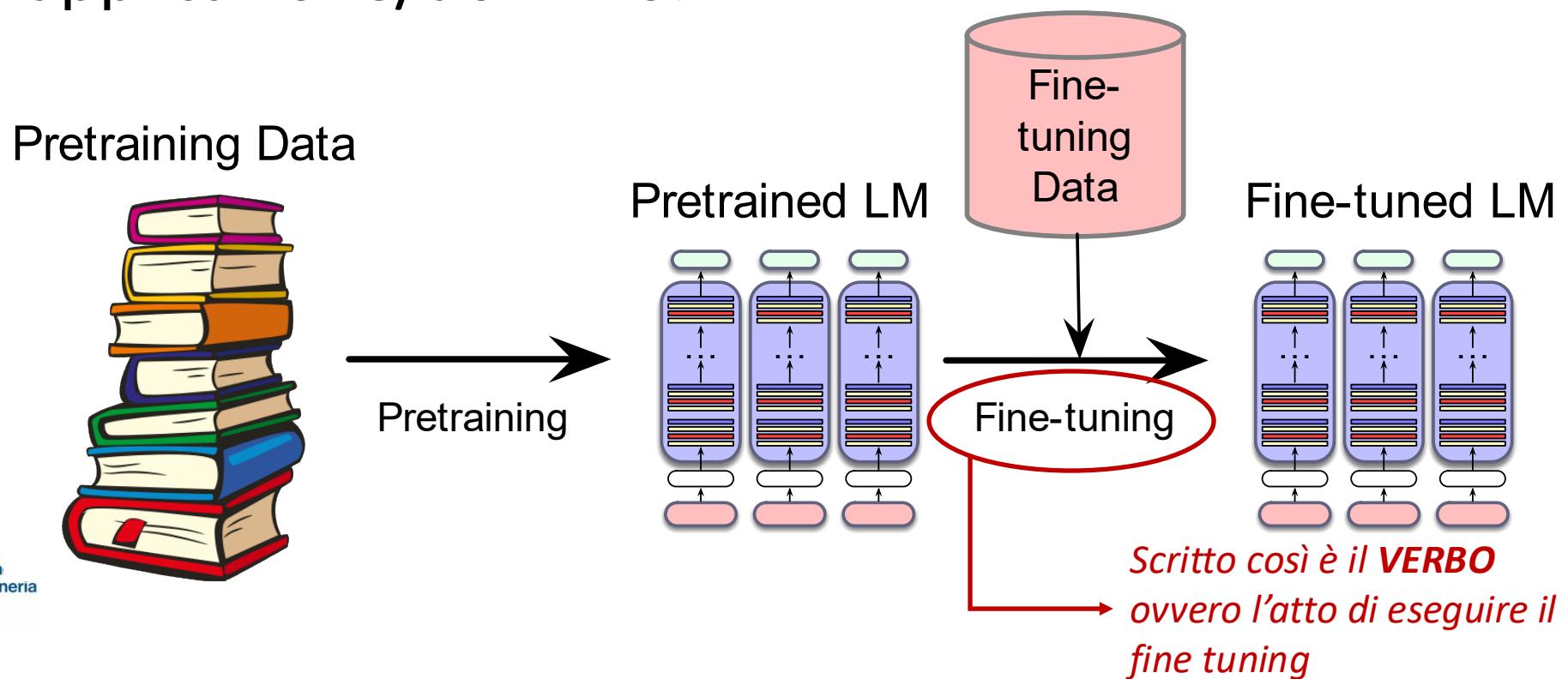
- Il testo contiene un'enorme quantità di conoscenza
- Il modello apprende della conoscenza generale in base ai dati su cui è stato addestrato
  - Non era grande, era enorme
  - L'autore della «Divina Commedia» è Dante Alighieri
  - Il medico mi ha detto che lui
  - La radice quadrata di 4 è 2

# Fine tuning

- Cosa succede se abbiamo bisogno che il nostro LLM funzioni bene in un dominio che non ha visto nel pretraining?
  - Dominio medico, legale, finanziario, dati aziendali
- Oppure, nel caso di un LLM multilingua, è necessario che questo venga addestrato su una lingua che era rara nel pretraining?

# Fine tuning

- Il fine tuning è il processo in cui un modello pre-addestrato è sottoposto ad ulteriori *pass* di training su nuovi dati specifici per un'applicazione/dominio.



# Fine tuning – tipologie

- Aggiungere circuiti neurali extra (*una testa ulteriore*) dopo lo strato superiore del modello per usarlo come classificatore
- Continued pretraining
- Parameter-efficient fine tuning (PEFT)
- Supervised fine tuning (SFT)

# Continued pretraining

- Addestrare ulteriormente *tutti i parametri* del modello su nuovi dati.
- Stessa strategia di addestramento e funzione di loss usati per il pretraining
  - Next-word prediction, Cross-Entropy Loss
- È come se i nuovi dati fossero in coda ai dati di pretraining.

# Parameter-Efficient Fine Tuning

- Riapplicare il training a tutti i parametri del modello è molto lento e costoso, soprattutto per grandi LLM.
- Possiamo congelare alcuni dei parametri (cioè, lasciarli invariati rispetto al loro valore di pretraining) e addestrare solo un *sottoinsieme di parametri* sui nuovi dati.
- PEFT perché selezioniamo in modo efficiente parametri specifici da aggiornare durante il fine tuning.

# Supervised Fine Tuning

- Usato per l'instruction finetuning, in cui vogliamo che un LLM sottoposto a pretraining impari a seguire istruzioni di testo.
  - Foundational LLM → Instruct LLM
- Essendo supervisionato, è necessario avere un data set annotato in cui ci sia siano coppie input-output
  - Question-Answering task

# Supervised Fine Tuning

- Creiamo un *data set di prompt* e risposte desiderate
  - domande e loro risposte, o comandi e loro esecuzioni.
- Addestriamo il modello utilizzando la cross-entropy loss per predire iterativamente *ogni token nella risposta desiderata*, cioè per produrre la risposta desiderata dal comando nel prompt.

# Valutazione

Le performance dei LLM possono essere valutate secondo diversi criteri:

- Accuracy
  - Accuratezza sulle predizioni a partire da un testo sconosciuto
- Task-oriented metrics
  - F1Score, BLEU, ROUGE che dipendono dal compito specifico
- Velocità di esecuzione
- Energia
- Fairness

# Valutazione

- Partendo dalla chain rule, tramite cui un LLM genera del testo

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\dots P(w_n|w_{1:n-1}) \\ &= \prod_{i=1}^n P(w_i|w_{<i}) \end{aligned}$$

- Due diversi LLM possono essere comparati tramite il confronto delle loro log likelihood

$$\text{log likelihood}(w_{1:n}) = \log \prod_{i=1}^n P(w_i|w_{<i})$$

# Valutazione

Ma le log-likelihood grezze hanno dei problemi:

- La probabilità dipende dalla grandezza del test set
- Le probabilità diventano sempre più piccole via via che il testo aumenta la sua lunghezza
- È preferibile una metrica che opera a livello della *singola parola* e che dopo sia normalizzata in base alla lunghezza del testo

# Perplexity

- La Perplexity è un'ottima scelta per i nostri scopi
- Per un *test set* di  $n$  *token*  $w_{1:n}$  la Perplexity di un modello  $\theta$  è:

$$\begin{aligned}\text{Perplexity}_\theta(w_{1:n}) &= P_\theta(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}}\end{aligned}$$

$$\text{Perplexity}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^n \frac{1}{P_\theta(w_i | w_{<i})}}$$

# Perplexity

- La perplexity è sensibile alle variazioni legate al processo di tokenizzazione
- E' bene usare lo stesso tokenizer per il calcolo della perplexity fra due modelli diversi.

# Valutazione

- Dimensione
  - I modelli grandi richiedono molte GPU e tempo per l'addestramento, nonché storage per essere archiviati, grandi quantità di dati di addestramento e risorse idonee per l'addestramento e l'inferenza.
- Consumo energetico
  - Può essere misurato in kWh o chilogrammi di CO<sub>2</sub> emessa.
- Fairness (Equità)
  - Valutazione del modello in base alle performance con benchmark che misurano stereotipi di genere e razziali
  - Si riferisce ai bias interni al modello

# Scalabilità

- I Large Language Models sono davvero *large*.
- Llama 3.1 405B Instruct model possiede:
  - 405B parametri divisi in 126 layers, di grandezza 16,384, con 128 attention heads
  - È stato addestrato su un corpus di 15.6 terabytes di tokens e con un vocabolario di 128k token
- La comunità scientifica ha indagato sulla scalabilità dei LLM e su come implementarli in contesti con risorse limitate.
  - Leggi di scaling, KV cache, PEFT, LoRA, Quantizzazione

# Leggi di scaling

Le performance degli LLM dipendono da:

- Model size
  - Numero di parametri senza contare gli *embeddings*
- Dataset size
  - Quantità di dati di training
- Compute
  - Floating point Operations Per Second (FLOPS)

# Leggi di scaling

- Si può migliorare un modello aggiungendo parametri (più strati, contesti più ampi), più dati o addestrando per più iterazioni.
- La performance di un LLM, in termini di loss, scala come una legge di potenza con ognuno di questi tre.

# Leggi di scaling

- Loss  $L$  in funzione del numero di parametri  $N$ , dimensione del dataset  $D$ , budget di compute  $C$  (se gli altri due sono tenuti costanti).
- Le leggi di scaling possono essere usate all'inizio del training per prevedere quale sarebbe la loss se si aggiungessero più dati o se si aumentasse la dimensione del modello.

$$L(N) = \left( \frac{N_c}{N} \right)^{\alpha_N}$$
$$L(D) = \left( \frac{D_c}{D} \right)^{\alpha_D}$$
$$L(C) = \left( \frac{C_c}{C} \right)^{\alpha_C}$$

# Numero di parametri non-embedding

$$\begin{aligned} N &\approx 2 d n_{\text{layer}} (2 d_{\text{attn}} + d_{\text{ff}}) \\ &\approx 12 n_{\text{layer}} d^2 \\ &\quad (\text{assuming } d_{\text{attn}} = d_{\text{ff}}/4 = d) \end{aligned}$$

Quindi GPT-3, con  $n=96$  layer e dimensionalità  $d=12288$ , ha  $12 \times 96 \times 150994944 \approx 175$  miliardi di parametri.

# KV Cache

- Nel training, possiamo calcolare l'attention in modo molto efficiente in parallelo:

$$\mathbf{A} = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$

- Ma *non in fase di inferenza*, dove i token vengono generati progressivamente uno per volta
- Per un nuovo *token*  $x$ , è necessario moltiplicare per  $\mathbf{W}^Q$ ,  $\mathbf{W}^K$ , and  $\mathbf{W}^V$  per ottenere query, key, values.

# KV Cache

- Non vogliamo ricalcolare i vettori *key* e *value* per tutti i *tokens* precedenti  $x_{<i}$ .
- Invece, archiviamo i vettori *key* e *value* in memoria nella KV cache, e poi possiamo semplicemente prenderli dalla cache.

# KV Cache

$$\begin{array}{c} Q \\ \left[ \begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} \right] \\ N \times d_k \end{array} \times \begin{array}{c} K^T \\ \left[ \begin{array}{c|c|c|c} \kappa & \bar{\kappa} & \bar{\kappa} & \bar{\kappa} \end{array} \right] \\ d_k \times N \end{array} = \begin{array}{c} QK^T \\ \left[ \begin{array}{cccc} q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 & q_1 \cdot k_4 \\ q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 & q_2 \cdot k_4 \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & q_3 \cdot k_4 \\ q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 \end{array} \right] \\ N \times N \end{array} \quad \left[ \begin{array}{c} V \\ \left[ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \right] \\ N \times d_V \end{array} \right] \times \begin{array}{c} A \\ \left[ \begin{array}{c} a_1 \\ a_2 \\ a_3 \\ a_4 \end{array} \right] \\ N \times d_V \end{array} =$$

# KV Cache

$$\begin{bmatrix} Q \\ \times \quad \begin{array}{|c|c|c|c|} \hline k_1 & k_2 & k_3 & \textcolor{blue}{k_4} \\ \hline \end{array} \quad = \quad QK^T \\ q4 \\ 1 \times d_k \end{bmatrix} \quad d_k \times N \quad \begin{bmatrix} q4 \cdot k_1 & q4 \cdot k_2 & q4 \cdot k_3 & q4 \cdot k_4 \\ \hline 1 \times N \end{bmatrix} \quad \begin{bmatrix} V \\ \times \quad \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline \end{array} \quad = \quad A \\ a4 \\ 1 \times d_v \end{bmatrix} \quad N \times d_v \end{array}$$

# Parameter-Efficient Fine Tuning

- L'adattamento a un nuovo dominio tramite *continued pretraining* (fine tuning) è un problema con i grandi LLM.
  - Notevole quantità di parametri da addestrare.
  - Per ogni batch, si deve fare un passaggio di gradient descent e fare backpropagation attraverso i layer di tutto il modello.
- Costi notevoli legati alla potenza di calcolo richiesta, memoria e tempo.

# Parameter-Efficient Fine Tuning

- Il modello per *la sola inferenza* utilizza circa un numero di GB pari a un po' più di 2 volte il numero di parametri.

LLaMA-2 7B → fino a 16GB

- # parametri in 16-bit (2 bytes) → per 7B parametri servono 14 GB
- Embedding models → 1GB
- Input + output al modello ca 1MB/token (512 token → 1GB)

# Parameter-Efficient Fine Tuning

- In fase di fine-tuning, non ho il solo modello in memoria

Finetuning di LLaMA-2 7B (*batch size* = 1)

- Parametri del modello (*float a 16 bit*) →  $7B \times 2 =$  14GB
- *Gradienti* (*float a 16 bit*) →  $7B \times 2 =$  14GB
- *Stati dell'ottimizzatore* (2× parametri *float a 32 bit*) →  $7B \times 4 \times 2 =.$  56GB

**TOTALE: 84GB**

# Parameter-Efficient Fine Tuning

- Le tecniche di Parameter-Efficient Fine Tuning (PEFT) consentono di:
  - Selezionare in modo efficiente un sottoinsieme di parametri da aggiornare durante il *fine-tuning*.
  - Congelare alcuni dei parametri.
  - Aggiornare solo alcuni pochi parametri.
- Generalmente si addestrano gli ultimi layer del modello

# LoRA (Low-Rank Adaptation)

- I Transformers hanno molti layer di moltiplicazione di matrici dense, come gli strati  $W^Q$ ,  $W^K$ ,  $W^V$ ,  $W^O$  nei layer di attenzione.
- Invece di aggiornare questi layer durante il fine-tuning:
  - Congeliamoli
  - Aggiorniamo una loro approssimazione a basso rango (Low-Rank) con meno parametri.

# LoRA (Low-Rank Adaptation)

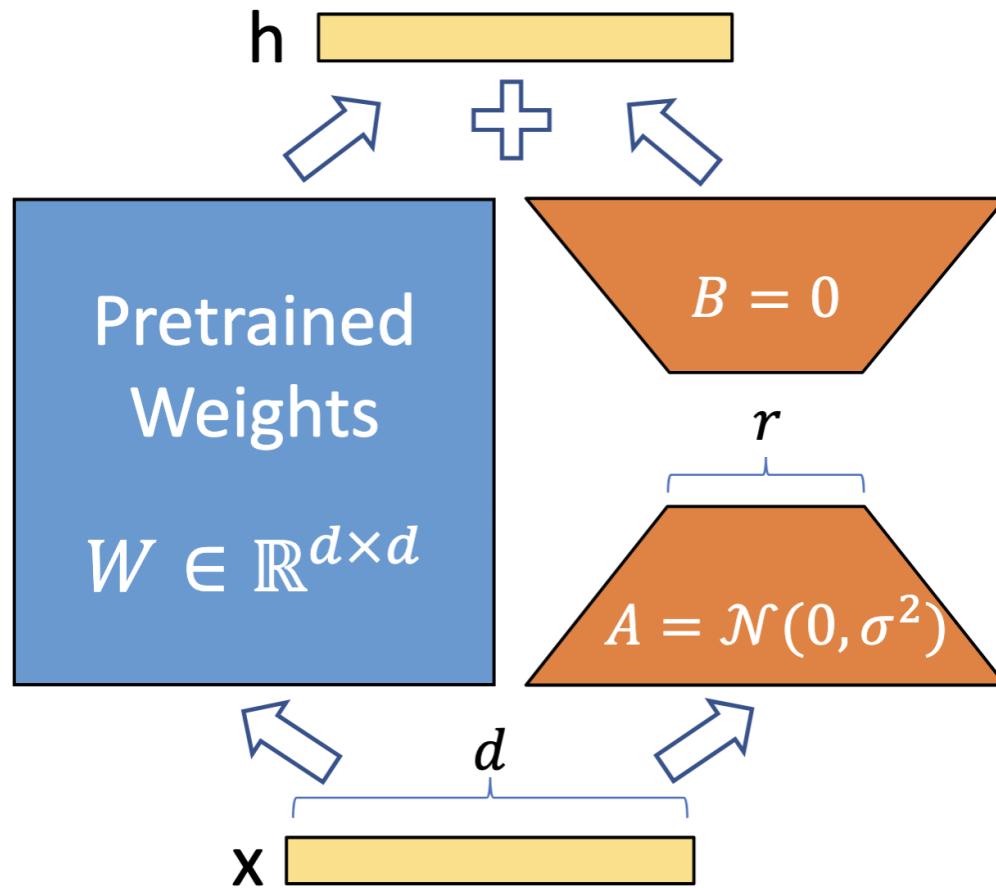
- Considerata una matrice  $\mathbf{W}$  [ $N \times d$ ] che deve essere aggiornata durante il fine-tuning tramite gradient descent.
- Normalmente gli aggiornamenti sono di tipo additivo  $\mathbf{W} \rightarrow \mathbf{W} + \Delta\mathbf{W}$ , dove  $\Delta\mathbf{W}$  ( $[N \times d]$ )
- Nel forward pass, dopo l'aggiornamento, calcoliamo  $\mathbf{h} = \mathbf{x}\mathbf{W}$

# LoRA (Low-Rank Adaptation)

- In LoRA, congeliamo  $\mathbf{W}$  e aggiorniamo invece una decomposizione  $\mathbf{AB}$  a *basso rango* di  $\mathbf{W}$  in cui:
  - $\mathbf{A}$  [ $N \times r$ ]
  - $\mathbf{B}$  [ $r \times d$ ], con  $r \ll \min(N, d)$
- Durante il fine-tuning aggiorniamo  $\mathbf{A}$  e  $\mathbf{B}$  invece di  $\mathbf{W}$ .
- Sostituiamo  $\mathbf{W} + \Delta\mathbf{W}$  con  $\mathbf{W} + \mathbf{AB}$ .
- Nel forward pass, invece di  $\mathbf{h} = \mathbf{xW}$  calcoliamo:

$$\mathbf{h} = \mathbf{xW} + \mathbf{xAB}$$

# LoRA (Low-Rank Adaptation)



# Quantizzazione

- I pesi di un modello sono memorizzati in float 32, li posso ridurre di dimensionalità in modo che occupino meno spazio?

## Comparing number formats

