

# Anatomia degli LLMs



## 4. Utilizzo dei modelli generativi



Alessio Miaschi

ItaliaNLP Lab, Istituto di Linguistica Computazionale (CNR-ILC), Pisa

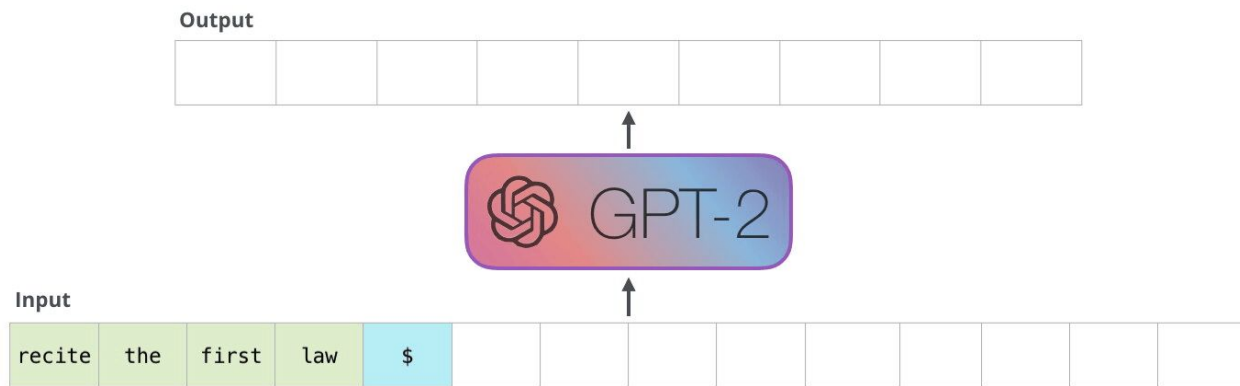
[alessio.miaschi@ilc.cnr.it](mailto:alessio.miaschi@ilc.cnr.it)

<https://alemmaschi.github.io/>

<http://www.italianlp.it/alessio-miaschi/>

# GPT (Radford et al, 2018), GPT-2 (Radford et al, 2019), etc

- Decoder Transformer model
- Addestrato per approssimare la funzione di **Language Modeling (LM)**
- Modello generativo → pensato per generare testo

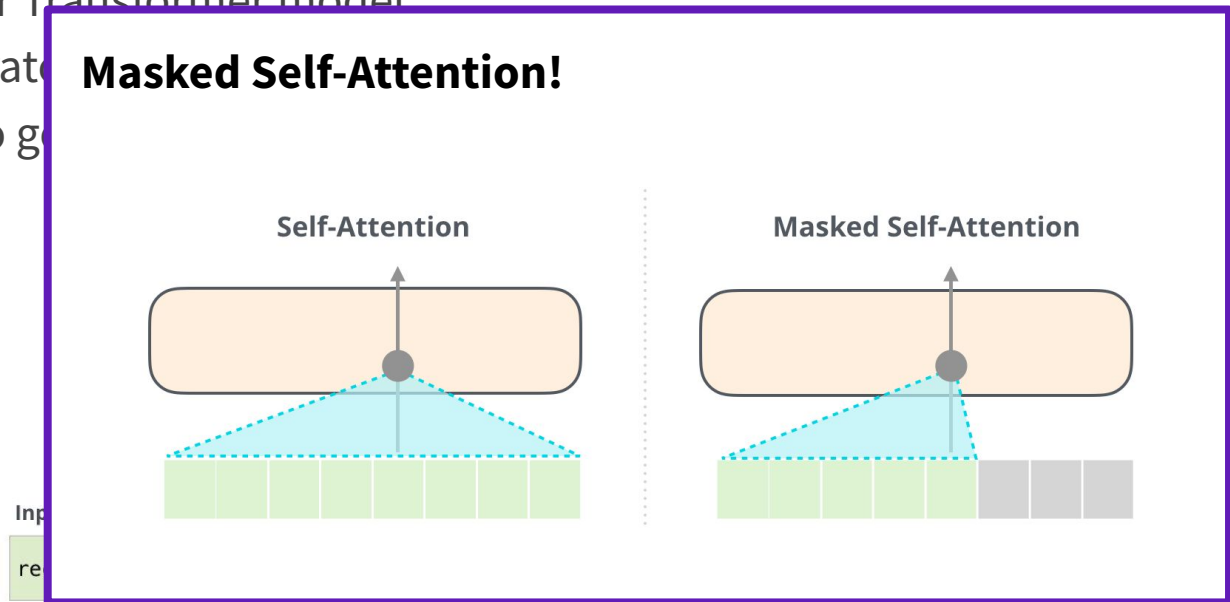


[Improving Language Understanding by Generative Pre-Training \(Radford et al., 2018\), https://openai.com/research/language-unsupervised](https://openai.com/research/language-unsupervised)

[Language Models are Unsupervised Multitask Learners \(Radford et al., 2019\), https://openai.com/research/better-language-models](https://openai.com/research/better-language-models)

# GPT (Radford et al, 2018), GPT-2 (Radford et al, 2019), etc

- Decoder Transformer model
- Addestrato
- Modello g



[Improving Language Understanding by Generative Pre-Training \(Radford et al., 2018\), https://openai.com/research/language-unsupervised](https://openai.com/research/language-unsupervised)

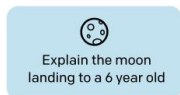
[Language Models are Unsupervised Multitask Learners \(Radford et al., 2019\), https://openai.com/research/better-language-models](https://openai.com/research/better-language-models)

# Instruction Tuning e RLHF: da GPT-3 a InstructGPT

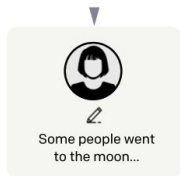
## Step 1

**Collect demonstration data, and train a supervised policy.**

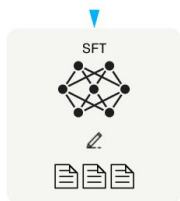
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



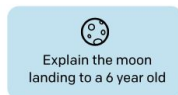
This data is used to fine-tune GPT-3 with supervised learning.



## Step 2

**Collect comparison data, and train a reward model.**

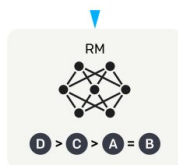
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



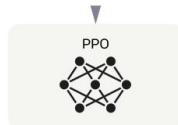
## Step 3

**Optimize a policy against the reward model using reinforcement learning.**

A new prompt is sampled from the dataset.



The policy generates an output.

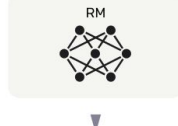


Once upon a time...

The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.

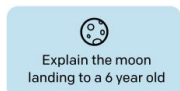


# Instruction Tuning e RLHF: da GPT-3 a InstructGPT

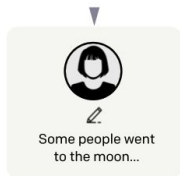
Step 1

**Collect demonstration data, and train a supervised policy.**

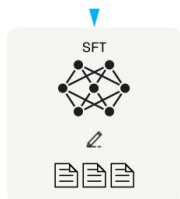
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



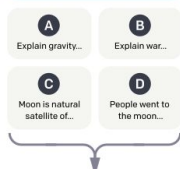
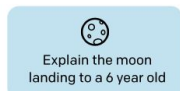
This data is used to fine-tune GPT-3 with supervised learning.



Step 2

**Collect comparison data, and train a reward model.**

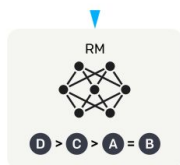
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



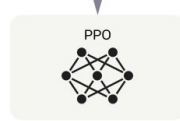
Step 3

**Optimize a policy against the reward model using reinforcement learning.**

A new prompt is sampled from the dataset.



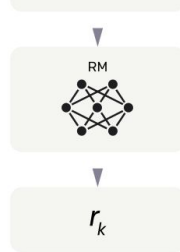
The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.

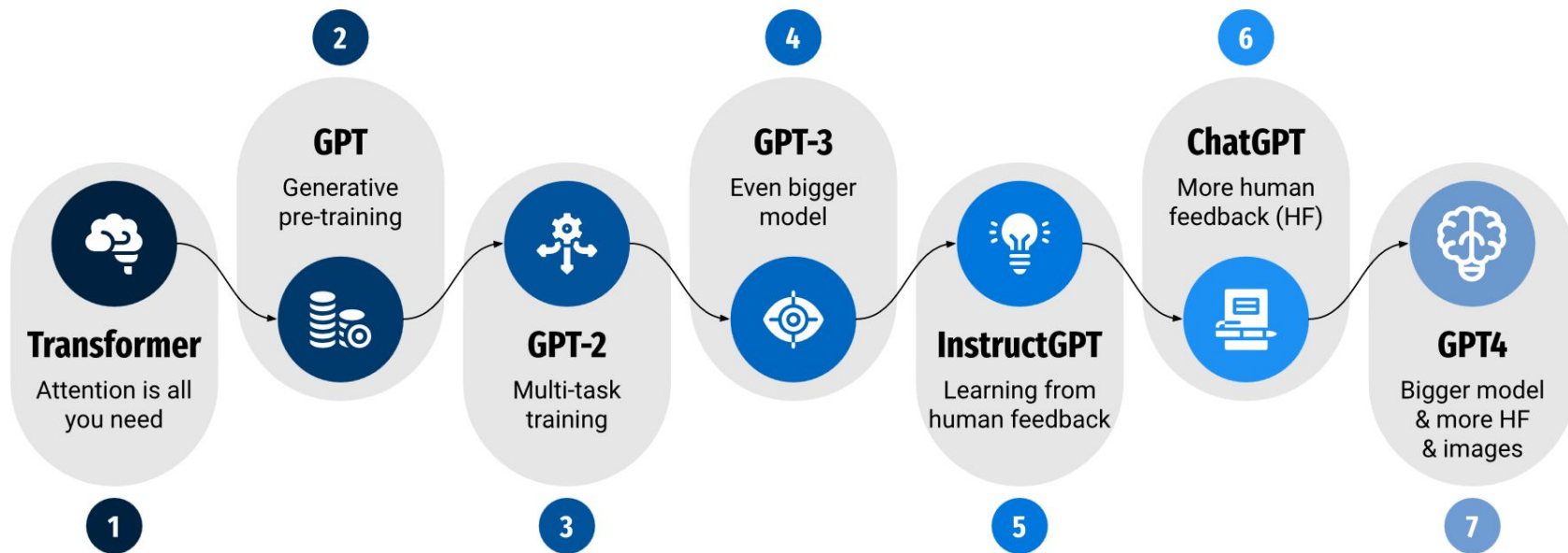


**Reinforcement Learning from Human Feedback (RLHF)**

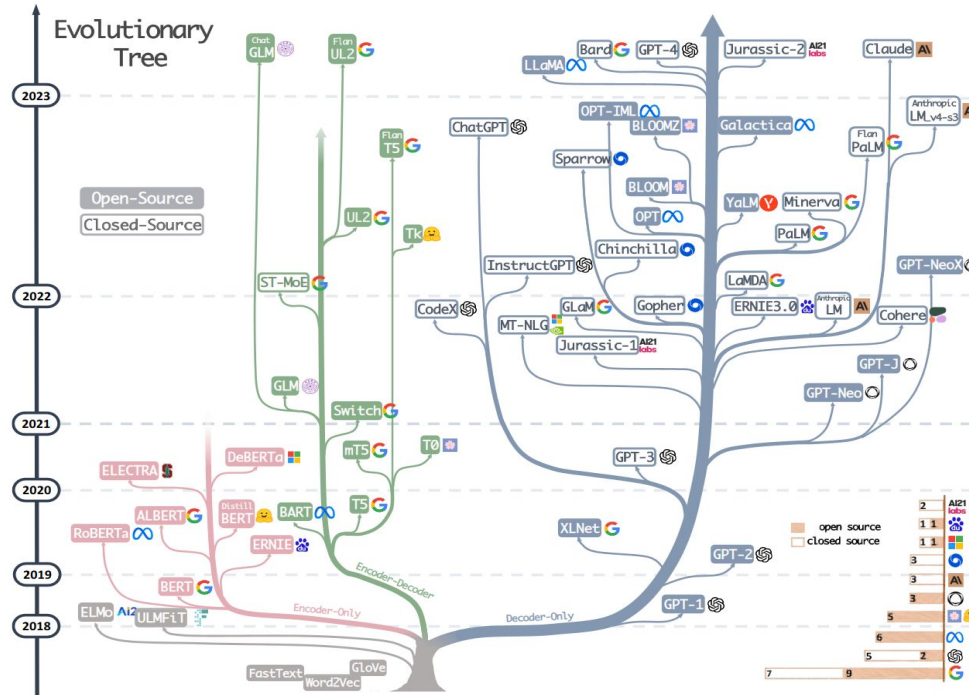
<https://huggingface.co/blog/rlhf>

# Dal Transformer a GPT4

## Evolution from Transformer architecture to ChatGPT



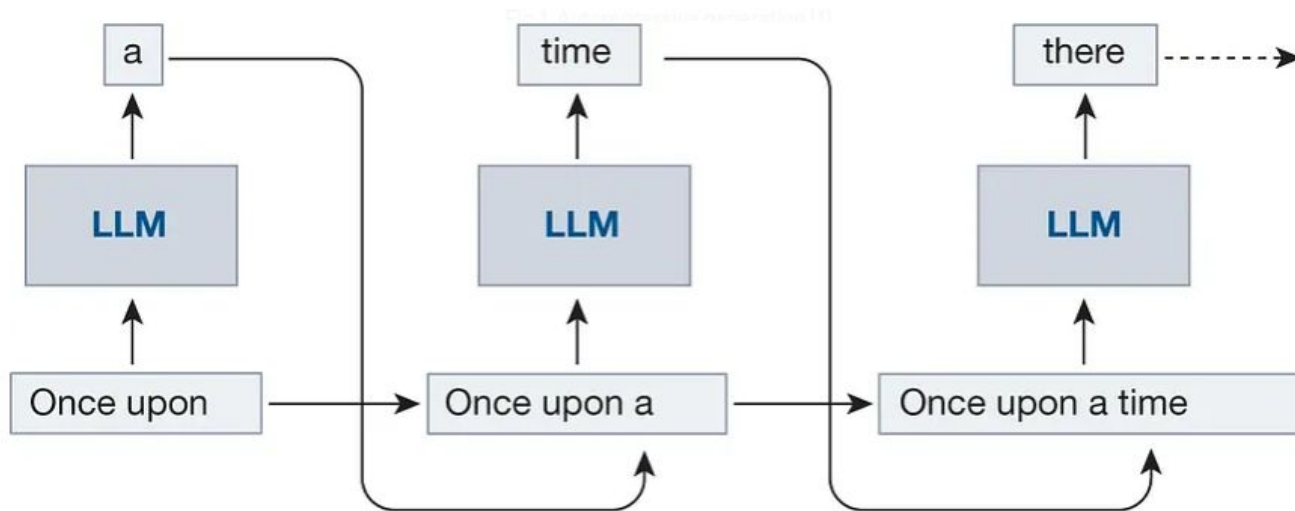
# “Evolutionary Tree” dei recenti NLMs



# Inferenza con un LLM



# Inferenza con un LLM



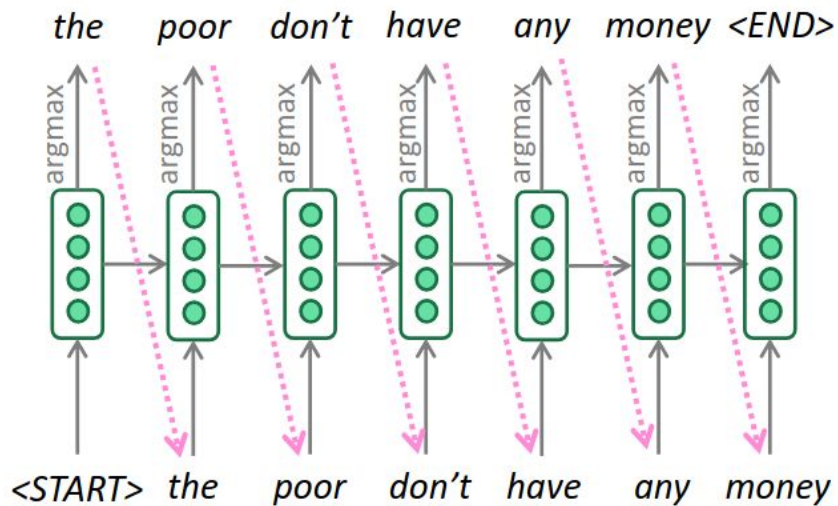
# Inferenza con un LLM

- Durante l'inferenza, il modello valuta continuamente il modo in cui i diversi token (i.e. subtoken) dell'input contribuiscono alla generazione dell'output
- Per determinare il token successivo, il modello produce i **logits** per ogni parola del vocabolario che verranno poi convertiti in probabilità (**softmax**) per determinare il token successivo più probabile
- Parametri per controllare il processo di inferenza:
  - **Temperature**: regola la casualità dell'output → valori di temperature più elevati ( $>1,0$ ) portano a risposte più varie e creative, mentre valori più bassi ( $<1,0$ ) producono output più mirati e deterministici
  - **Top-p (Nucleus) Sampling**: restringe la rosa dei candidati al più piccolo insieme di parole la cui probabilità cumulativa supera una soglia scelta (e.g. 90%)
  - **Top-k Filtering**: il modello limita le scelte alle prime k parole (i.e. token) più probabili

# Greedy e Beam Search

- Tra le varie strategie, la **greedy search** e la **beam search** sono due degli approcci più utilizzati per gestire l'inferenza di LLM
- **Greedy Search**: il modello sceglie sempre il token successivo con la probabilità più alta a ogni passo durante la fase di generazione
- **Beam Search**: tecnica che mantiene aperte contemporaneamente diverse sequenze, chiamate appunto “beams”

# Greedy Search



**Limite principale:** il modello non può tornare indietro e rivedere le decisioni precedenti

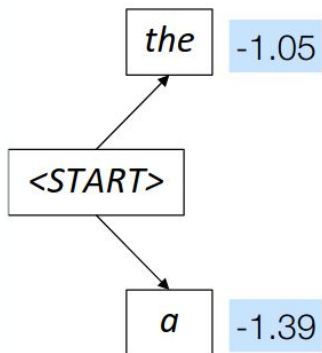
# Beam Search

Algoritmo della Beam Search:

1. L'algoritmo mantiene un insieme di sequenze parziali candidate (**beams**), tipicamente di dimensione fissa (es. 5–10)
2. Per ogni sequenza, vengono valutate le probabilità dei possibili token successivi
3. Ogni sequenza viene espansa con i token più probabili, generando nuove sequenze
4. Tra tutte le nuove sequenze generate, vengono selezionate le k migliori in base alla probabilità cumulativa
5. Il processo viene ripetuto fino al raggiungimento di una condizione di terminazione (e.g. lunghezza massima o token di fine).
6. Alla fine, viene scelta la sequenza completa con la probabilità più alta come output definitivo

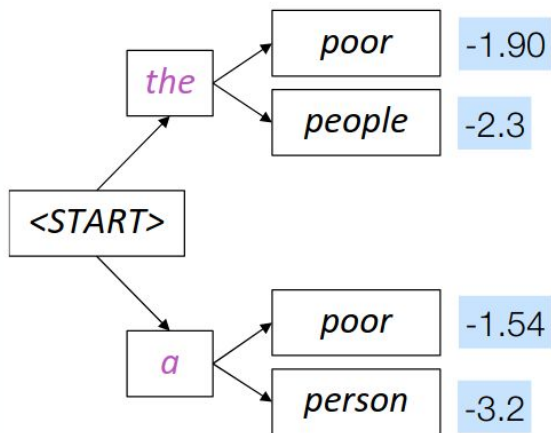
# Beam Search

Beam size = 2



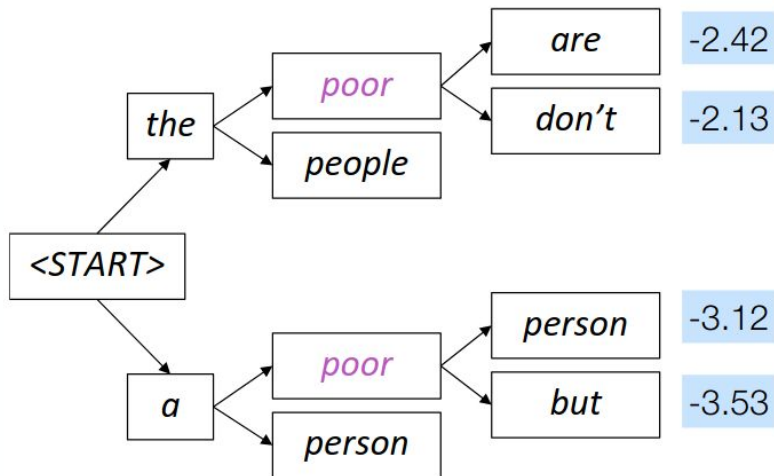
# Beam Search

Beam size = 2



# Beam Search

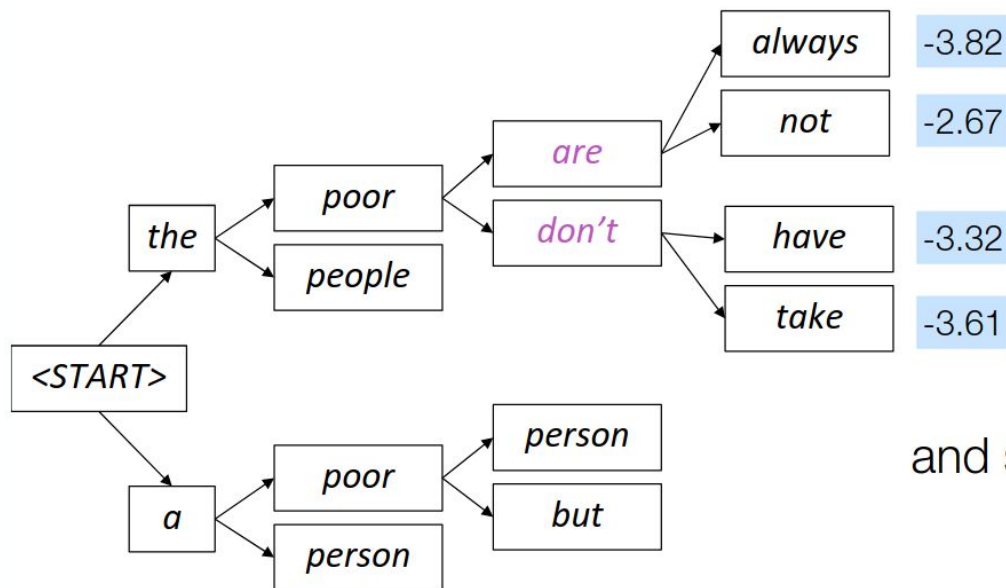
Beam size = 2





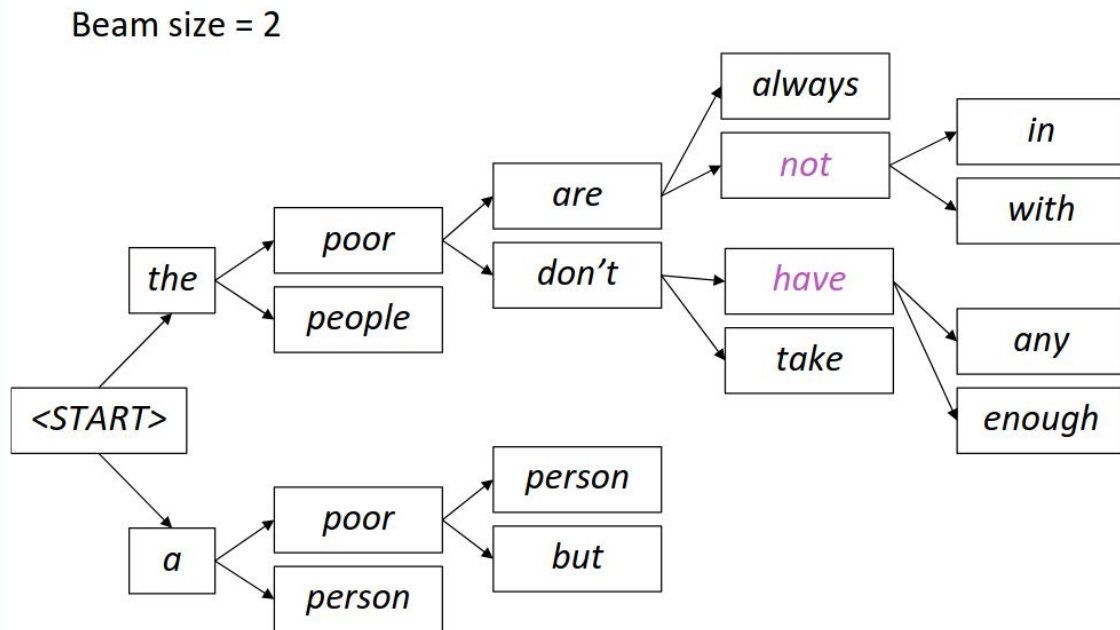
# Beam Search

Beam size = 2



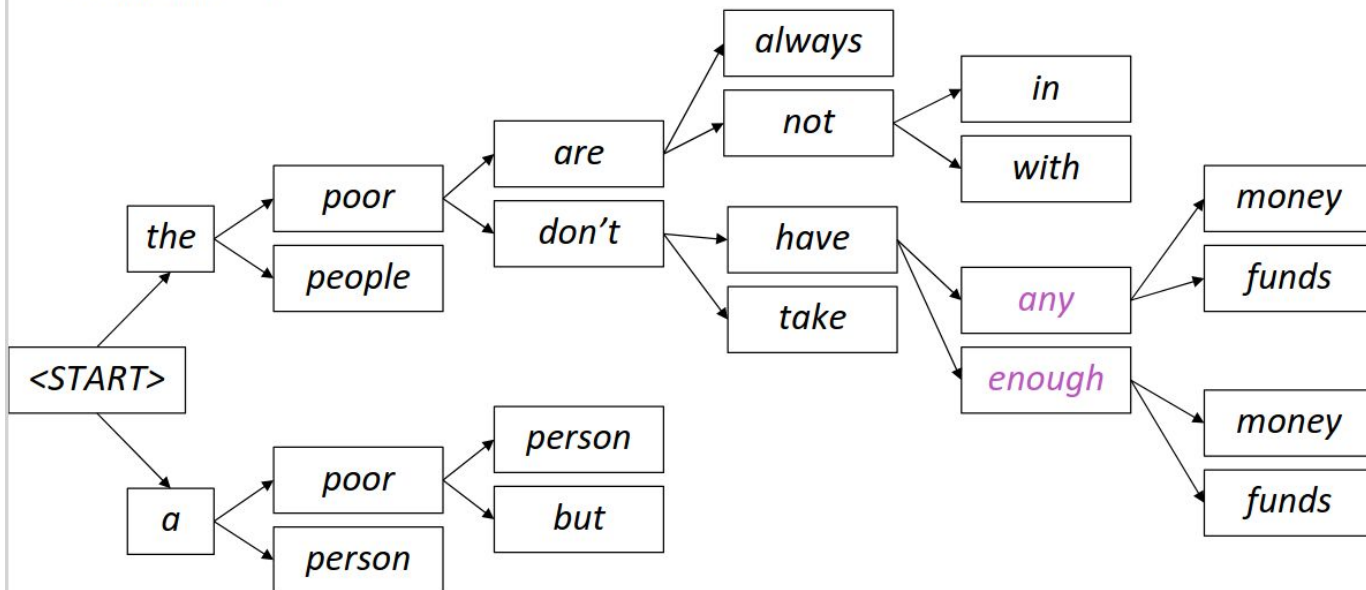
and so on...

# Beam Search



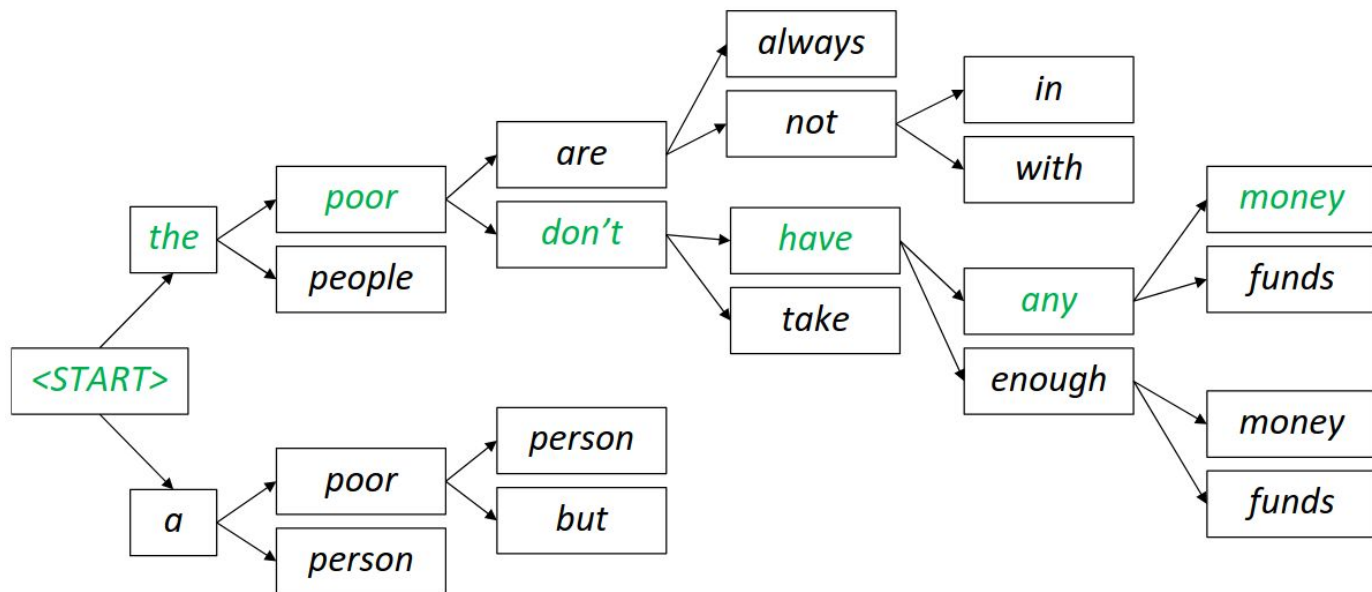
# Beam Search

Beam size = 2



# Beam Search

Beam size = 2



# Beam Search

Qual è l'effetto della modifica nel numero di beams (**k**)?

- Valori di k **bassi** (e.g.  $k=1$ ) ha lo stesso problema della greedy decoding:
  - output non grammaticali, innaturali, non plausibili
- Valori di k **più alti** riduce i problemi menzionati sopra, ma:
  - è più costoso computazionalmente
  - in open-ended task (e.g. chat), valori di k alti possono rendere l'output più generico

# Valutazione della generazione

# Valutazione della generazione

- Tradizionalmente, la metrica più comunemente utilizzata per la valutazione dell'output di un modello generativo è la **perplexità**
- La perplexità valuta l'incertezza di un modello nel predire il token successivo in una sequenza
- La perplexità affonda le sue radici nella teoria dell'informazione → **entropia**

$$H(P) = - \sum_i p(w_i) \log p(w_i)$$

- dove  $p(w_i)$  rappresenta la probabilità della parola  $w_i$  in una sequenza di input

# Cross-Entropy

- Mentre l'entropia misura l'incertezza media di una singola distribuzione di probabilità, la **cross-entropy** quantifica la differenza tra due distribuzioni di probabilità
- Nel caso di un LM, si tratta della distribuzione della sequenza originale,  $P$ , e della distribuzione generata dal modello,  $Q$

$$H(P, Q) = - \sum_i p(x_i) \log q(x_i)$$

- Valori di cross-entropy più bassi indicano una maggiore certezza da parte del modello, in quanto implicano che le probabilità sono più strettamente allineate con la reale distribuzione



# Perplexità

- La perplexità è definita in questo modo:

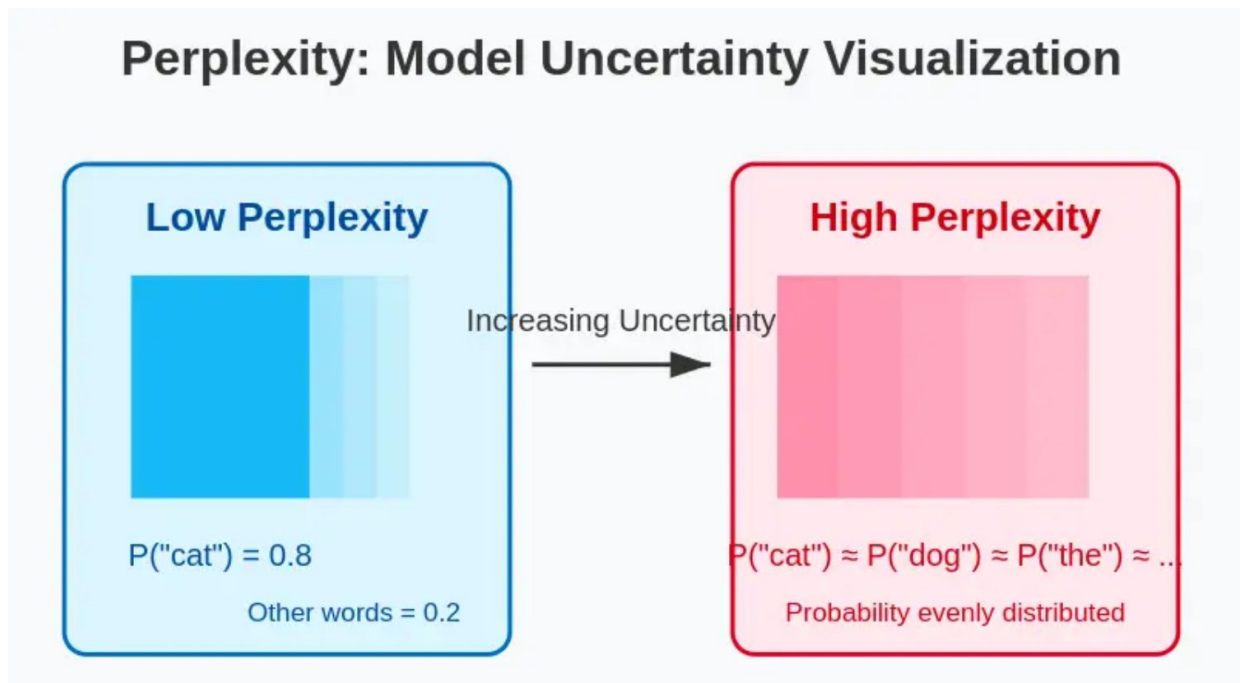
$$\text{Perplexity} = e^{H(P,Q)} = e^{-\frac{1}{N} \sum_{i=1}^N \log q(w_i)}$$

- La perplexità non è altro che la cross-entropy con l'ulteriore applicazione della funzione esponenziale
- Poiché la cross-entropy è una misura logaritmica negativa, applicarne l'esponenziale equivale a invertire il logaritmo, riportando il valore nello spazio delle probabilità
- Questo risultato indica quante opzioni diverse il modello considera effettivamente a ogni passo

# Perplexità



# Perplexità



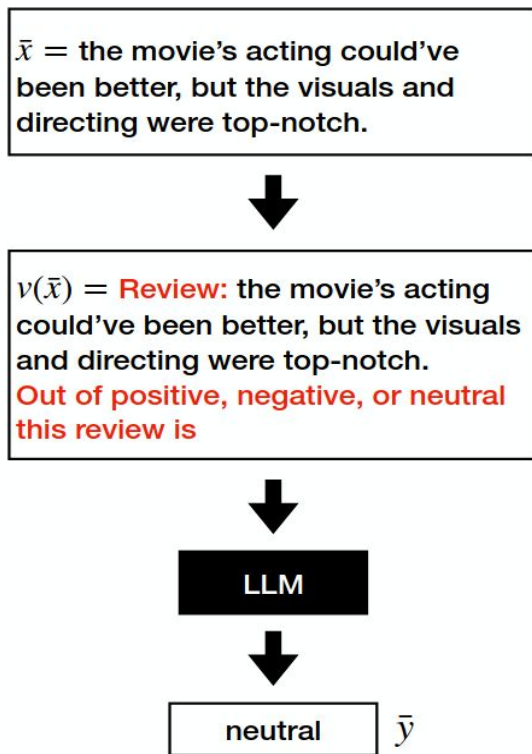
# LLM Prompting

# LLM Prompting

- Una volta addestrato un LLM, è sempre possibile applicare una fase di fine-tuning per addestrarlo alla risoluzione di uno (o più) task specifici:
  - Stesso approccio di fine-tuning con Encoder-based models: addestramento del modello a partire da un set di dati annotati
- Oltre a questo approccio, i più recenti (e grandi) LLM offrono un'ulteriore modalità che non richiede alcuna modifica dei suoi parametri: il **prompting**
  - Con o senza dati annotati: **zero-shot** o **in-context learning (few-shot)**
  - con o senza catene di ragionamento intermedie: **chain-of-thought** prompting

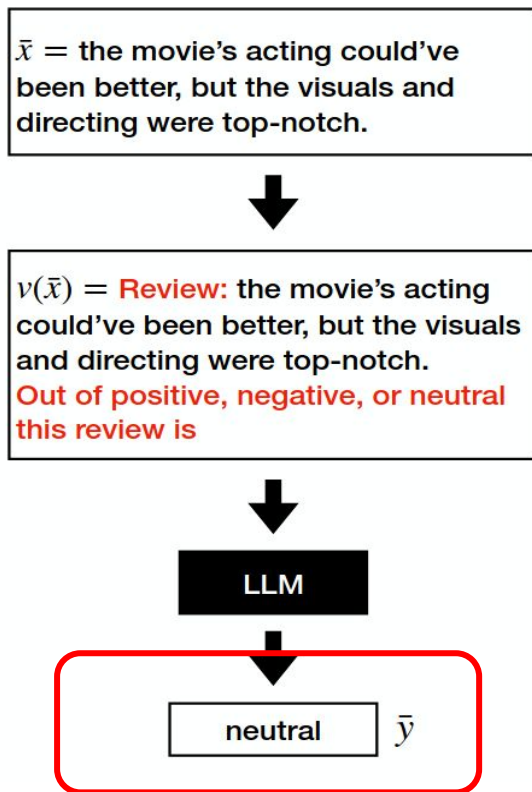
# Zero-shot Prompting

- Input: singolo esempio non annotato
- Output: label da predire
- Task: qualsiasi NLP task (i.e. classificazione, summarization, etc.)
- Pre-processing: racchiudere il testo di input in un template (**verbalizer**)



# Zero-shot Prompting

- Input: singolo esempio non annotato
- Output: label da predire
- Task: qualsiasi NLP task (i.e. classificazione, summarization, etc.)
- Pre-processing: racchiudere il testo di input in un template (**verbalizer**)



# Zero-shot Prompting

- Input: singolo esempio non annotato
- Output: label d
- Task: qualsiasi classificazione,
- Pre-processing: racchiudere il testo di input in un template (**verbalizer**)

Cosa fare nel caso in cui l'output del modello non si adatta al formato previsto, anche se è semanticamente corretto?

$\bar{x}$  = the movie's acting could've been better, but the visuals and directing were top-notch.

ie's acting  
ut the visuals  
notch.  
e, or neutral

LLM

neutral

$\bar{y}$



# Zero-shot Prompting

- Input: singolo esempio non annotato

$\bar{x}$  = the movie's acting could've been better, but the visuals and directing were top-notch.

- Output: label d

Cosa fare nel caso in cui l'output del modello non si adatta al formato previsto, anche se è semanticamente corretto?

- Task: qualsiasi classificazione,

$$\arg \max_{\bar{y} \in \{1,2,3,4\}} p(\bar{y} | v(\bar{x}))$$

ie's acting  
ut the visuals  
notch.  
e, or neutral

- Pre-processing: racchiudere il testo di input in un template (**verbalizer**)

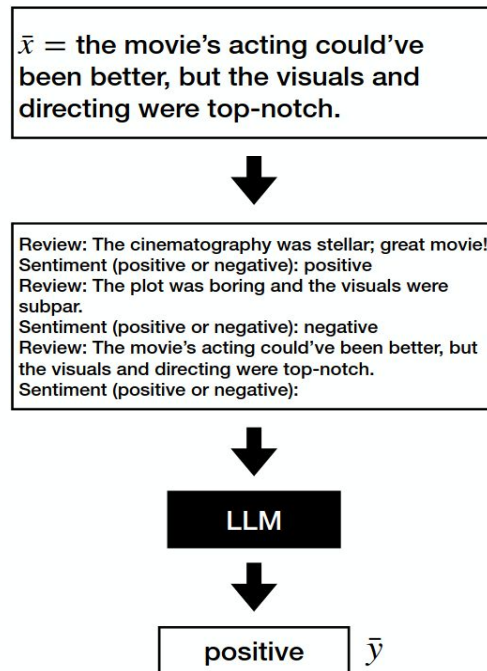
LLM

neutral

$\bar{y}$

# In-context learning (ICL) (Few-shot Prompting)

- Si prende un sample di training examples già annotati e li si converte tramite il verbalizer
- Si concatenano alla sequenza di input della quale vogliamo ottenere la risposta da parte del modello



# Chain-of-thought (CoT) Prompting

- Task più complessi richiedono diverse fasi di ragionamento
- **CoT Prompting** (Wei et al., 2022): mostrare al modello gli step di ragionamento nella fase di in-context learning
- Gli step di ragionamento possono essere dedotti dal modello anche senza l'accesso a degli esempi risolti

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. ❌

## Chain-of-Thought Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✅

Paper CoT:

[https://openreview.net/pdf?id=VjQlMeSB\\_J](https://openreview.net/pdf?id=VjQlMeSB_J)

# Chain-of-thought (CoT) Prompting

- Task più complessi richiedono diverse fasi di ragionamento
- **CoT Prompting** (Wei et al., 2022): mostrare al modello gli step di ragionamento nella fase di in-context learning
- **Gli step di ragionamento possono essere dedotti dal modello anche senza l'accesso a degli esempi risolti**
  - “Let’s think step by step”

## (b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are  $16 / 2 = 8$  golf balls. Half of the golf balls are blue. So there are  $8 / 2 = 4$  blue golf balls. The answer is 4. ✓

## (d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let’s think step by step.**

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Large Language Models are Zero-Shot Reasoners:  
<https://openreview.net/pdf?id=e2TBb5y0yFf>

# Fine-tuning di un LLM

# Fine-tuning di un LLM

Come si fa a passare da un modello che predire il token successivo in una sequenza di input




Il gatto insegue il topo nel \_\_\_\_

# Fine-tuning di un LLM

Come si fa a passare da un modello che predire il token successivo in una sequenza di input

Il gatto insegue il topo nel \_\_\_\_

A questo?

ChatGPT		
 Examples	 Capabilities	 Limitations
"Explain quantum computing in simple terms"	Remembers what user said earlier in the conversation	May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?"	Allows user to provide follow-up corrections	May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?"	Trained to decline inappropriate requests	Limited knowledge of world and events after 2021

# Instruction Tuning

## Model input (Disambiguation QA)

Q: In the following sentences, explain the antecedent of the pronoun (which thing the pronoun refers to), or state that it is ambiguous.

Sentence: The reporter and the chef will discuss their favorite dishes.

Options:

- (A) They will discuss the reporter's favorite dishes
- (B) They will discuss the chef's favorite dishes
- (C) Ambiguous

A: Let's think step by step.

## Before instruction finetuning

The reporter and the chef will discuss their favorite dishes.

The reporter and the chef will discuss the reporter's favorite dishes.

The reporter and the chef will discuss the chef's favorite dishes.

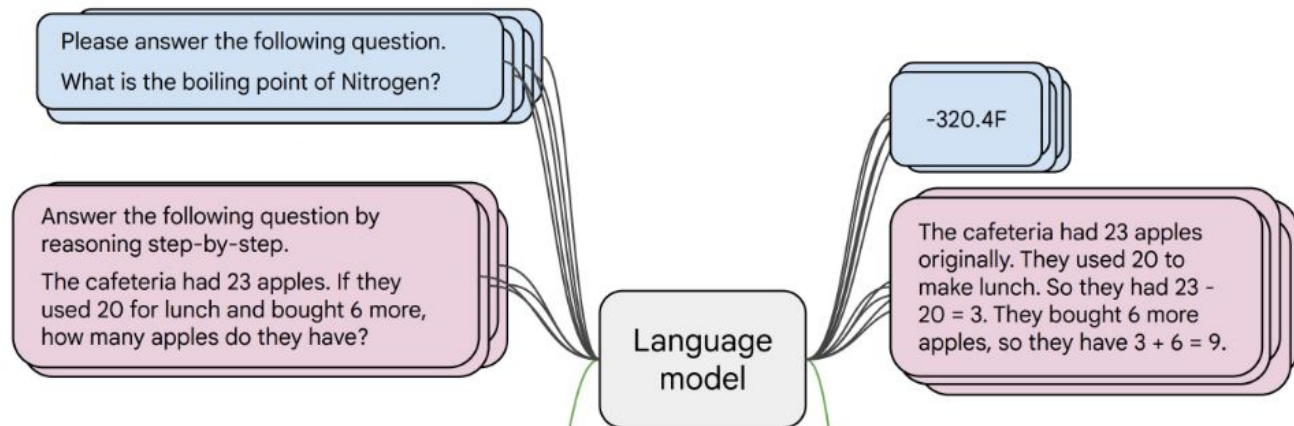
The reporter and the chef will discuss the reporter's and the chef's favorite dishes.

✗ (doesn't answer question)



# Instruction Tuning

- **Collect examples** of (instruction, output) pairs across many tasks and finetune an LM



- **Evaluate on unseen tasks**



# Instruction Tuning

## Model input (Disambiguation QA)

Q: In the following sentences, explain the antecedent of the pronoun (which thing the pronoun refers to), or state that it is ambiguous.


Sentence: The reporter and the chef will discuss their favorite dishes.

Options:

- (A) They will discuss the reporter's favorite dishes
- (B) They will discuss the chef's favorite dishes
- (C) Ambiguous

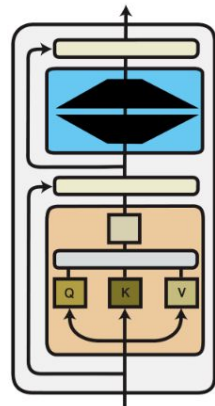
A: Let's think step by step.

## After instruction finetuning

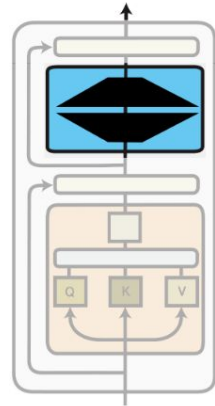
The reporter and the chef will discuss their favorite dishes does not indicate whose favorite dishes they will discuss. So, the answer is (C). 

# Parameter-Efficient Fine-tuning (PEFT)

- Il fine-tuning completo di un LLM (i.e. aggiornamento di tutti i parametri) è un processo particolarmente costoso
- È stato dimostrato che i recenti LLM sono eccessivamente parametrizzati



Full Fine-tuning  
Update **all model**  
**parameters**



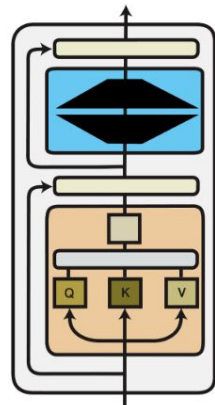
Parameter-efficient Fine-tuning  
Update a **small subset** of model  
parameters

# Parameter-Efficient Fine-tuning (PEFT)

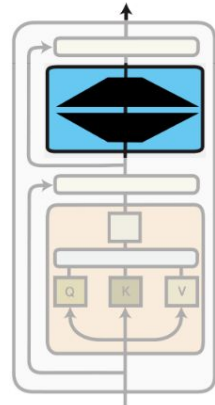
- Il fine-tuning completo di un LLM (i.e. aggiornamento di tutti i parametri) è un processo particolarmente costoso
- È stato dimostrato che i recenti LLM sono eccessivamente parametrizzati



- **Parameter-Efficient Fine-Tuning (PEFT):** fine-tuning di un set limitato di parametri



Full Fine-tuning  
Update **all** model  
parameters



Parameter-efficient Fine-tuning  
Update a **small subset** of model  
parameters

# Parameter-Efficient Fine-tuning (PEFT)

- Supponiamo di avere un LLM pre-addestrato  $P_{\phi}(y|x)$ 
  - E.g. GPT, LLaMA, Mistral
- Abbiamo un dataset di fine-tuning costituito da coppie *context-target*  $\{(x_i, y_i)\}_{i=1, \dots, N}$
- Durante il full fine-tuning, si aggiornano i parametri del modello
- Nel full fine-tuning, aggiorniamo i parametri del modello da  $\phi_0$  a  $\phi_0 + \Delta\phi$ , addestrandolo tramite la funzione di Language Modeling (LM):

$$\max_{\phi} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_{\phi}(y_t|x, y_{<t}))$$

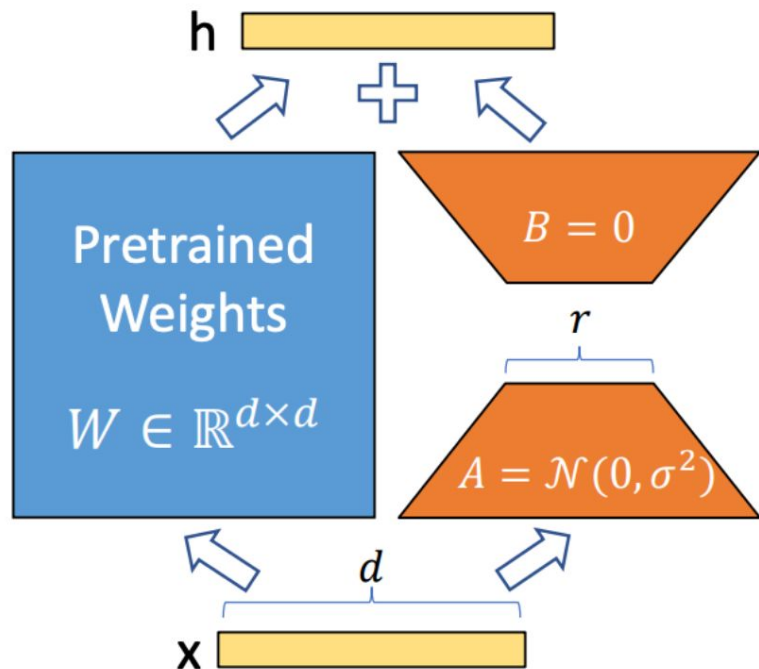
# LoRA (Low Rank Adaptation)

- Normalmente, per ogni downstream task abbiamo che:
  - $|\phi_0| = |\Delta\phi|$
- LoRA: apprendere i parametri specifici del downstream task  $\Delta\phi$  sfruttando solamente un insieme di parametri più piccolo  $\Theta$ :
  - $|\Theta| \ll |\phi_0|$

$$\max_{\Theta} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_{\phi_0 + \Delta\phi(\Theta)}(y_t | x, y_{<t}))$$

# LoRA (Low Rank Adaptation)

- Data  $W^0 \in \mathbb{R}^{d \times k}$
- Aggiorniamo la matrice dei pesi del modello con una *low-rank decomposition*
  - $W^0 + \Delta W = W^0 + BA$
  - dove  $\mathbf{B} \in \mathbb{R}^{d \times r}$ ,  $\mathbf{A} \in \mathbb{R}^{r \times k}$ ,  $r \ll \min(d, k)$
- Solo  $\mathbf{A}$  e  $\mathbf{B}$  vengono inizializzati casualmente e addestrati, il resto del modello (compreso  $\mathbf{W}$ ) resta fisso



# LoRA (Low Rank Adaptation)

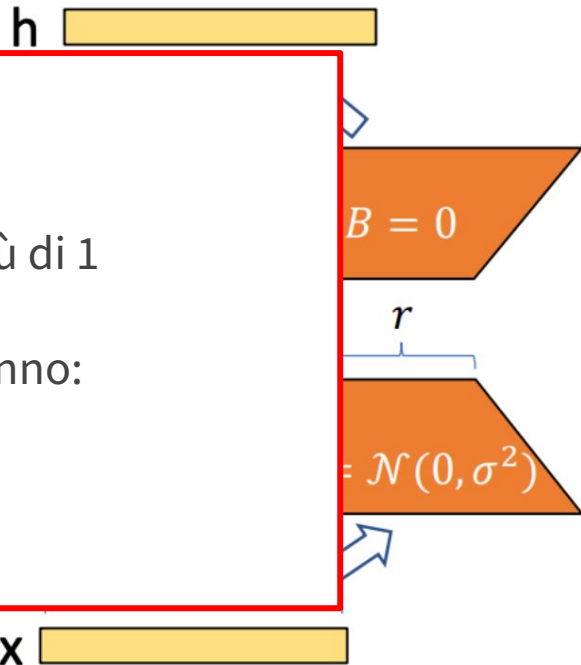
- Data  $W^0 \in \mathbb{R}^{d \times k}$

- Aggiorniamo il modello con:
  - $W^0 + \Delta W$
  - dove  $\Delta W$  è la matrice di adattamento

- Solo **A** e **B** sono addebi-  
casualme-  
modello (compreso **W**) resta fisso

## Esempio pratico:

- Se  $W \in \mathbb{R}^{1024 \times 1024}$ , il modello ha più di 1 milione di parametri
- Usando LoRa con e.g.  $r = 8$ , si avranno:
  - $A \in \mathbb{R}^{1024 \times 8}$
  - $B \in \mathbb{R}^{8 \times 1024}$





# LoRA (Low Rank Adaptation)

- Data  $W^0 \in \mathbb{R}^{d \times k}$

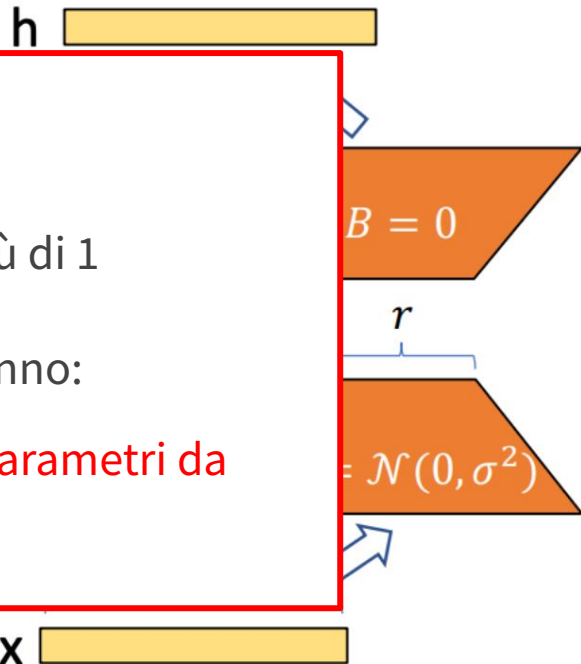
- Aggiorniamo il modello con:
  - $W^0 + \Delta W$
  - dove  $\Delta W$  è la matrice di adattamento

- Solo **A** e **B** sono addebi-  
casualme-  
modello (compreso **W**) resta fisso

## Esempio pratico:

- Se  $W \in \mathbb{R}^{1024 \times 1024}$ , il modello ha più di 1 milione di parametri
- Usando LoRa con e.g.  $r = 8$ , si avranno:
  - $A \in \mathbb{R}^{1024 \times 8}$
  - $B \in \mathbb{R}^{8 \times 1024}$

Solo 16.384 parametri da allenare!



# Reinforcement Learning e Preference Optimization

# Limiti dell'Instruction Tuning

- **Problema 1:** la raccolta di dati annotati è particolarmente costosa
- **Problema 2:** i task di generazione “open-ended” non sempre hanno una risposta giusta
- **Problema 3:** sebbene il modello venga addestrato, non lo si sta esplicitamente avvicinando alle preferenze umane

# Reinforcement Learning from Human Feedback (RLHF)

- **Task:** LLM da addestrare sul task di text summarization
- Per ogni sample del modello  $s$ , immaginiamo di avere un “human reward”:  $R(s) \in \mathbb{R}$

SAN FRANCISCO,  
California (CNN) --  
A magnitude 4.2  
earthquake shook the  
San Francisco  
...  
overturn unstable  
objects.

An earthquake hit  
San Francisco.  
There was minor  
property damage,  
but no injuries.

$$\begin{matrix} s_1 \\ R(s_1) = 8.0 \end{matrix}$$

The Bay Area has  
good weather but is  
prone to  
earthquakes and  
wildfires.

$$\begin{matrix} s_2 \\ R(s_2) = 1.2 \end{matrix}$$

- Il nostro task è massimizzare l'expected reward dai samples del nostro LLM:

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)}[R(\hat{s})]$$

# Reinforcement Learning from Human Feedback (RLHF)

- Il **RHLF** è un approccio di reinforcement learning che permette al modello di allineare le sue risposte alle preferenze degli esseri umani
- Step del RLHF:
  1. Abbiamo un pre-trained (o instruction-tuned) LLM  $p^{pt}(s)$
  2. Si mostrano a degli esseri umani alcune generazioni del modello e si raccolgono le preferenze (es. “la risposta A è più corretta/adatta di B”)
  3. Si crea un reward model  $RM_{\phi}(s)$  addestrato sulle preferenze espresse dagli esseri umani
  4. Addestramento del modello tramite reinforcement learning → algoritmo **PPO (Proximal Policy Optimization)**

## Reinforcement Learning from Human Feedback (RLHF)

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})] \approx \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta} \log p_{\theta}(s_i)$$

# Reinforcement Learning from Human Feedback (RLHF)

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})] \approx \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta} \log p_{\theta}(s_i)$$

che tradotto:

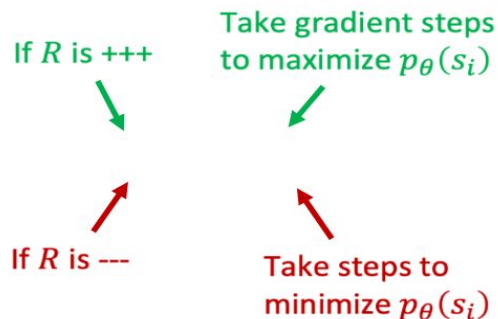
Per migliorare il nostro modello, aggiorniamo i parametri in base a quanto la risposta  $s_i$  è buona (cioè, ha reward alto) e a quanto il modello era propenso a generarla (log-probability).

# Reinforcement Learning from Human Feedback (RLHF)

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})] \approx \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta} \log p_{\theta}(s_i)$$

che tradotto:

Per migliorare il nostro modello, aggiorniamo i parametri in base a quanto la risposta  $s_i$  è buona (cioè, ha reward alto) e a quanto il modello era propenso a generarla (log-probability).



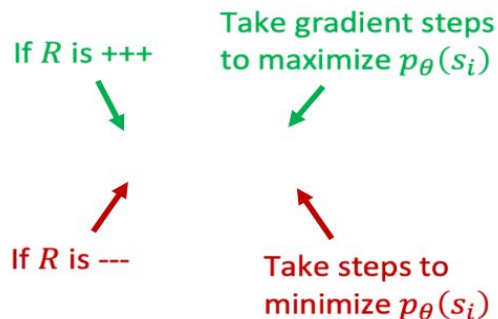


# Reinforcement Learning from Human Feedback (RLHF)

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})] \approx \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta} \log p_{\theta}(s_i)$$

che tradotto:

Per migliorare il nostro modello, aggiorniamo i parametri in base a quanto la risposta  $s_i$  è buona (cioè, ha reward alto) e a quanto il modello era propenso a generarla (log-probability).



# Reinforcement Learning from Human Feedback (RLHF)

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})] \approx \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta} \log p_{\theta}(s_i)$$

che tradotto:

Per migliorare  
è buona (cioè  
(log-probabil

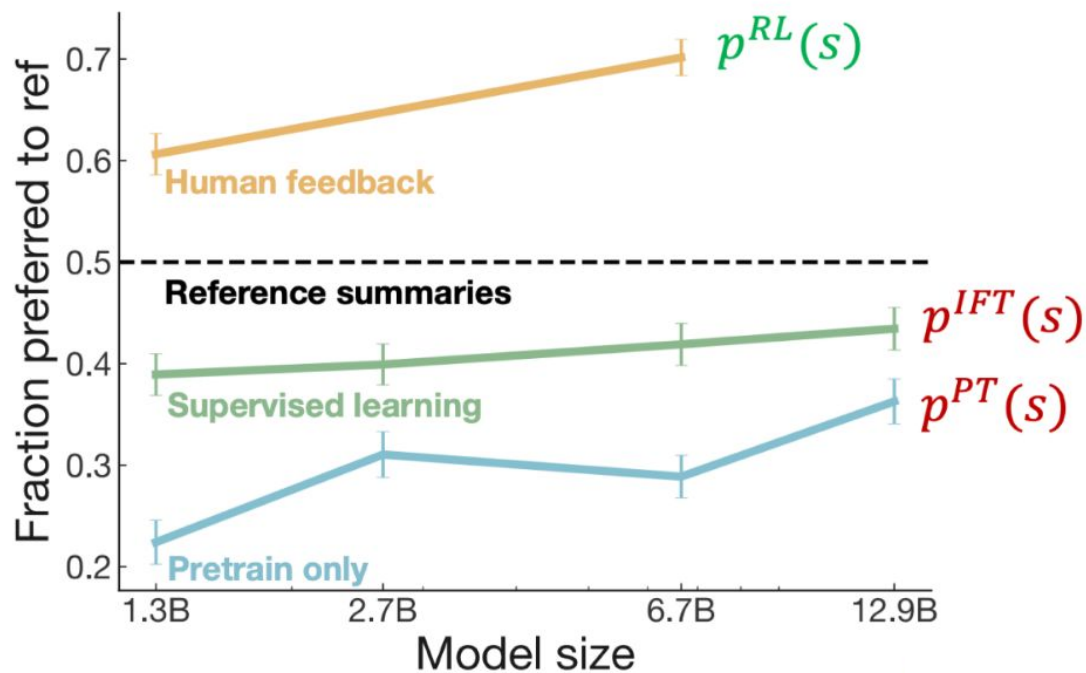
$$R(s) = RM_{\phi}(s) - \beta \log \left( \frac{p_{\theta}^{RL}(s)}{p^{PT}(s)} \right)$$

$RM_{\phi}(s)$  reward predetto dal reward model

$-\beta \log \left( \frac{p_{\theta}^{RL}(s)}{p^{PT}(s)} \right)$  penalizza deviazioni troppo grandi dal  
comportamento originale del modello  
preaddestrato

risposta  $s_i$   
a

# Reinforcement Learning from Human Feedback (RLHF)

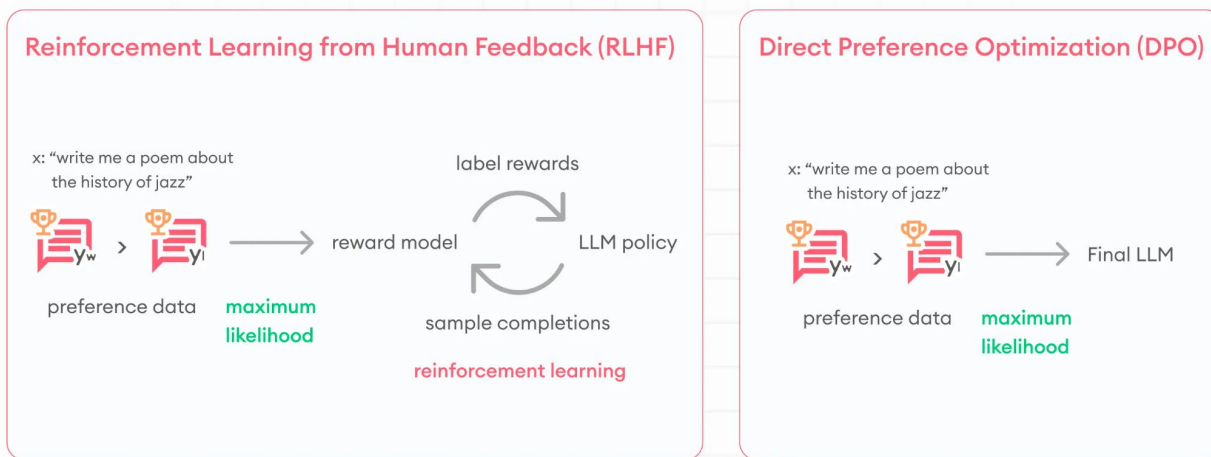


Learning to summarize from human feedback:

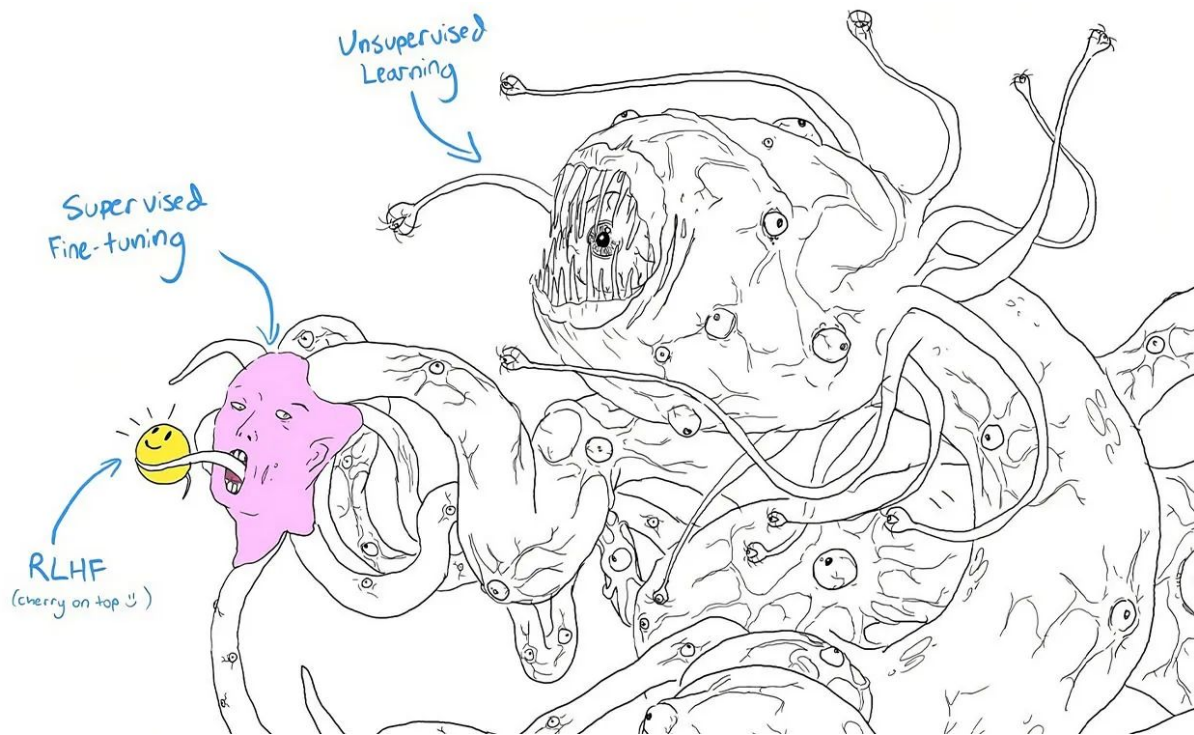
<https://proceedings.neurips.cc/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf>

# RLHF, DPO, ...

- Dopo il RLHF, sono stati sviluppati diversi metodi di RL per poter allineare e controllare il comportamento degli LLM
- **DPO (Direct Preference Optimization)** (Rafailov R. et al., 2023)



# Pre-training + Instruction Tuning + Reinforcement Learning



# Lezione Pratica

