Training Large Language Models : from Supervised Fine-Tuning to Reinforcement Learning

Pierre Clavier

Large Scale Machine Learning

Ecole des Mines de Paris

5 of March, 2026

Some Images from textbook are taken from [here]

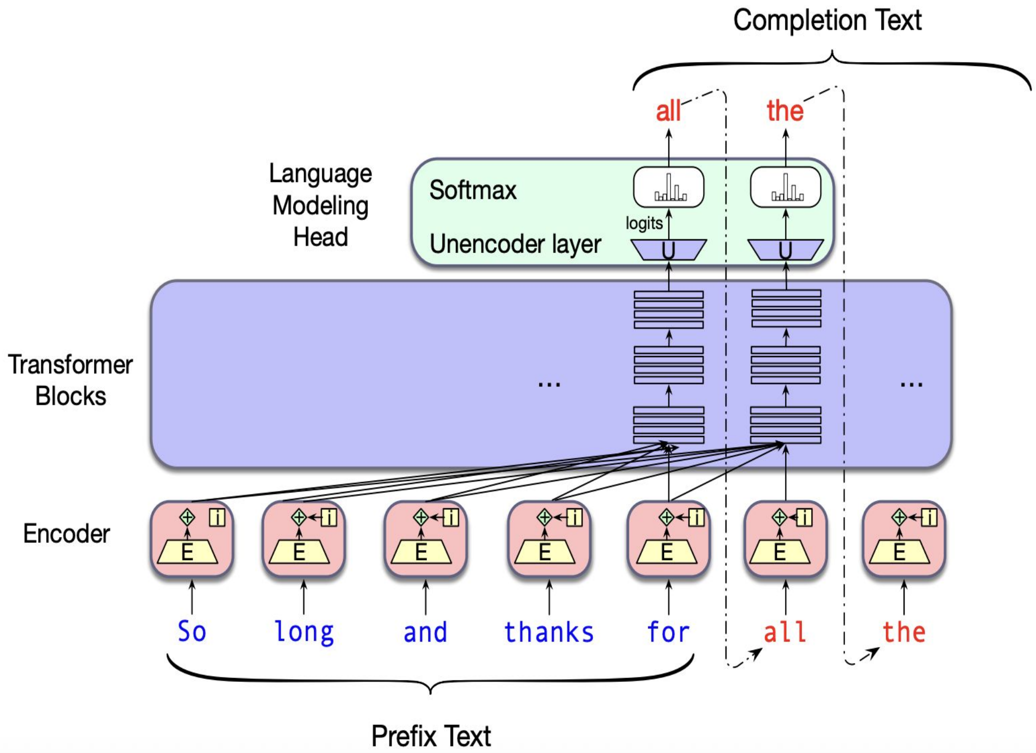
I. **Large Language Model : architecture and sampling**
   A. What is an Large Language Model ?
   B. Attention Mechanism
   C. LLM architecture
      Embedding, Positional embedding Layer norm, Residual Connection
   D. LLM Sampling/Decoding
      1. Top p, temperature scaling, Modern Sampling engine : vLLM
   E. Memory Optimization and Computation optimization
      1. KV cache trick
      2. Mixture of experts (MoEs)
      3. Sharding of Transformer weights ? Data Parallelism example

II. **Training LLMs, losses, optimization : from SFT to Reinforcement Learning**
   A. How to Train an LLM ?
   B. Very small note on Modern Optimization for Large Models
   C. Imitation Learning vs Preference Learning vs Reinforcement Learning
   D. Offline Learning
      1. Pretraining and Supervised Fine-Tuning : the cross entropy loss
      2. Offline Preference Learning : Direct Preference Optimization (DPO)
   E. Reinforcement Learning : from offline to online learning
      1. KL regularised Policy Gradient
      2. Variance Reduction and Leave-One-Out baseline
      3. Contrastive Policy Gradient and convergence in offline setting
      4. Stabilizing online RL with Group Relative Policy Optimization (GRPO)

I. Large Language Model : architecture and sampling

A. What is an Large Language Model ?

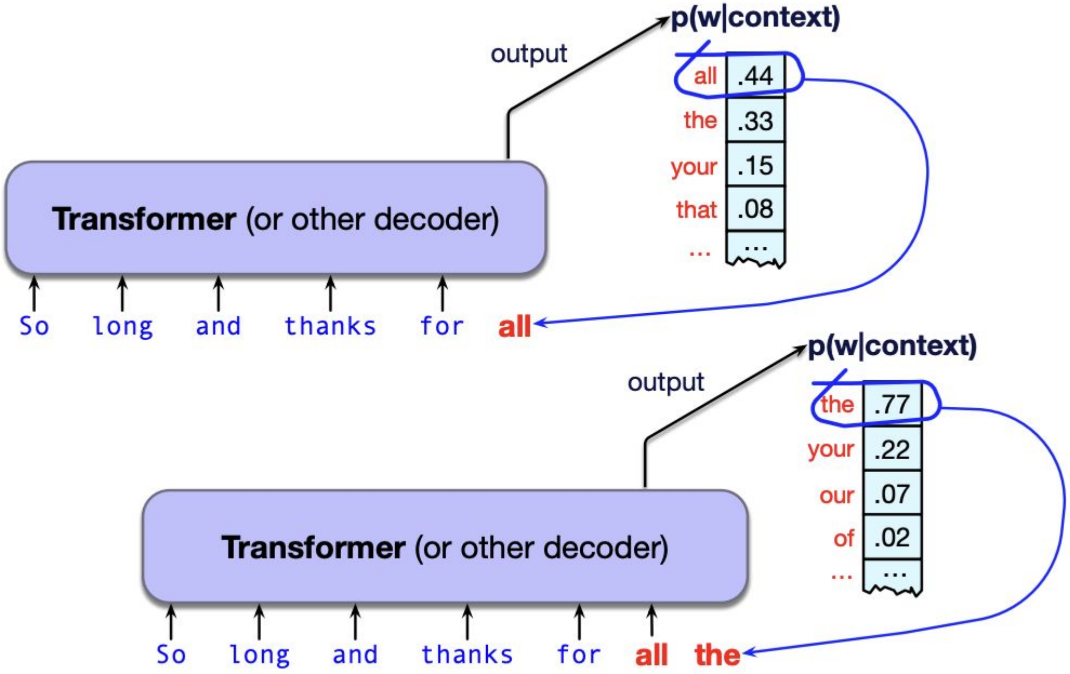
LLM Conditional Generation: Generating text conditioned on previous text



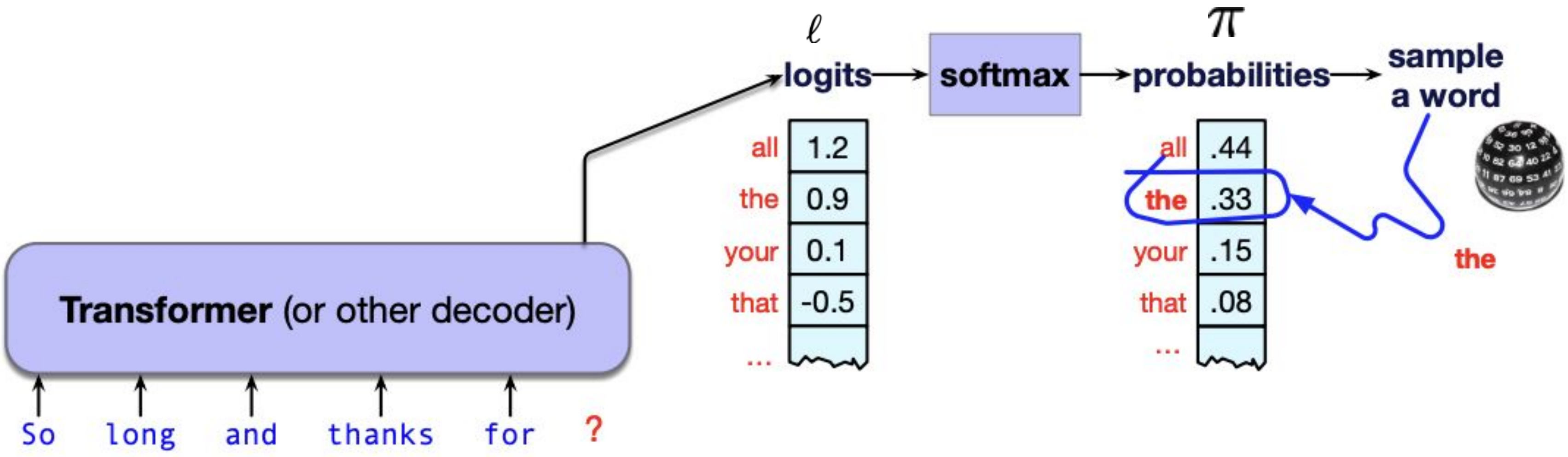
- Assigns probabilities to sequences of words
- Generate text by sampling possible next words
- Are trained initially by learning to guess the next word

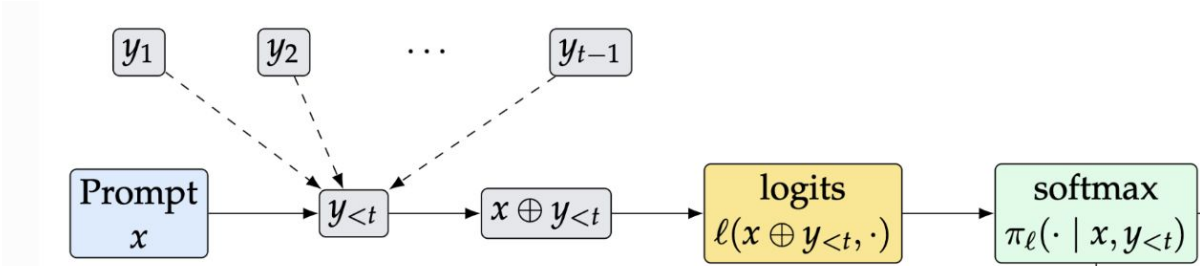
Images are taken from [here]

Autoregressive models : compute probability of the next word given previous words



Sampling from an LLM : predict the next word





Factorization of the distribution tokens and pros and cons

$$\pi_\ell(v \mid x, y_{<t}) = \frac{\exp(\ell(x \oplus y_{<t}, v))}{\sum_{v' \in \mathcal{V}} \exp(\ell(x \oplus y_{<t}, v'))} \longrightarrow \pi_\ell(y \mid x) = \prod_{t=1}^{T} \pi_\ell(y_t \mid x \oplus y_{<t})$$

$$\log \pi_\ell(y \mid x) = \sum_{t=1}^{T} \log \pi_\ell(y_t \mid x \oplus y_{<t}) \longrightarrow \mathcal{L}(y \mid x) = -\frac{1}{T} \sum_{t=1}^{T} \log \pi_\ell(y_t \mid x, y_{<t})$$

Pros

a) Tractable training objective!

(b) Efficient supervision ("teacher forcing") During training you condition on the *true* prefix, so you can compute losses for every position.

(c) Easy generation : Sampling/decoding is natural: repeatedly sample (or choose) next token

(d) Expressive dependencies Even though it's factorized, each conditional can depend on the entire past, so it can represent complex sequence structure.

Cons

a) Exposure bias Training conditions on the true history, but at inference the model conditions on its own generated tokens. Early mistakes can compound.

(b) Sequential generation cost To generate TTT tokens you do TTT steps. That's slower than models that can generate many tokens in parallel.

(c) Directionality / fixed order The factorization is tied to an ordering (typically left-to-right). Some tasks might benefit from bidirectional or alternative factorizations.

(d) Long-range errors While the model *can* condition on long history, in practice it may forget, drift, or struggle with very long contexts.

Perplexity of a model : a measure of accuracy of the model

$$\log \pi_\ell(y \mid x) \;=\; \sum_{t=1}^{T} \log \pi_\ell(y_t \mid x \oplus y_{<t}) \quad \longrightarrow \quad \mathrm{PPL}(y \mid x) \;=\; \exp\left(-\frac{1}{T} \sum_{t=1}^{T} \log \pi_\ell(y_t \mid x, y_{<t}) \right)$$

$$\mathrm{PPL}(y \mid x) = \left(\prod_{t=1}^{T} \pi_\ell(y_t \mid x, y_{<t}) \right)^{-\frac{1}{T}} .$$

- The perplexity of an LLM on a test set is the inverse probability of the test set normalized by the number of or tokens.
- The lower the perplexity of a model on the data, the better the model

The tokenizer : map language to integers

$$\text{enc} : \Sigma \to \{1, \ldots, |V|\}^n \quad \Rightarrow \quad (t_1, \ldots, t_n) \in \{1, \ldots, |V|\}^n$$

$$\text{dec} : \{1, \ldots, |V|\}^n \to \Sigma$$

Instead of 1 token = 1 word, the vocabulary contains subwords (pieces). Common words may be 1 token; rare words get split.

A common training objective is: build a vocabulary V so that typical text can be represented with few tokens while still covering everything.

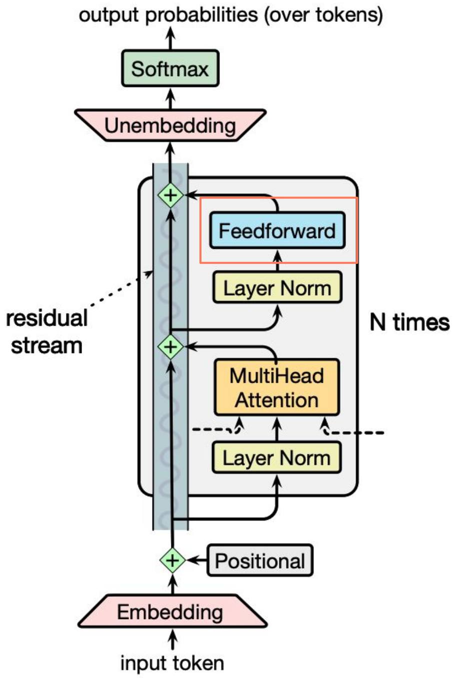
- Start with basic symbols (bytes or characters).
- Repeatedly merge frequent adjacent pairs to create new tokens.

Ideally : $\text{dec}(\text{enc}(x)) = x$

I. Large Language Model : architecture and sampling

A. What is an Large Language Model ?
B. Attention Mechanism

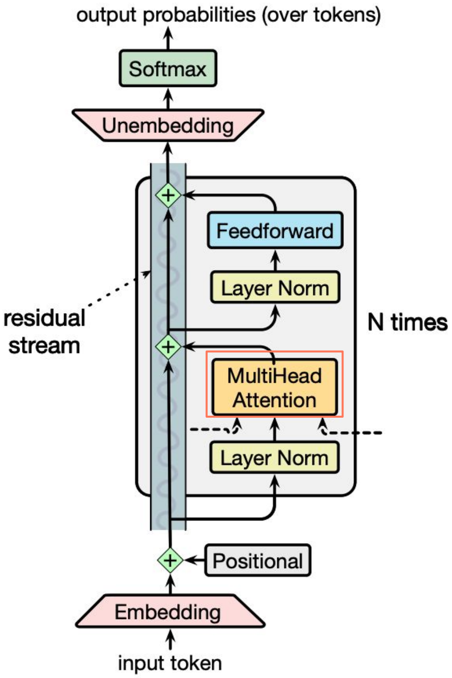
Under the hood ? The transformer

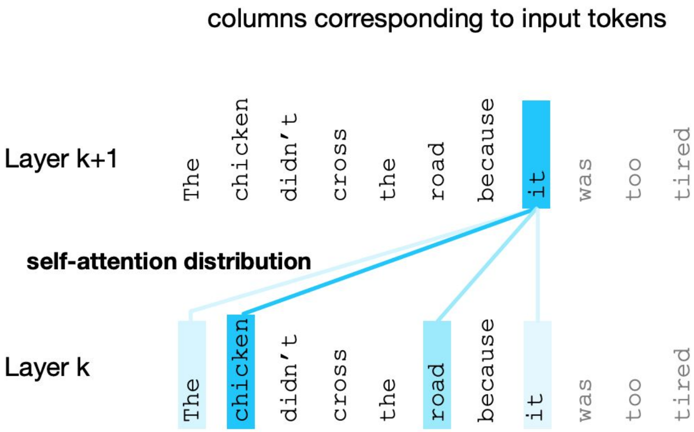


Feedforward

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W_1} + b_1)\mathbf{W_2} + b_2$$

Intuition of the Attention mechanism





One head attention layer equations

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{V}}$$

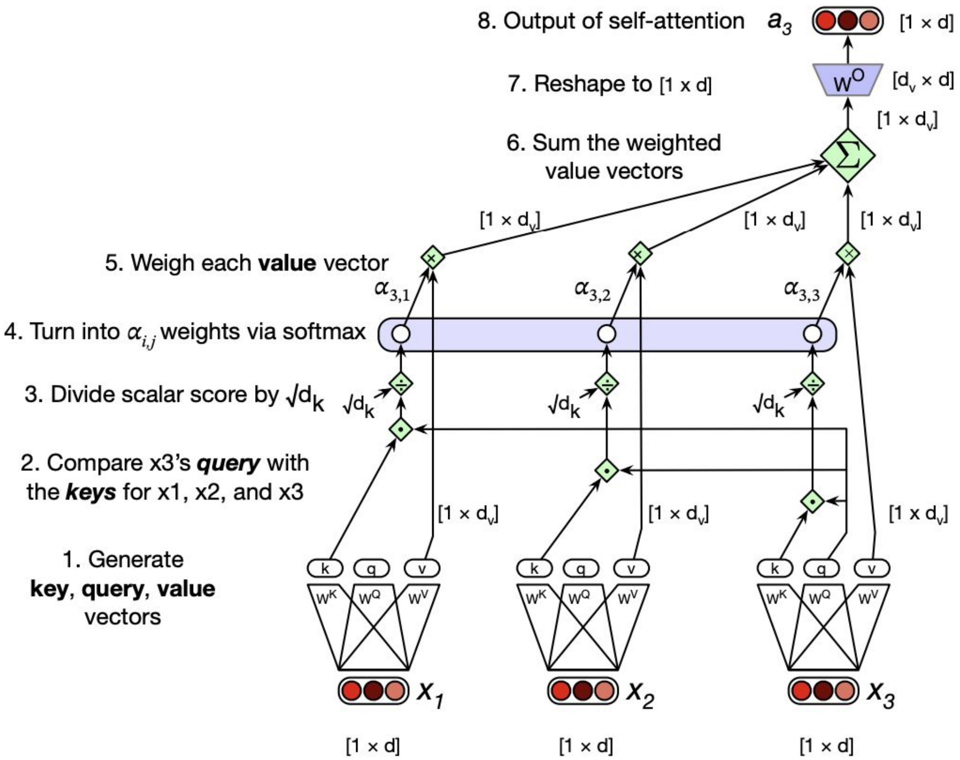
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\mathbf{head}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

$$\mathbf{a}_i = \mathbf{head}_i \mathbf{W}^{\mathbf{O}}$$

One head attention layer



$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}};\quad \mathbf{k}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{K}};\quad \mathbf{v}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{V}}$$

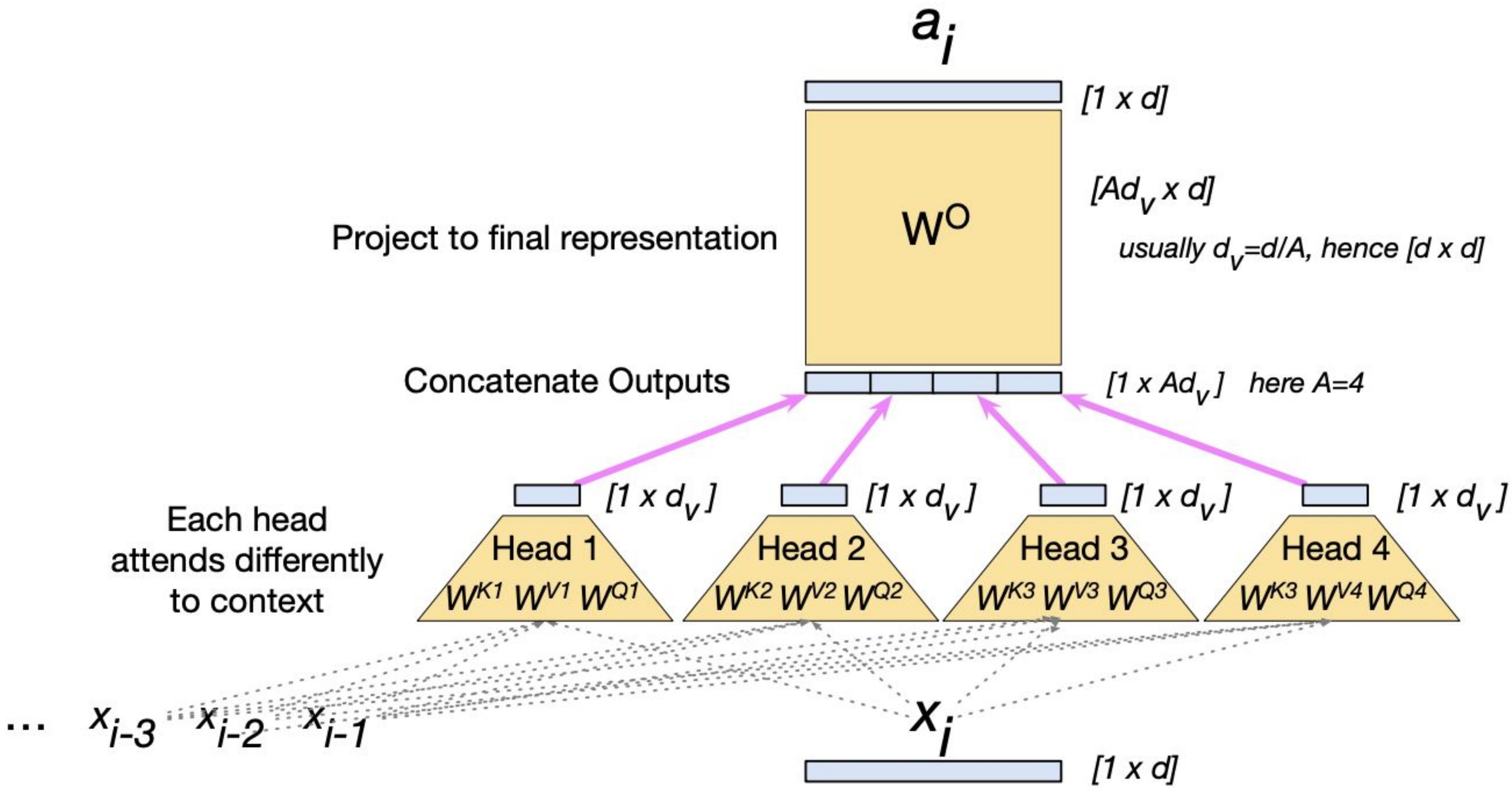
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j))\ \forall j \leq i$$

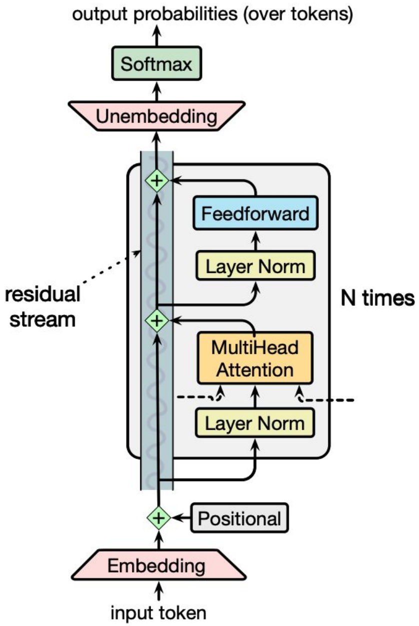
$$\mathbf{head}_i = \sum_{j \leq i} \alpha_{ij}\mathbf{v}_j$$

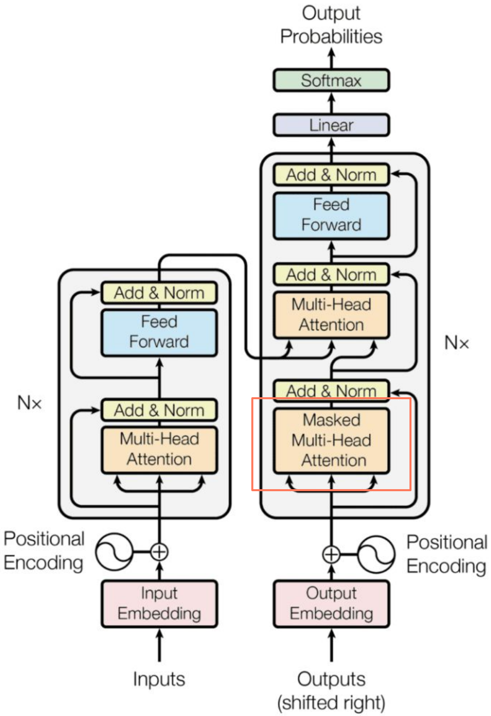
$$\mathbf{a}_i = \mathbf{head}_i\mathbf{W}^{\mathbf{O}}$$

Multi-head attention layer



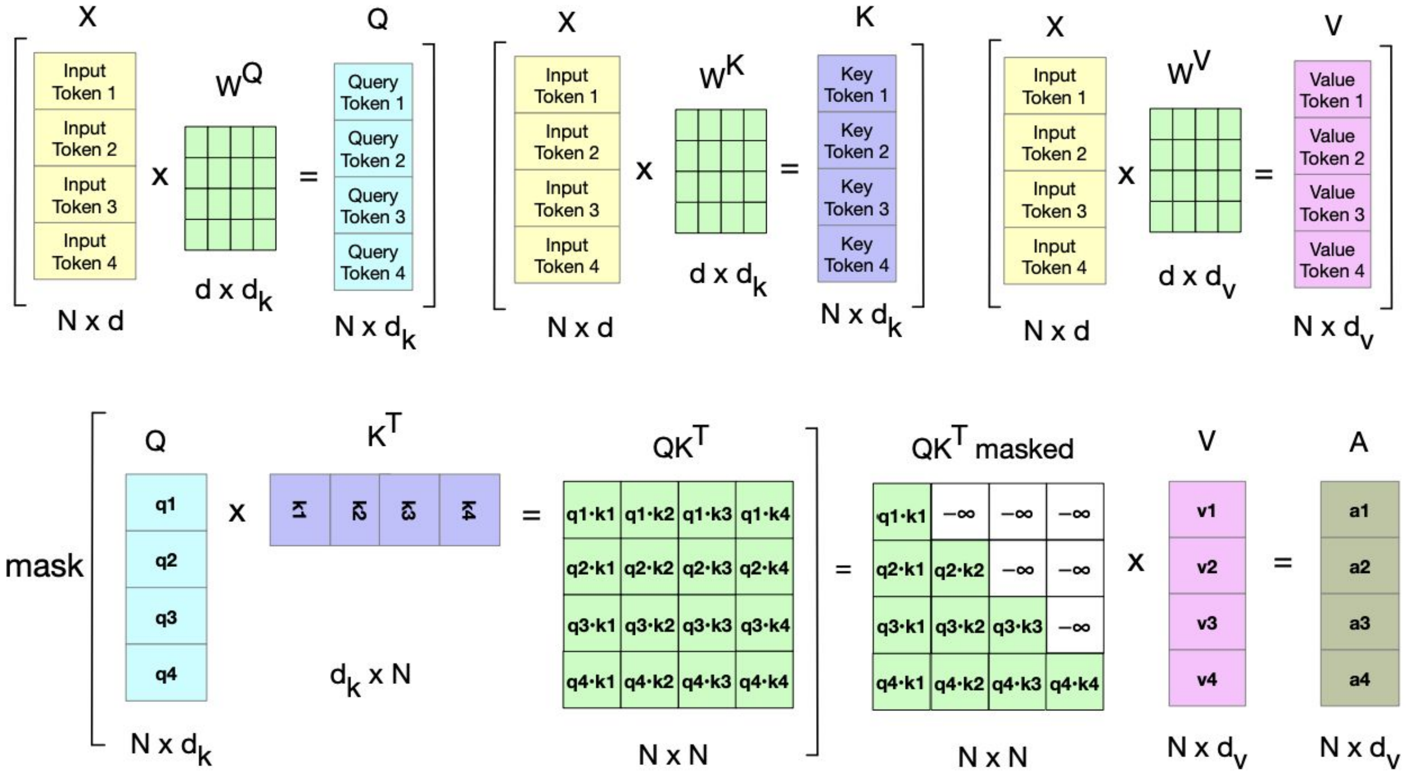
The transformer block used in LLMs use Masked Attention





From "Attention in all you need"   https://fleuret.org/public/lbdl.pdf

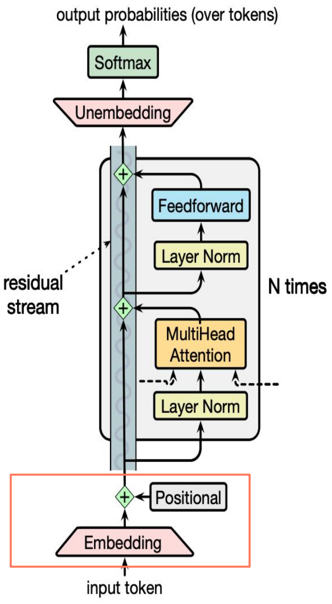
Masked-Attention computation

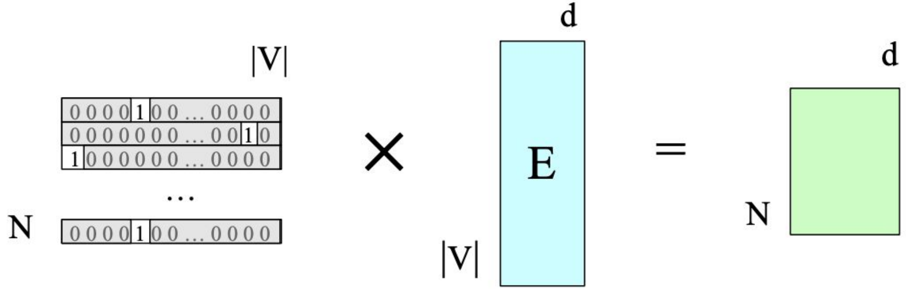


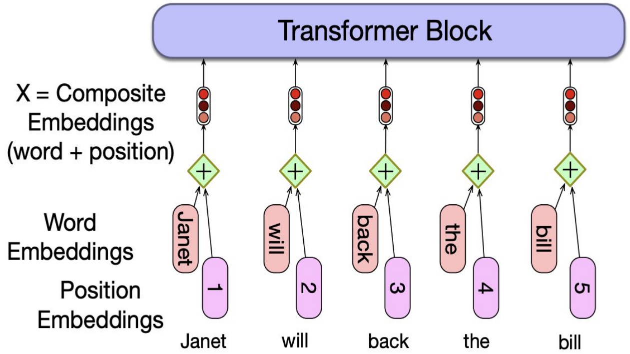
I. Large Language Model : architecture and sampling

A. What is an Large Language Model ?
B. Attention Mechanism
C. LLM architecture
Embedding, Positional embedding, Layer norm, Residual Connection

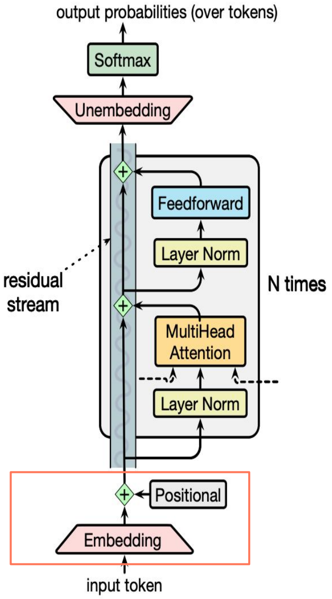
Embeddings and Position Embeddings







Positional Encoding : RoPE



Self-attention is permutation-invariant !

il you reorder tokens, the dot-products don't "know" positions.
So we inject position information :

p = position index (0..L-1), **d = model dimension (even)**, $i = 0..(d/2 - 1)$

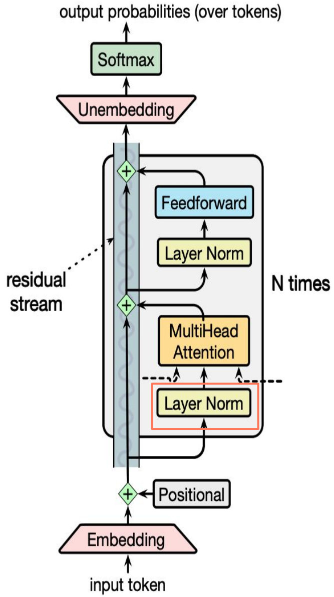
$$\mathrm{PE}(p)_{2i} = \sin\left(\frac{p}{10000^{2i/d}}\right), \qquad \mathrm{PE}(p)_{2i+1} = \cos\left(\frac{p}{10000^{2i/d}}\right).$$

$$\tilde{x}_p = x_p + \mathrm{PE}(p).$$

Why sin/cos? Each pair (2i,2i+1) is a sinusoid at frequency $\omega_i = 10000^{-2i/d}$
This creates a smooth, multi-scale "signature" for positions, and (importantly) lets the model infer relative offsets using trig identities.

Exercice : Show that you can differentiate tokens using trigonometric identities using RoPE in attention mechanism.

Layer norm



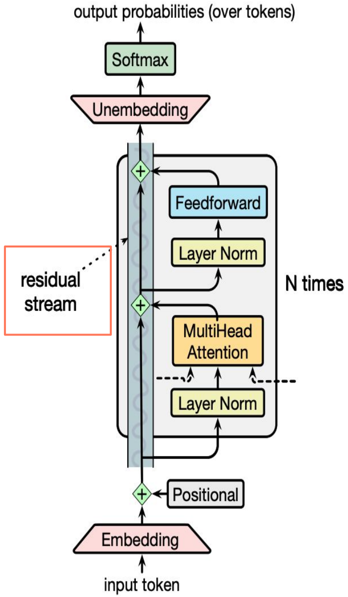
$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

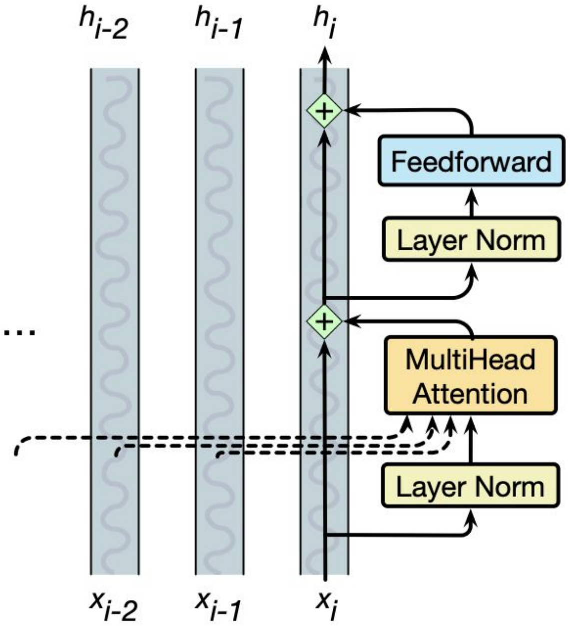
$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

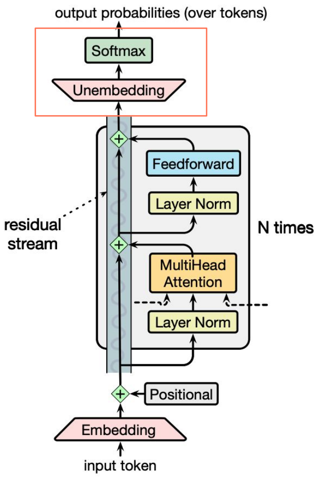
Residual connection

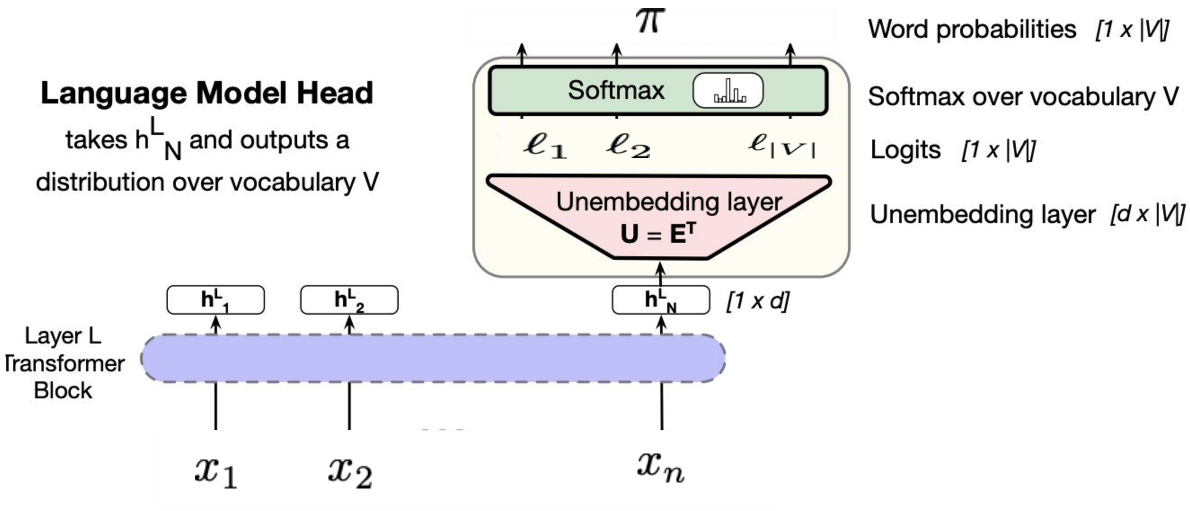




$$
\begin{aligned}
\mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{t}_1^1, \cdots, \mathbf{t}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\
\mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
\end{aligned}
$$

Projection to vocabulary size and softmax

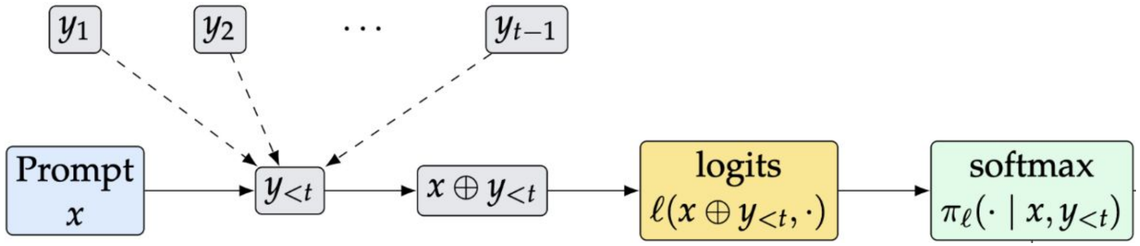




I. Large Language Model : architecture and sampling

A. What is an Large Language Model ?
B. Attention Mechanism
C. LLM architecture
    Toenizer, Embedding, Positional embedding Layer norm, Residual Connection
D. LLM Sampling/Decoding

Classical Decoding/Sampling from an LLM



Greedy decoding (argmax) :

$$y_t = \arg\max_{a \in \mathcal{V}} \pi_\ell(a \mid x, y_{<t}).$$

Stochastic sampling :

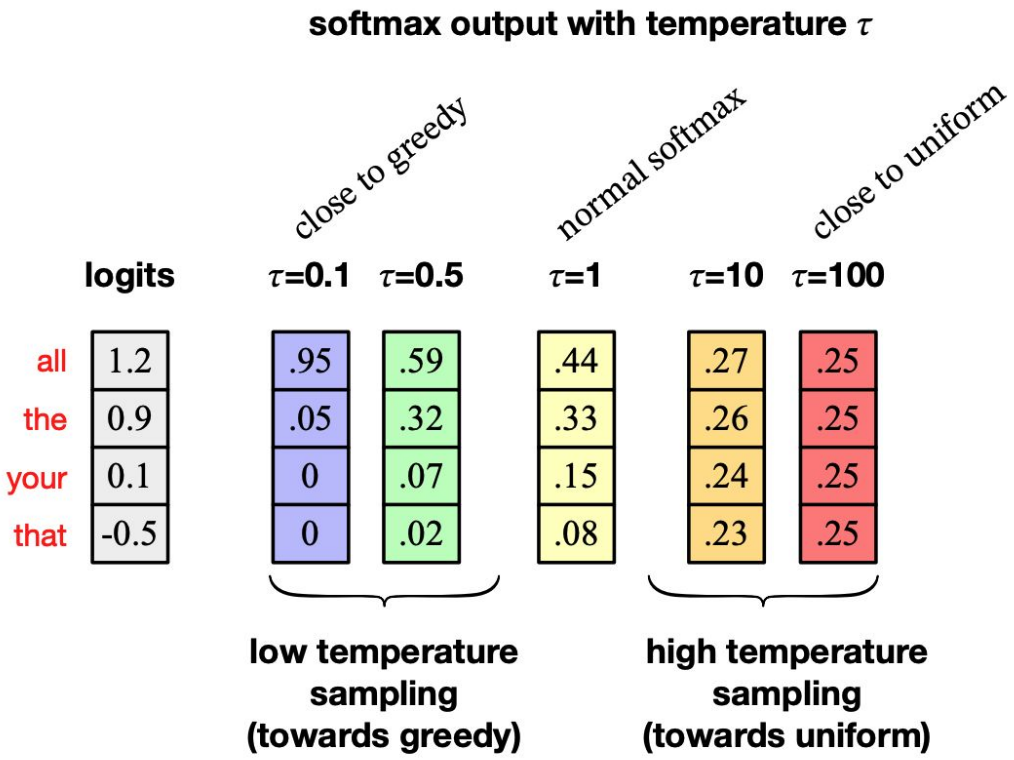
$$y_t \sim \pi_\ell(\cdot \mid x, y_{<t}).$$

Temperature scaling :

$$\pi_\ell^{(\tau)}(a \mid x, y_{<t}) = \frac{\pi_\ell(a \mid x, y_{<t})^{1/\tau}}{\sum_{b \in \mathcal{V}} \pi_\ell(b \mid x, y_{<t})^{1/\tau}}.$$

$$y_t \sim \pi_\ell^{(\tau)}(\cdot \mid x, y_{<t}).$$

Temperature Scaling



Top-p decoding

Idea : Limit sampling to the smallest set of tokens whose cumulative probability mass is at least $p \in (0, 1]$

Step 1: sort tokens by probability. $\{a_1, a_2, \ldots, a_{|\mathcal{V}|}\}$

$$\pi_\ell(a_1 \mid x, y_{<t}) \geq \pi_\ell(a_2 \mid x, y_{<t}) \geq \cdots \geq \pi_\ell(a_{|\mathcal{V}|} \mid x, y_{<t}).$$

Step 2: Let K be the smallest index such that $\sum_{i=1}^{K} \pi_\ell(a_i \mid x, y_{<t}) \geq p.$

Define the top-p set (nucleus) as $\mathcal{V}_p(x, y_{<t}) = \{a_1, a_2, \ldots, a_K\}.$

Step 3: renormalize and sample.

$$y_t \sim \pi_\ell^{(p)}(\cdot \mid x, y_{<t}).$$

$$\pi_\ell^{(p)}(a \mid x, y_{<t}) = \begin{cases} \dfrac{\pi_\ell(a \mid x, y_{<t})}{\sum_{b \in \mathcal{V}_p(x, y_{<t})} \pi_\ell(b \mid x, y_{<t})}, & a \in \mathcal{V}_p(x, y_{<t}), \\ 0, & a \notin \mathcal{V}_p(x, y_{<t}). \end{cases}$$

Sampling of LLMs : to go further

Top-p (nucleus) sampling : reference

This is the paper that defines **nucleus sampling (top-p)** and shows how it avoids degeneration compared to greedy / top-k

Speculative decoding (SD) / speculative sampling : reference

Introduces speculative sampling: draft model proposes multiple tokens, target model verifies with a modified rejection sampling scheme.

Theoretical analysis : reference

Medusa: Simple Framework for Accelerating LLM Generation with Multiple Heads : reference

vLLM project : reference

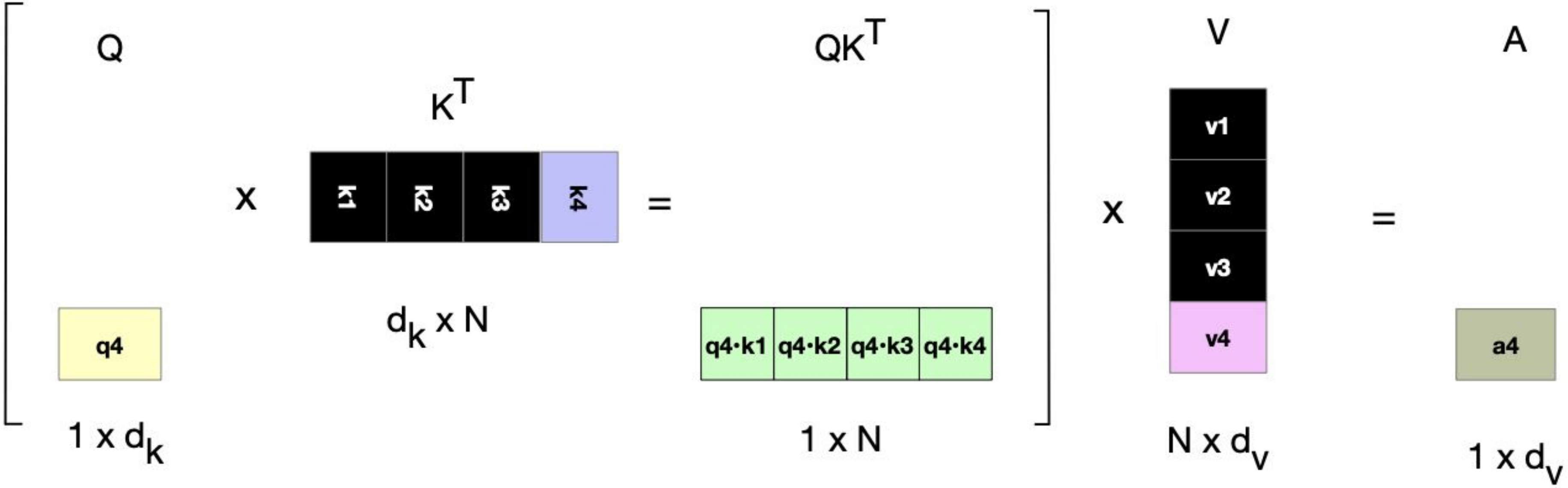
This is the technical paper behind **vLLM**, focusing on **Page Attention**, continuous batching, and high-throughput decoding.

In all these methods except SD, we sample from different policy than the current policy $y_t \sim \pi_\ell(\cdot \mid x, y_{<t})$.

I. Large Language Model : architecture and sampling

A. Attention Mechanism
B. LLM architecture
   Toenizer, Embedding, Positional embedding Layer norm, masked attention
C. LLM Sampling/Decoding
   1. Top p, temperature scaling, Modern Sampling engine : vLLM
D. Computation Optimization and Speed optimization
   1. KV cache

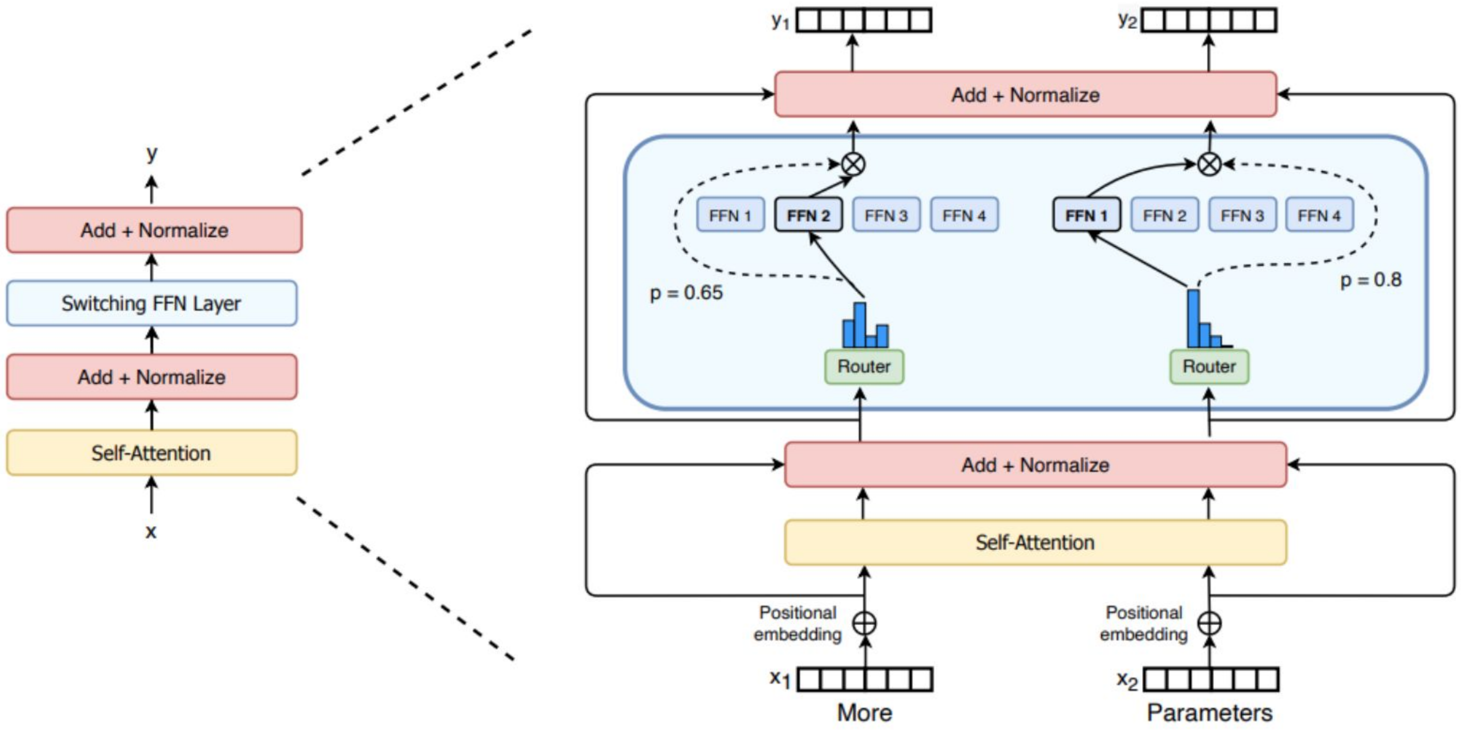
KV Cache



The factor gained in the attention complexity computation using KV cache is :

L = max sequence length of your training !

Mixture of Experts (MoE) architecture



Mixture of Experts (MoE) architecture

Classical dense Transformer layer

$$\tilde{x} = x^{(l)} + \mathrm{MHSA}\Big(\mathrm{LN}\big(x^{(l)}\big)\Big), \quad x^{(l+1)} = \tilde{x} + \mathrm{FFN}(\mathrm{LN}(\tilde{x})), \qquad \mathrm{FFN}(h) = W_2\,\phi(W_1 h + b_1) + b_2.$$

1) MoE layer: replace FFN with Router + Experts

$$\tilde{x} = x^{(l)} + \mathrm{MHSA}\Big(\mathrm{LN}\big(x^{(l)}\big)\Big).$$

$$h_t = \mathrm{LN}(\tilde{x}_t) \in \mathbb{R}^d.$$

$$s_t = W_r h_t + b_r \in \mathbb{R}^E,$$

$$p_t = \mathrm{softmax}(s_t), \qquad p_{t,e} = \frac{\exp(s_{t,e})}{\sum_{j=1}^{E} \exp(s_{t,j})}.$$

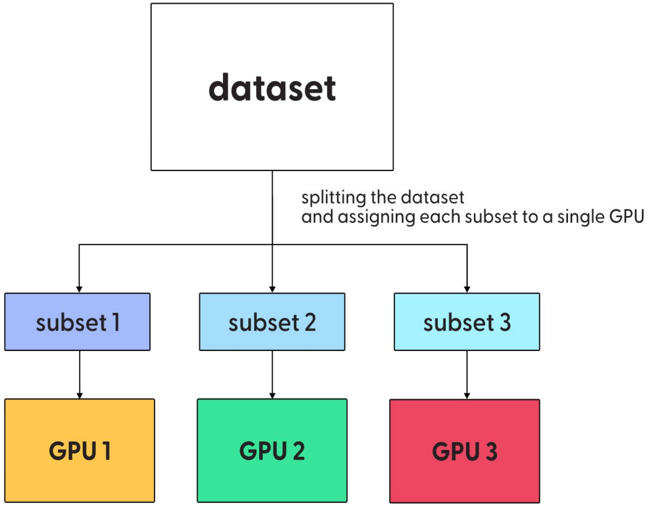
2) Top-k selection and sparse gate weights:

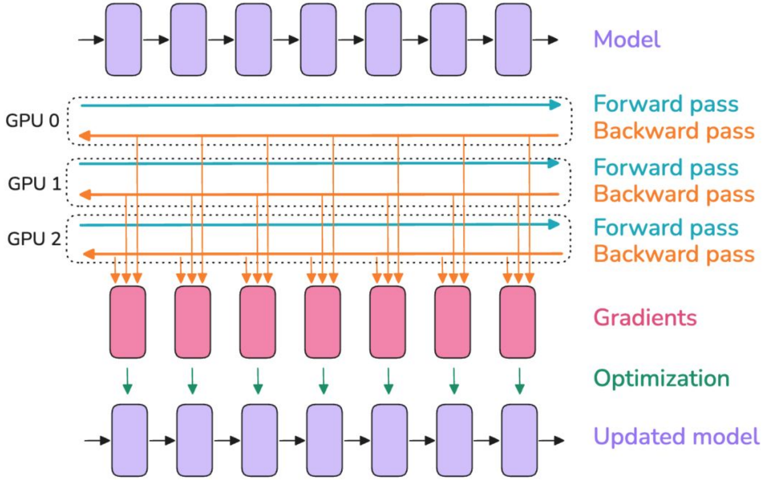
$$\mathcal{T}_t = \mathrm{TopK}(s_t, k),$$

$$g_{t,e} = \begin{cases} \dfrac{\exp(s_{t,e})}{\sum_{j \in \mathcal{T}_t} \exp(s_{t,j})}, & e \in \mathcal{T}_t, \\ 0, & \text{otherwise.} \end{cases}$$

3) MoE combination (only k experts are nonzero) $\mathrm{MoE}(h_t) = \sum_{e=1}^{E} g_{t,e}\, f_e(h_t).$

A example of memory optimisation : Data Parallelism





To go further : Tensort Parallelism , Fully-Sharded-Data-Parallelism (FSDP), Sequence Parallelism etc… in [Hugging Face Book !]

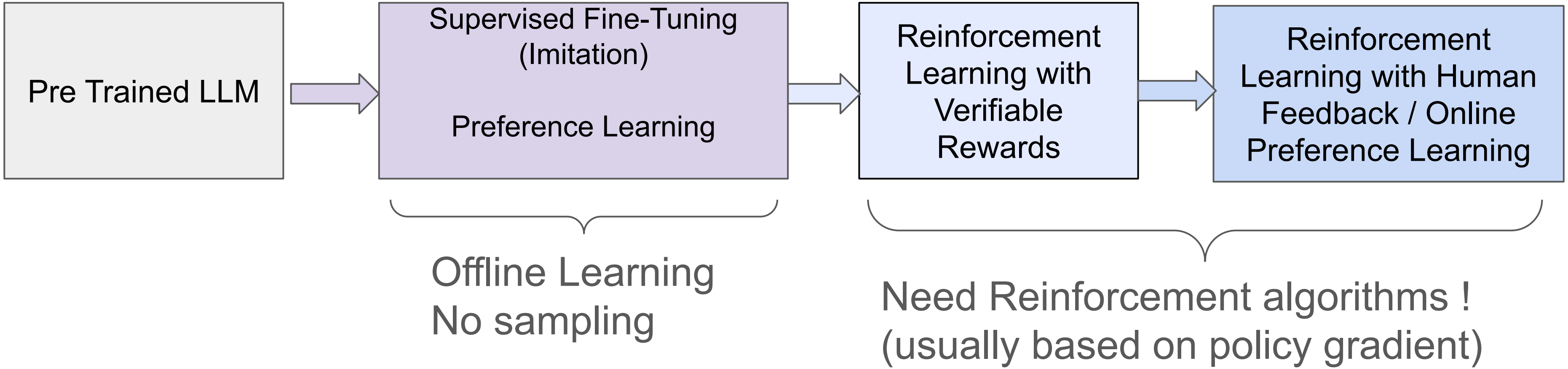
II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?

II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?

LLM training pipeline



II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models

Classical optimizers used in practice

Vanilla SGD

$$g_t = \nabla_\theta \mathcal{L}(\theta_t)$$
$$\theta_{t+1} = \theta_t - \eta\, g_t$$

SGD + momentum

$$v_t = \mu v_{t-1} + g_t \quad ; \quad \theta_{t+1} = \theta_t - \eta\, v_t$$

Adam :

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)(g_t \odot g_t)$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \theta_{t+1} = \theta_t - \eta\, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Muon optimizer : Muon paper form Kimi ; Torch implementation

II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models
c) Imitation Learning vs Preference Learning vs Reinforcement Learning

Reinforcement Learning vs Imitation vs Preference Learning

~~What is the best way to learn playing chess?~~

How to become the best chess player?

A) Watch people playing chess?

→ Imitation

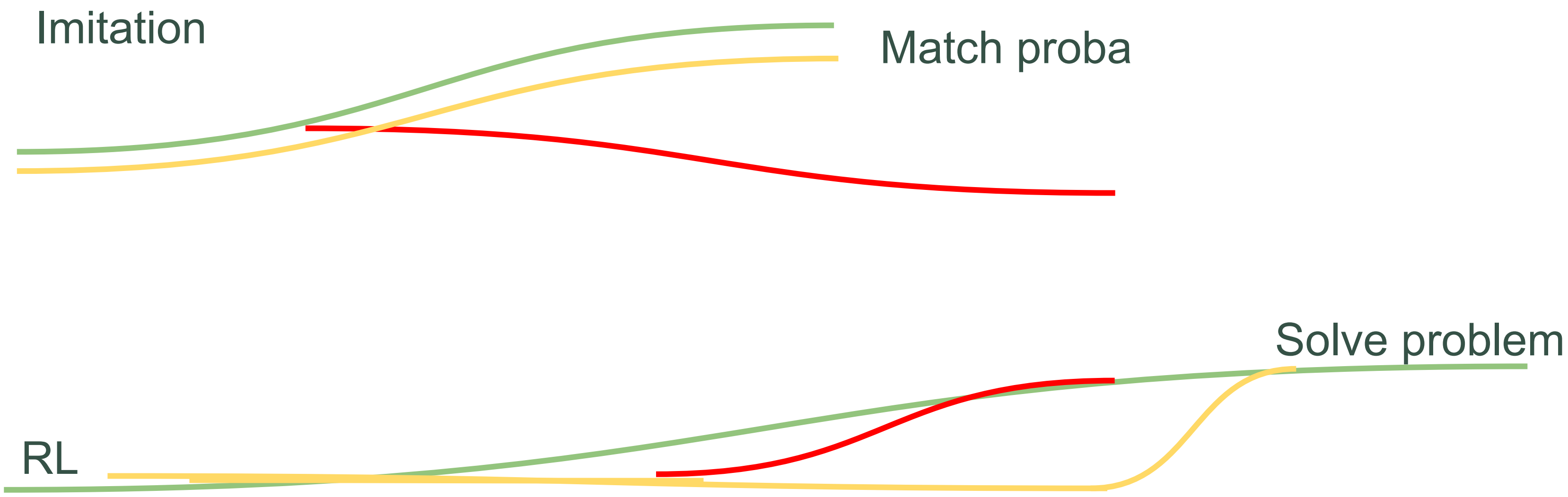
B) Look which moves are the best?

→ Preference

C) Play for the win?

→ RL

Reinforcement Learning vs Imitation

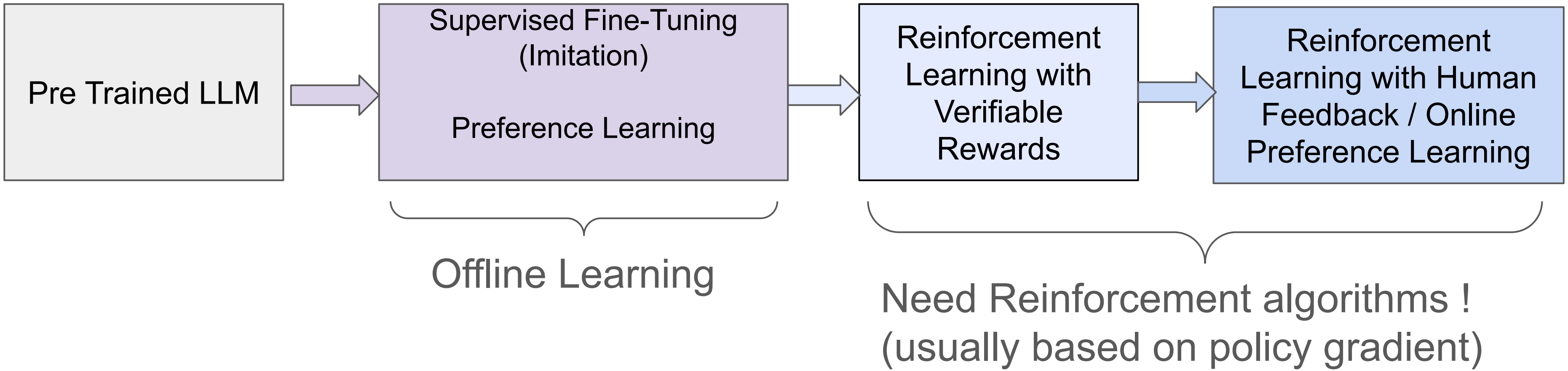


II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

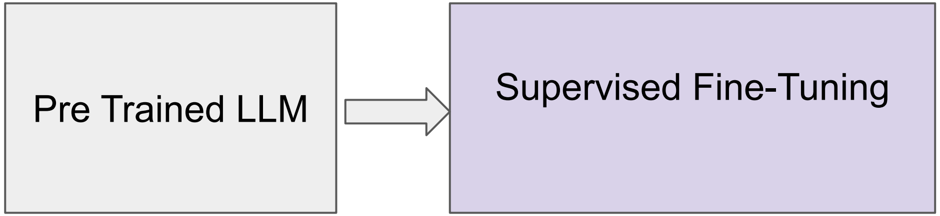
a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models
c) Imitation Learning vs Preference Learning vs Reinforcement Learning
d) Offline Learning
   (1) Pretraining and Supervised Fine-Tuning : the cross entropy loss

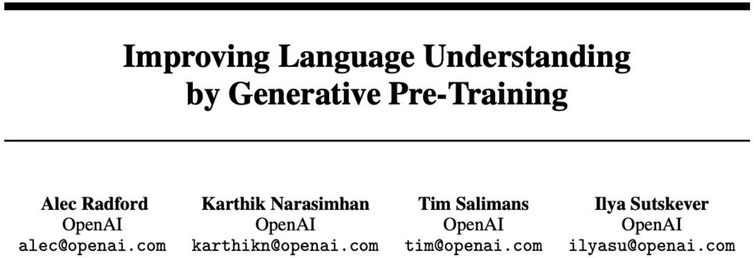
Online vs Offline Learning

LLM training pipeline



LLM training pipeline

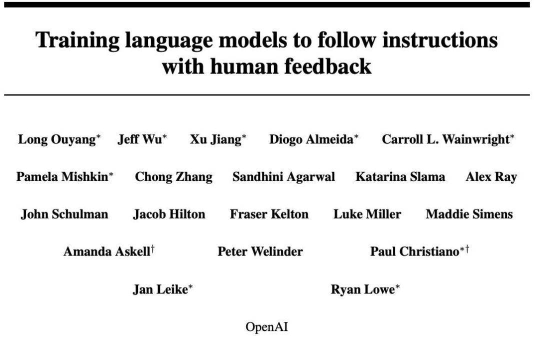




Improving Language Understanding by Generative Pre-Training

Alec Radford OpenAI alec@openai.com
Karthik Narasimhan OpenAI karthikn@openai.com
Tim Salimans OpenAI tim@openai.com
Ilya Sutskever OpenAI ilyasu@openai.com

GPT 1 paper

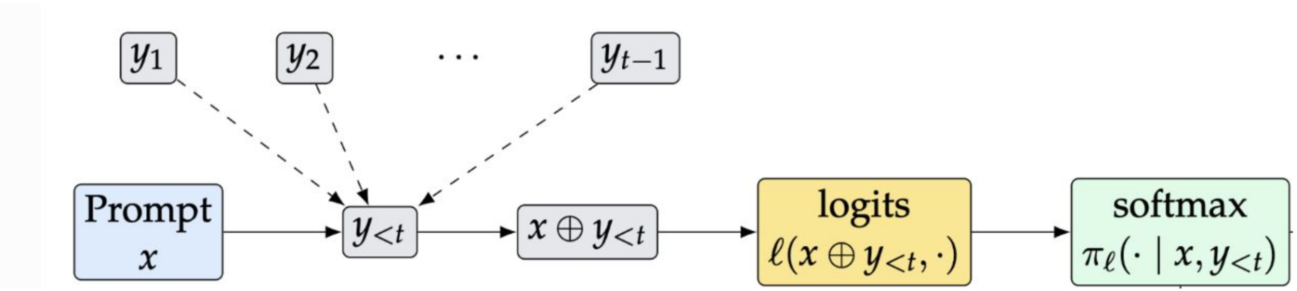


Training language models to follow instructions with human feedback

Long Ouyang* **Jeff Wu*** **Xu Jiang*** **Diogo Almeida*** **Carroll L. Wainwright***
Pamela Mishkin* **Chong Zhang** **Sandhini Agarwal** **Katarina Slama** **Alex Ray**
John Schulman **Jacob Hilton** **Fraser Kelton** **Luke Miller** **Maddie Simens**
Amanda Askell† **Peter Welinder** **Paul Christiano***†
Jan Leike* **Ryan Lowe***
OpenAI

GPT 3 paper

A) Supervised Fine-Tuning (or Imitation)



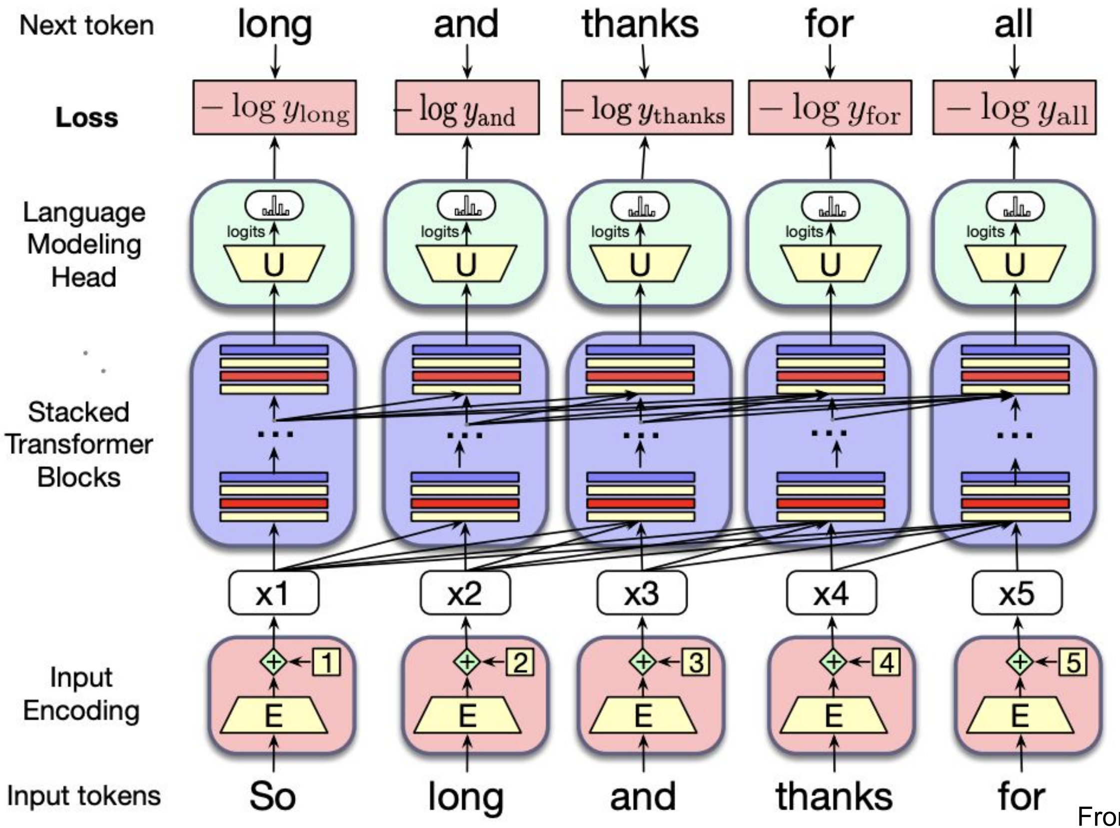
$$\mathcal{L}_{\mathrm{SFT}} = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\mathrm{SFT}}}\left[-\sum_{t=1}^{T_y} \log \pi_\ell\big(y_t \mid x, y_{<t}\big)\right].$$

Pros

- **Directly aligns to instructions** : you explicitly train “given prompt x, say y”
- **Simple & stable** : cross-entropy loss
- **Data efficient** : a relatively small SFT set can dramatically improve behavior
- **Deterministic supervision** : no reward model, no RL instability.

Cons

- **Exposure bias** : during training the model always sees *perfect* prefixes; at test time it sees its own mistakes, which it was never trained on.
- **Can overwrite skills** : naive SFT can cause catastrophic forgetting of some pretrained capabilities.
- **Needs high-quality labels**: humans must write good demonstrations; expensive to scale.



From CS 336: Language Modeling from Scratch (Stanford)

SFT vs Pre-training Self-Supervised losses

Supervised Fine-Tuning

$$\mathcal{L}_{\mathrm{SFT}} = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\mathrm{SFT}}}\left[-\sum_{t=1}^{T_y} \log \pi_\ell\big(y_t \mid x, y_{<t}\big)\right].$$

Pre Trained LLM

$$\mathcal{L}_{\mathrm{pre}} = \mathbb{E}_{w\sim\mathcal{D}_{\mathrm{pre}}}\left[-\sum_{t=1}^{T_w} \log \pi_\ell\big(w_t \mid w_{<t}\big)\right].$$

Train LLMs to predict the next word! **"Self-supervised"** learning is just uses the next word as the label.

Pros

- **Scales with cheap data** – uses unlabeled text; no humans needed.
- **Very stable & simple** – classic cross-entropy.
- **Strong general capabilities** – learns language, world knowledge, reasoning patterns, coding, etc.
- **Great transfer** – a good pretrained model can be adapted to many tasks with small extra data.

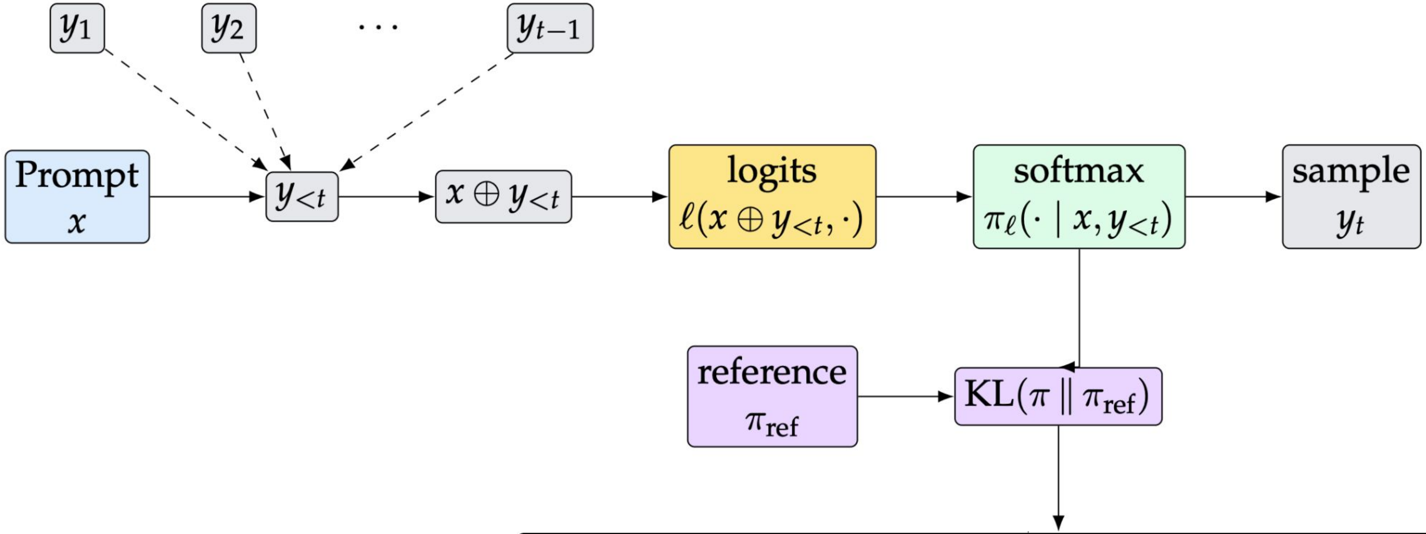
Cons

- **Not aligned** – it learns to mimic the internet, not to follow instructions or be “helpful/harmless”.
- **Copies undesirable behavior** – toxicity, bias, unsafe instructions, etc.
- **No task conditioning** – doesn’t know what *you* want; just continues text.
- **Harder controllability** – steering purely with prompts is limited.

II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models
c) Imitation Learning vs Preference Learning vs Reinforcement Learning
d) Offline Learning
    (1) Pretraining and Supervised Fine-Tuning : the cross entropy loss
    (2) Offline Preference Learning

II) The case of off policy Learning



RL Objective:

$$J(\pi) = \mathbb{E}_{x \sim \rho}\, \mathbb{E}_{y \sim \pi(\cdot|x)} \left[R(x, y) - \beta\, \mathrm{KL}(\pi \,\|\, \pi_{\text{ref}}) \right]$$

Online : $y \sim \pi(.|x)$

Offline : $y \sim \mu(.|x)$ $\mu(.|x) \neq \pi(.|x)$

Kullback-Leibler divergence and property

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_a P(a) \log \frac{P(a)}{Q(a)}.$$

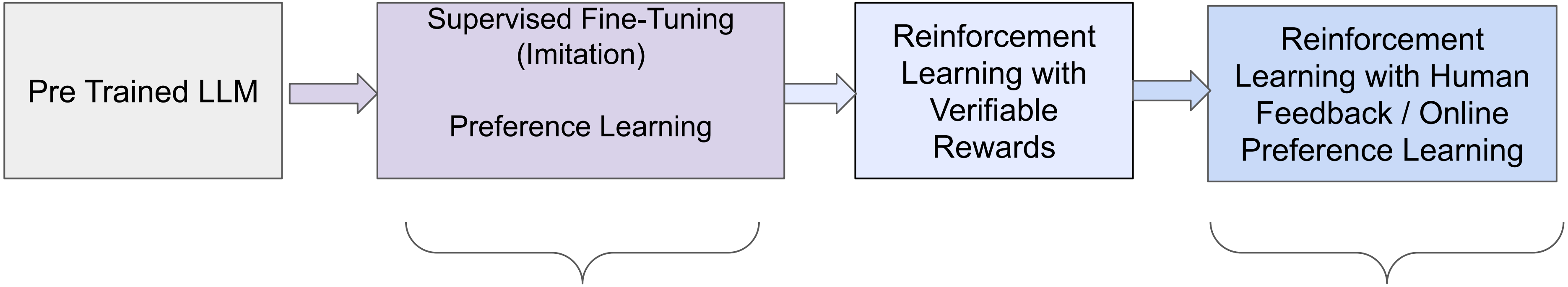
Properties :

$$D_{\mathrm{KL}}(P \parallel Q) \geq 0.$$

$$D_{\mathrm{KL}}(P \parallel Q) = 0 \iff P(a) = Q(a) \text{ for all } a \text{ (up to measure-zero sets)}.$$

$$D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P) \quad \text{in general.}$$

LLM training pipeline



Offline preference Learning

Online Preference Learning

Direct Preference Optimization: Your Language Model is Secretly a Reward Model

Rafael Rafailov*† Archit Sharma*† Eric Mitchell*†
Stefano Ermon†‡ Christopher D. Manning† Chelsea Finn†
†Stanford University ‡CZ Biohub
{rafailov,architsh,eric.mitchell}@cs.stanford.edu

Abstract

DPO

A General Theoretical Paradigm to Understand Learning from Human Preferences

Mohammad Gheshlaghi Azar Mark Rowland Bilal Piot
Daniel Guo Daniele Calandriello Michal Valko Rémi Munos
Google DeepMind

Abstract **1 Introduction**

IPO

DPO & co

When measuring psychological values or intangible factors, use pairwise comparisons

For subjective concepts, e.g., helpfulness, engagement, moral… language quality, there is no absolute rules. Therefore, we need to extract this subjectivity from human.

A LAW OF COMPARATIVE JUDGMENT[1]

BY L. L. THURSTONE

The University of Chicago

The object of this paper is to describe a new psychophysical law which may be called the *law of comparative judgment* and to show some of its special applications in the measurement of psychological values. The law of comparative

Relative Measurement

and its Generalization in Decision Making

Why Pairwise Comparisons are Central in Mathematics for the Measurement of Intangible Factors

The Analytic Hierarchy/Network Process

(To the Memory of my Beloved Friend Professor Sixto Rios Garcia)

Thomas L. Saaty*

Preference Learning : the Bradley-Terry Model

- Learn to reproduce the pairwise comparison to capture psychological values or intangible factors.
- Turn this pairwise comparison into a score to optimize it.

$$P(w > l) = \frac{p_w}{p_w + p_l}$$

Bradley-Terry model:

$$= \frac{e^{r_w}}{e^{r_w} + e^{r_l}}$$

where $p^w = e^{r_w}$ and $p^l = e^{r_l}$

$$= \frac{1}{1 + e^{r_l - r_w}}$$

$$= \sigma(r_w - r_l), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Preference Learning : the Bradley-Terry Model

Bradley-Terry model

$$P(w > l) = \sigma(r_w - r_l), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Using two pairs of completion of our dataset (w for win and l for loose) :

$$P(y_w > y_l \mid x) = \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big),$$

Use simple classification loss, **maximizing the probability of the winning completion** compared to the worst completion :

$$L(\phi) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big)\right].$$

Questino : what is the reward for the best/worst completion for our RL objective function ?

Preference Learning : from Bradley-Terry Model to DPO

$$J(\pi) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi(\cdot|x)}\left[R(x,y) \;-\; \beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} \right].$$

$$\mathcal{L}(\pi) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi(\cdot|x)}\left[\log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} \;-\; \frac{1}{\beta} R(x,y) \right]$$

$$= \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi(\cdot|x)}\left[\log \frac{\pi(y \mid x)}{\frac{1}{Z(x)}\pi_{\text{ref}}(y \mid x)\exp\left(\frac{1}{\beta}R(x,y)\right)} \;-\; \log Z(x) \right],$$

With the Partition function $Z(x) \;=\; \sum_{y} \pi_{\text{ref}}(y \mid x)\, \exp\left(\frac{1}{\beta}R(x,y)\right).$

$$\mathcal{L}(\pi) = \mathbb{E}_{x\sim\mathcal{D}}\left[D_{\text{KL}}\left(\pi(\cdot \mid x) \,\middle\|\, \frac{1}{Z(x)} \pi_{\text{ref}}(\cdot \mid x) \exp\left(\frac{1}{\beta}R(x,\cdot)\right) \right) \;-\; \log Z(x) \right].$$

Preference Learning : from Bradley-Terry Model to DPO

$$\mathcal{L}(\pi) = \mathbb{E}_{x \sim \mathcal{D}} \left[D_{\mathrm{KL}} \left(\pi(\cdot \mid x) \,\middle\|\, \frac{1}{Z(x)} \pi_{\mathrm{ref}}(\cdot \mid x) \exp\left(\tfrac{1}{\beta} R(x, \cdot)\right) \right) - \log Z(x) \right].$$

Optimal policy (dropping the $Z(x)$ term that is independent of the policy in the optimization)

$$\pi^\star(y \mid x) = \frac{1}{Z(x)} \pi_{\mathrm{ref}}(y \mid x) \exp\left(\tfrac{1}{\beta} R(x, y)\right), \qquad Z(x) = \sum_y \pi_{\mathrm{ref}}(y \mid x) \exp\left(\tfrac{1}{\beta} R(x, y)\right)$$

$$R^\star(x, y) = \beta \log \frac{\pi^\star(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta \log Z(x).$$

Plug into Bradley-Terry Model!

Preference Learning : from Bradley-Terry Model to DPO

$$p^\star(y_w \succ y_l \mid x) = \frac{\exp\left(\beta \log \frac{\pi^\star(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x)\right)}{\exp\left(\beta \log \frac{\pi^\star(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x)\right) + \exp\left(\beta \log \frac{\pi^\star(y_l|x)}{\pi_{\text{ref}}(y_l|x)} + \beta \log Z(x)\right)}$$

$$= \frac{1}{1 + \exp\left(\beta \log \frac{\pi^\star(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta \log \frac{\pi^\star(y_w|x)}{\pi_{\text{ref}}(y_w|x)}\right)}.$$

Fitting the policy via logistic loss with $\sigma(z) = 1/(1 + e^{-z}))$

$$\mathcal{L}(\theta) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right].$$

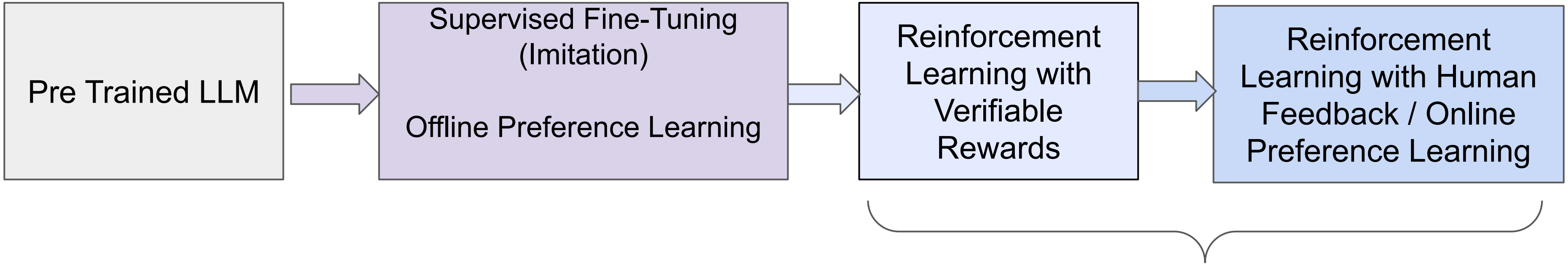
Increase **reward/likelihood** of preferred completion

Decrease **reward/likelihood** of preferred completion

II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models
c) Imitation Learning vs Preference Learning vs Reinforcement Learning
d) Offline Learning
    (1) Pretraining and Supervised Fine-Tuning : the cross entropy loss
    (2) Offline Preference Learning
e) Reinforcement Learning : from offline to online learning
    (1) Policy Gradient with KL regularization

LLM training pipeline



Need Reinforcement algorithms !
(usually based on policy gradient)

Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs

Arash Ahmadian, *Cohere For AI* · Chris Cremer, *Cohere* · Matthias Gallé, *Cohere*
Marzieh Fadaee, *Cohere For AI* · Julia Kreutzer, *Cohere For AI* · Olivier Pietquin, *Cohere*
Ahmet Üstün, *Cohere For AI* · Sara Hooker, *Cohere For AI*
{arash,olivier,ahmet,sarahooker}@cohere.com

RLOO

Contrastive Policy Gradient: Aligning LLMs on sequence-level scores in a supervised-friendly fashion

Yannis Flet-Berliac[†], Nathan Grinsztajn[†], Florian Strub, Bill Wu, Eugene Choi, Chris Cremer, Arash Ahmadian, Yash Chandak , Mohammad Gheshlaghi Azar, Olivier Pietquin, Matthieu Geist[*]

Cohere

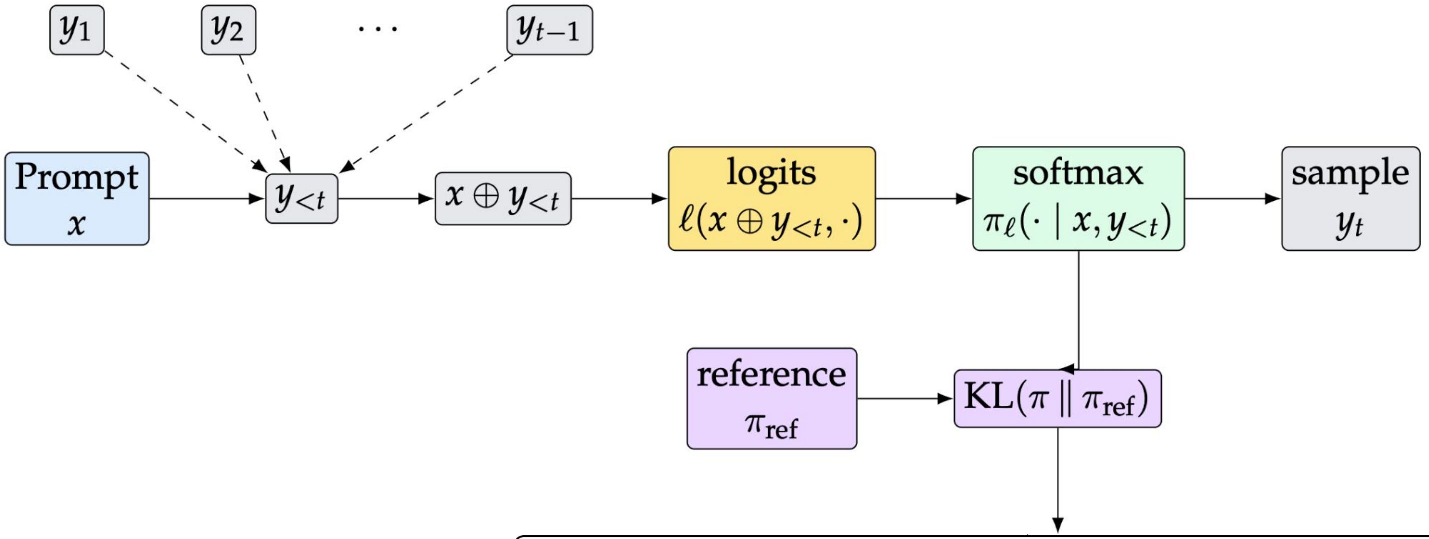
CoPG

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI
research@deepseek.com

GRPO

II) The case of off policy Learning



RL Objective:

$$J(\pi) = \mathbb{E}_{x \sim \rho}\, \mathbb{E}_{y \sim \pi(\cdot|x)} \left[R(x, y) - \beta\, \mathrm{KL}(\pi \,\|\, \pi_{ref}) \right]$$

Online : $y \sim \pi(.|x)$

Offline : $y \sim \mu(.|x)$ $\mu(.|x) \neq \pi(.|x)$ → First Idea ? Policy Gradient Method !

Short Recall on Policy Gradient Method in offline/hybrid case

RL Objective:

$$J(\pi) = \mathbb{E}_{x\sim\rho}\,\mathbb{E}_{y\sim\pi(\cdot|x)}\left[R(x,y) - \beta\,\mathrm{KL}(\pi \,\|\, \pi_{\mathrm{ref}})\right]$$

$$R^{\pi}_{\beta}(x,y) = R(x,y) - \beta \ln \frac{\pi(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)}$$

Policy Gradient Theorem : $\nabla J(\pi) = \mathbb{E}_{\substack{x\sim\rho \\ y\sim\pi(\cdot\,|\,x)}}\left[\left(R(x,y) - \beta \log \frac{\pi(y\,|\,x)}{\pi_{\mathrm{ref}}(y\,|\,x)}\right)\nabla \log \pi(y\,|\,x)\right].$

$$\widehat{\nabla_\theta J} = \frac{1}{B}\sum_{i=1}^{B}\left(R(x_i,y_i) - \beta \log \frac{\pi_\theta(y_i \mid x_i)}{\pi_{\mathrm{ref}}(y_i \mid x_i)}\right)\nabla_\theta \log \pi_\theta(y_i \mid x_i).$$

Short Recall on Policy Gradient Method (log trick)

$$\begin{aligned}\nabla_\theta \mathbb{E}_{Z\sim p_\theta}[f(Z)] &= \nabla_\theta \int f(z)\, p_\theta(z)\, dz \\ &= \int f(z)\, \nabla_\theta p_\theta(z)\, dz \\ &= \int f(z)\, p_\theta(z)\, \nabla_\theta \log p_\theta(z)\, dz \\ &= \mathbb{E}_{Z\sim p_\theta}[f(Z)\, \nabla_\theta \log p_\theta(Z)]\,.\end{aligned}$$

Two simple Corollary :

$$\mathbb{E}_{Z\sim p_\theta}[\nabla_\theta \log p_\theta(Z)] = \nabla_\theta \int p_\theta(z)\, dz = \nabla_\theta 1 = 0,$$

★

$$\mathbb{E}_{Z\sim p_\theta}[c\, \nabla_\theta \log p_\theta(Z)] = 0.$$

★★

Policy Gradient Theorem : $\nabla J(\pi) = \mathbb{E}_{\substack{x\sim\rho \\ y\sim\pi(\cdot\,|\,x)}}\left[\left(R(x,y) - \beta \log \frac{\pi(y\,|\,x)}{\pi_{\text{ref}}(y\,|\,x)}\right)\nabla \log \pi(y\,|\,x)\right].$

$$R^{\pi}_{\beta}(x,y) \;=\; R(x,y) \;-\; \beta \,\ln \frac{\pi(y\mid x)}{\pi_{\text{ref}}(y\mid x)}$$

This depends twice on the current policy : one in the sampled generation y and one in the regularizer

Proof :

$$\nabla J(\pi) = \mathbb{E}_{x\sim\rho}\left[\nabla\,\mathbb{E}_{y\sim\pi(\cdot|x)}\left[R(x,y) - \beta \ln \frac{\pi(y\mid x)}{\pi_{\text{ref}}(y\mid x)}\right]\right]$$

Derivative of a product (ab)' =a'b +b'a

$$= \mathbb{E}_{x\sim\rho}\left[\mathbb{E}_{y\sim\pi(\cdot|x)}\left[\left(R(x,y) - \beta \ln \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}\right)\nabla \ln \pi(y\mid x)\right] + \mathbb{E}_{y\sim\pi(\cdot|x)}[-\beta\,\nabla \ln \pi(y\mid x)]\right]$$

$$= \mathbb{E}_{x\sim\rho,\; y\sim\pi(\cdot|x)}\left[\left(R(x,y) - \beta \ln \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}\right)\nabla \ln \pi(y\mid x)\right] \qquad (\text{since } \mathbb{E}_{y\sim\pi}[\nabla \ln \pi(y\mid x)] = 0)$$

$$= \mathbb{E}_{x\sim\rho,\; y\sim\pi(\cdot|x)}\left[R^{\pi}_{\beta}(x,y)\,\nabla \ln \pi(y\mid x)\right].$$

KL-Regularised Policy Gradient Method
Pros and Cons

Pros

- Simple, general, and black-box: works with any differentiable policy and **arbitrary rewards.**
- KL-to-reference naturally stabilizes offline/hybrid learning and encodes **“stay-close-to-data.”**
- Straightforward to implement with **mini-batches.**

Cons related to the Reward + KL formulation :

- Sensitive to beta : if beta too small → drift off-support; too large → under-exploration / slow improvement.
- Careful reward/log-ratio scaling for numeric stability.

Cons of classical policy gradient formulation :

- I) High variance gradients; can be sample-inefficient without strong baselines/critics ? **Reinforcement Leave One Out**

  II) Convergence in offline setting if we cannot sample ?

  Lead to an algorithm that converge in offline setting : **Contrastive Policy Gradient**

II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models
c) Imitation Learning vs Preference Learning vs Reinforcement Learning
d) Offline Learning
   (1) Pretraining and Supervised Fine-Tuning : the cross entropy loss
   (2) Offline Preference Learning
e) Reinforcement Learning : from offline to online learning
   (1) Policy Gradient with KL regularised
   (2) Variance Reduction and Leave-One-Out baseline

I) The idea of baseline to reduce the variance in policy gradient

1) Finding a baseline $b(x)$ without introducing bias in the gradient ?

$$\mathbb{E}_{x\sim\rho,\ y\sim\pi(\cdot|x)}[b(x)\,\nabla \ln \pi(y \mid x)] = \mathbb{E}_{x\sim\rho}\left[b(x)\,\mathbb{E}_{y\sim\pi(\cdot|x)}[\nabla \ln \pi(y \mid x)]\right] \overset{\star}{=} 0,$$

$$\nabla J(\pi) = \mathbb{E}_{x\sim\rho,\ y\sim\pi(\cdot|x)}\left[\left(R^{\pi}_{\beta}(x,y) - b(x)\right)\nabla \ln \pi(y \mid x)\right].$$

Subtract the baseline!

$$\widehat{\nabla J}(\pi) = \frac{1}{N}\sum_{i=1}^{N}\left(R(x_i,y_i) - \beta \ln \frac{\pi(y_i \mid x_i)}{\pi_{\text{ref}}(y_i \mid x_i)} - b(x_i)\right)\nabla \ln \pi(y_i \mid x_i).$$

I) Reduce the variance in policy gradient while being unbiased ? RLOO

1) **Finding a baseline $b(x)$ without introducing bias in the gradient.**
2) **Find a baseline that reduce the variance of the loss function.**

$$\mathrm{Var}\big((A-b)g\big) = \mathbb{E}\big[(A-b)^2\, g^2\big] - \big(\mathbb{E}[A\, g]\big)^2.$$

Variance-minimizing constant baseline (exact, for scalar g)

$$b^*(x) = \frac{\mathrm{Cov}(A,g)}{\mathrm{Var}(g)} + \mathbb{E}[A].$$

Assuming the gradient g independent of A

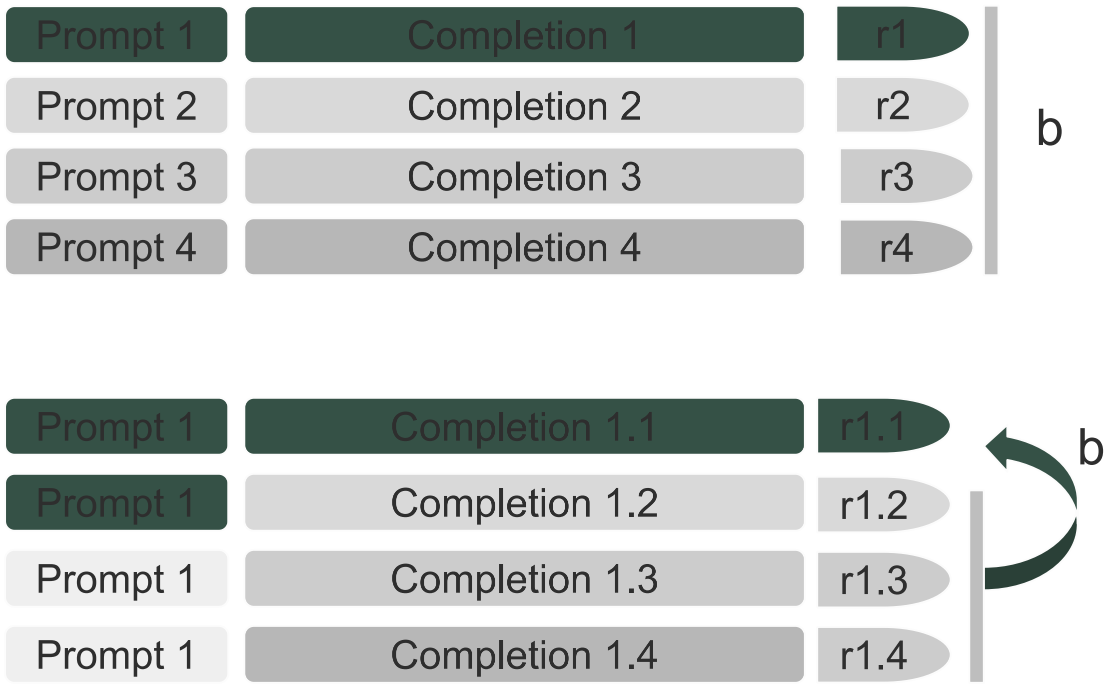
$$b^\star(x) \approx \mathbb{E}[A].$$

That is our target distribution we are trying to optimize, but we can find something close to the expectation !

I) An unbiased baseline : the Leave-One-Out baseline

Mean baseline

Leave-one-out baseline



I) RLOO or Reinforcement Learning Leave-One-Out

$$b_i^{\text{LOO}}(x, y^{1:K}) \;:=\; \frac{1}{K-1} \sum_{\substack{j=1 \\ j \neq i}}^{K} R_\beta^\pi(x, y^j).$$

$$\widehat{g}_{\text{RLOO}}(x) \;:=\; \frac{1}{K} \sum_{i=1}^{K} \Big(R_\beta^\pi(x, y^i) - b_i^{\text{LOO}}(x, y^{1:K}) \Big) \nabla \ln \pi(y^i \mid x).$$

I) Why RLOO is a unbiased PG method ?

$$\mathbb{E}\left[\widehat{g}_{\mathrm{RLOO}}(x)\right] = \frac{1}{K}\sum_{i=1}^{K}\mathbb{E}\left[R^{\pi}_{\beta}(x, y^i)\,\nabla \ln \pi(y^i \mid x)\right] - \frac{1}{K}\sum_{i=1}^{K}\mathbb{E}\left[b_i^{\mathrm{LOO}}(x, y^{1:K})\,\nabla \ln \pi(y^i \mid x)\right]$$

$$= \underbrace{\mathbb{E}_{y\sim\pi(\cdot|x)}\left[R^{\pi}_{\beta}(x, y)\,\nabla \ln \pi(y \mid x)\right]}_{\text{desired gradient}} - \underbrace{\frac{1}{K}\sum_{i=1}^{K}\mathbb{E}\left[\frac{1}{K-1}\sum_{j\neq i} R^{\pi}_{\beta}(x, y^j)\,\nabla \ln \pi(y^i \mid x)\right]}_{0}.$$

i.i.d or exchangeable variable ★

$$\mathbb{E}\left[R^{\pi}_{\beta}(x, y^j)\,\nabla \ln \pi(y^i \mid x)\right] = \mathbb{E}\left[R^{\pi}_{\beta}(x, y^j)\right]\,\mathbb{E}\left[\nabla \ln \pi(y^i \mid x)\right].$$

Score identity ★ $\mathbb{E}_{y\sim\pi(\cdot|x)}\left[\nabla \ln \pi(y \mid x)\right] = \nabla \int \pi(y \mid x)\,dy = \nabla 1 = 0.$

$$\longrightarrow \quad \mathbb{E}\left[\widehat{g}_{\mathrm{RLOO}}(x)\right] = \mathbb{E}\left[R^{\pi}_{\beta}(x, y)\nabla \ln \pi(y \mid x)\right].$$

II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

II) The case of off policy Learning : Contrastive Policy Gradient

$$\ell_{\text{CoPG}}(y, y'; \pi) = \left(R^{\pi}_{\beta/2}(y) - R^{\pi}_{\beta/2}(y')\right) \ln \frac{\pi(y)}{\pi_{\text{ref}}(y)} + \left(R^{\pi}_{\beta/2}(y') - R^{\pi}_{\beta/2}(y)\right) \ln \frac{\pi(y')}{\pi_{\text{ref}}(y')}. \quad (5)$$

→ $$L(\pi) = \mathbb{E}_{y \sim \mu_1,\, y' \sim \mu_2}[\ell_{\text{CoPG}}(y, y'; \pi)].$$

→ $$L(\pi) = \mathbb{E}_{y \sim \mu_1}\left[\left(R^{\pi}_{\beta/2}(y) - \overline{R^{\pi}_{\beta/2}}^{\mu_2}\right) \ln \frac{\pi(y)}{\pi_{\text{ref}}(y)}\right] + \mathbb{E}_{y' \sim \mu_2}\left[\left(R^{\pi}_{\beta/2}(y') - \overline{R^{\pi}_{\beta/2}}^{\mu_1}\right) \ln \frac{\pi(y')}{\pi_{\text{ref}}(y')}\right].$$

$$\overline{R^{\pi}_{\beta/2}}^{\mu} = \mathbb{E}_{y \sim \mu}\left[R^{\pi}_{\beta/2}(y)\right],$$

With k=2 and sample from current policy CoPG=RLOO!!

RL Objective:

$$J(\pi) = \mathbb{E}_{x \sim \rho}\, \mathbb{E}_{y \sim \pi(\cdot|x)}\left[R(x, y) - \beta\, \text{KL}(\pi \,\|\, \pi_{\text{ref}})\right]$$

Theorem (informal):

“Given offline data generated with same support than the reference policy, the minimizer of CoPG loss is the ***same*** *as the classical RL objective”*

II) The case of off policy Learning : Importance Sampling

Offline setting can come from :

- Data sampled not from current policy but from a dataset.
- Not the same hyperparameters during sampling eg temperature etc..
- Sampled using modern engine such as vLLM etc..

$$J_{\text{off}}(\pi) = \mathbb{E}_{x\sim\rho}\left[\mathbb{E}_{y\sim\mu(\cdot|x)}\left[\underbrace{\frac{\pi(y|x)}{\mu(y|x)}}_{w(x,y)} R(x,y)\right] - \beta\,\mathrm{KL}(\pi(\cdot\mid x)\,\|\,\pi_{\text{ref}}(\cdot\mid x))\right].$$

$$\longrightarrow \quad \hat{J}_{\text{off}}(\pi) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\pi(y_i\mid x_i)}{\mu(y_i\mid x_i)} R(x_i,y_i) - \beta\,\mathrm{KL}(\pi(\cdot\mid x_i)\,\|\,\pi_{\text{ref}}(\cdot\mid x_i))\right].$$

Cons : very big variance, need support condition between target and sampling condition to works well

Short Recall all the losses

RL Objective:

$$J(\pi) = \mathbb{E}_{x\sim\rho}\,\mathbb{E}_{y\sim\pi(\cdot|x)}\left[R(x,y) - \beta\,\mathrm{KL}(\pi \,\|\, \pi_{\mathrm{ref}})\right]$$

$$R^{\pi}_{\beta}(x,y) = R(x,y) - \beta \ln \frac{\pi(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)}$$

Policy Gradient Theorem : $\nabla J(\pi) = \mathbb{E}_{\substack{x\sim\rho \\ y\sim\pi(\cdot\,|\,x)}}\left[\left(R(x,y) - \beta \log \frac{\pi(y\,|\,x)}{\pi_{\mathrm{ref}}(y\,|\,x)}\right)\nabla \log \pi(y\,|\,x)\right].$

➡ **Want to Converge in offline setting ? Use Contrastive Policy Gradient loss (CoPG) !**

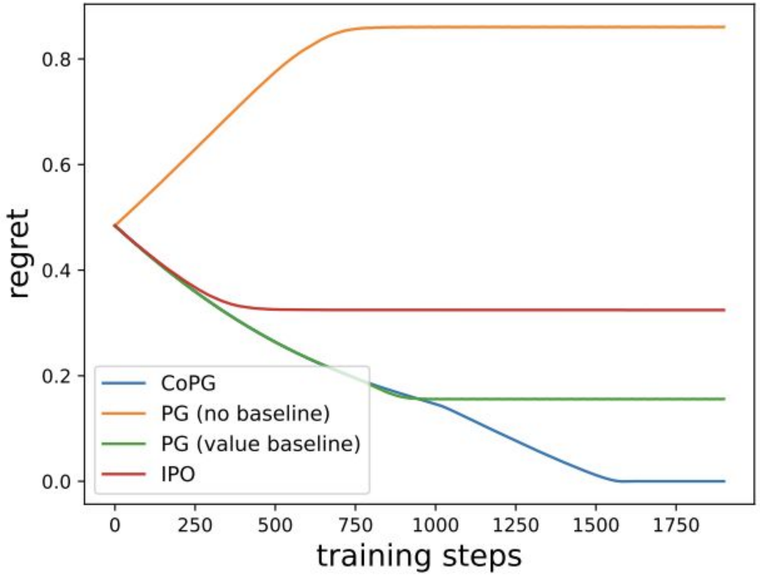
$$\ell_{\mathrm{CoPG}}(x,y,y';\pi) = \left(R^{\pi}_{\beta/2}(x,y) - R^{\pi}_{\beta/2}(x,y')\right) \ln \frac{\pi(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \left(R^{\pi}_{\beta/2}(x,y') - R^{\pi}_{\beta/2}(x,y)\right) \ln \frac{\pi(y' \mid x)}{\pi_{\mathrm{ref}}(y' \mid x)}.$$

➡ **Not having access to rewards but to pairs of completion ?** $\mathcal{L}_{\mathrm{DPO}} = -\log\sigma\left(\beta\left[\log\frac{\pi(y^+|x)}{\pi_{ref}(y^+|x)} - \log\frac{\pi(y^-|x)}{\pi_{ref}(y^-|x)}\right]\right)$

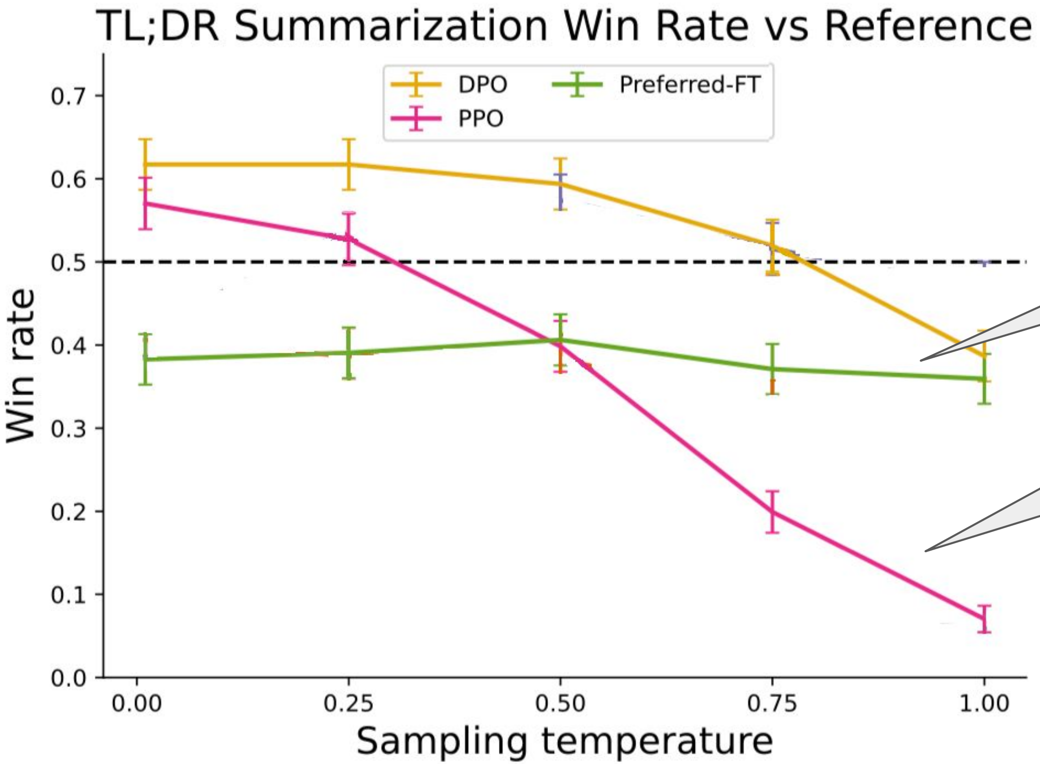
CoPG <u>converge in offline</u> setting without need of online generations while Classical PG with and without variance reduction fails. Same for DPO.

CoPG vs IPO (pref) vs PG in offline Setting (Bandit experiments)

- **Setting** : 3-armed bandit with rewards (2.5,2,1) starting from uniform reference policy.
- **Optimal policy :** $\pi_*(y) \propto \exp\left(R(y)/\beta\right)$
- **Metric** : $\text{regret} = J(\pi_*) - J(\hat{\pi}), \quad J(\pi) = \mathbb{E}_{y\sim\pi}[R(y)] - \beta\,\mathrm{KL}(\pi\|\pi_{\text{ref}})$



SFT vs DPO vs Classical PG algorithms

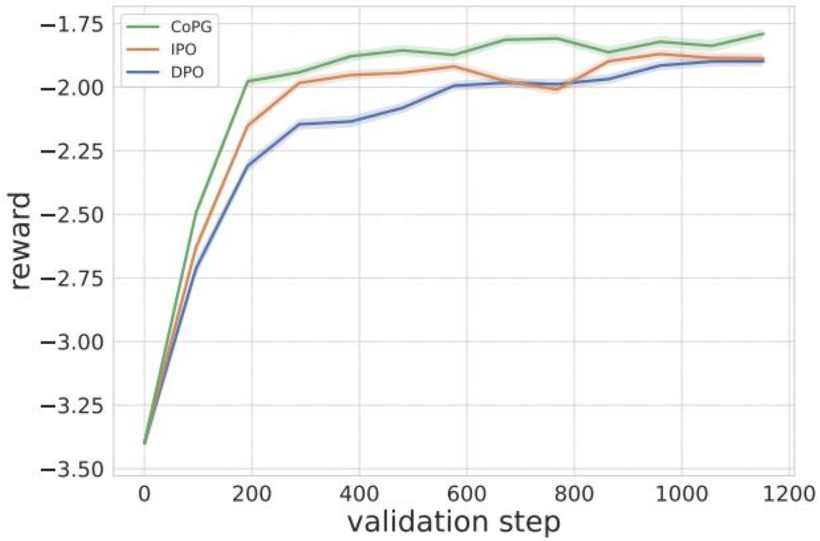


SFT on best in lower than DPO

Policy gradient is harder to tune on pref-models

Rafailov, Rafael, et al. "Direct preference optimization: Your language model is secretly a reward model." Advances in Neural Information Processing Systems 36 (2024).

CoPG vs DPO in offline Setting (LLM experiments)

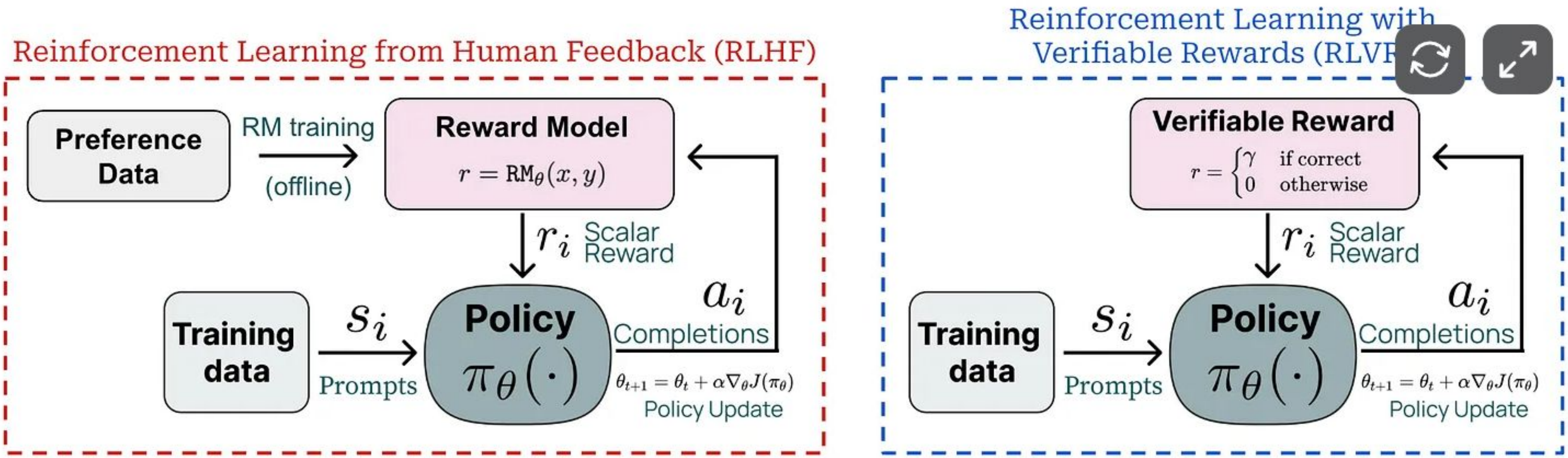


Experiments on 7B model on TL:DR dataset
https://arxiv.org/pdf/2406.19185

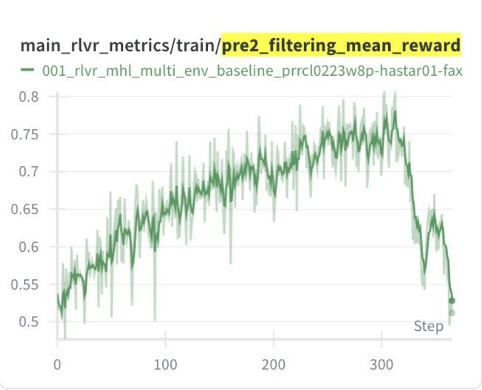
II. Training LLMs, losses, optimization : from SFT to Reinforcement Learning

a) How to Train an LLM ?
b) Small note on Modern Optimization for Large Models
c) Imitation Learning vs Preference Learning vs Reinforcement Learning
d) Offline Learning
    (1) Pretraining and Supervised Fine-Tuning : the cross entropy loss
    (2) Offline Preference Learning
e) Reinforcement Learning : from offline to online learning
    (1) Policy Gradient with KL regularised
    (2) Variance Reduction and Leave-One-Out baseline
    (3) Contrastive Policy Gradient and convergence in offline setting
    (4) **Stabilizing online RL with GRPO**

Two setting for Online RL but one big problem



Problem : Online RL suffer from bug variance and sometimes classical baseline is not enough for avoid collapse…



Trust Region ideas

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[A^\pi(x,y)\,\nabla_\theta \log \pi_\theta(y|x)\right]$$

$$A^\pi(x,y) = Q^\pi(x,y) - V^\pi(x).$$

When a single minibatch with large positive (or negative) advantages can push current policy **very far** from the previous policy. When the policy changes too much:

1. Your advantage estimates become “stale” (they were computed under the old policy).
2. The sampling distribution shifts, causing high variance / collapse.
3. With function approximation (neural nets), large steps can overshoot and degrade performance.

This is the motivation for **trust regions**: *only trust the local (small change) approximation of improvement*

Solution ? $$\max_\pi \mathbb{E}_{x,y\sim\pi_{\text{old}}}\left[\frac{\pi(y|x)}{\pi_{\text{old}}(y|x)} A_{\text{old}}(x,y)\right] \quad \text{s.t.} \quad \mathbb{E}_{x\sim\mathcal{D}}\left[D_{\text{KL}}(\pi_{\text{old}}(\cdot|x) \,\|\, \pi(\cdot|x))\right] \le \delta.$$

From Trust Region ideas to PPO using clipping

$$\max_{\pi} \ \mathbb{E}_{x,y\sim\pi_{\text{old}}}\left[\frac{\pi(y|x)}{\pi_{\text{old}}(y|x)}\,A_{\text{old}}(x,y)\right] \quad \text{s.t.} \quad \mathbb{E}_{x\sim\mathcal{D}}\left[D_{\text{KL}}(\pi_{\text{old}}(\cdot|x)\,\|\,\pi(\cdot|x))\right] \le \delta.$$

$$r(x,y) \;=\; \frac{\pi(y|x)}{\pi_{\text{old}}(y|x)}.$$

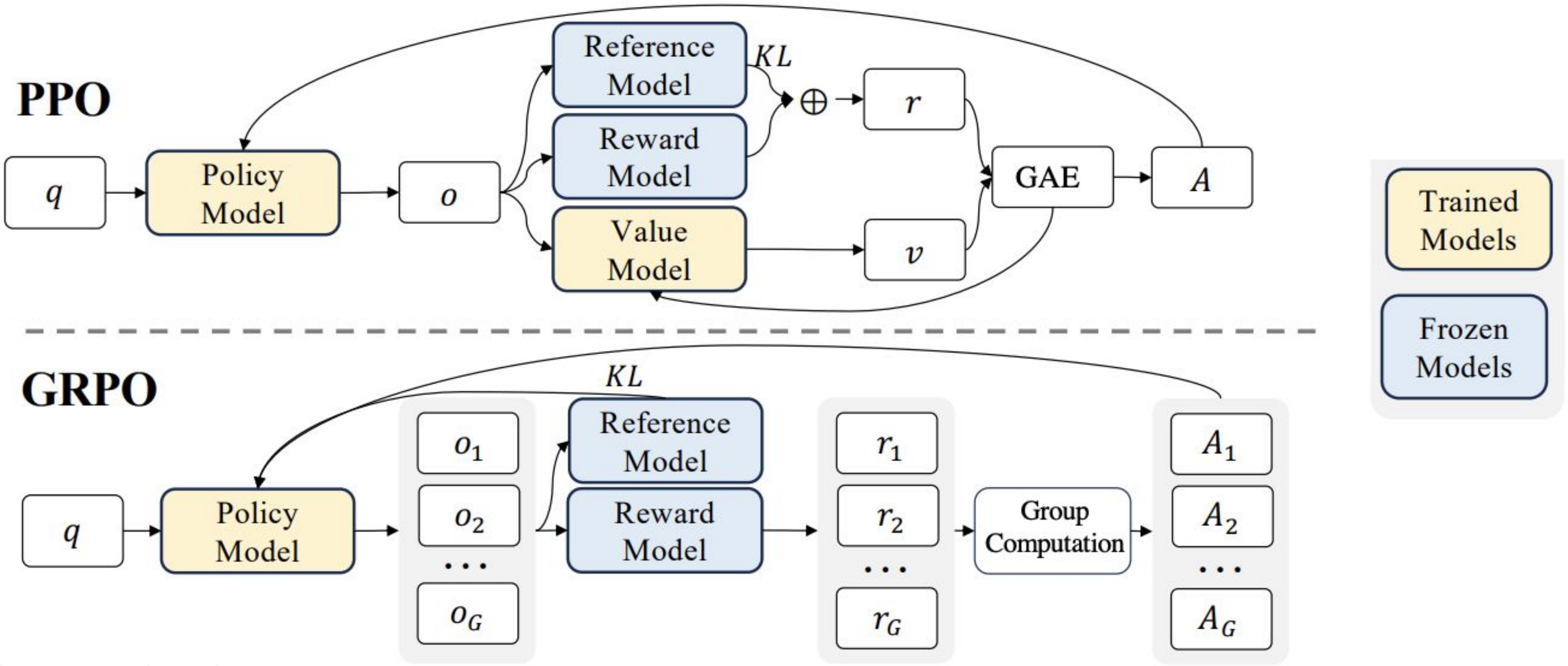
Interpretation:

- If $A_{\text{old}}(x,y) > 0$ we want $r(x,y) > 1$ (increase probability).
- If $A_{\text{old}}(x,y) < 0$ we want $r(x,y) < 1$ (decrease probability).
- But the KL constraint prevents from becoming extreme

$$\mathcal{L}_{PPO}(\pi) = -\mathbb{E}_{x,y\sim\pi_{\text{old}}}\left[\min\left(r(x,y)\,A_{\text{old}}(x,y),\ \text{clip}(r(x,y),\,1-\epsilon,\,1+\epsilon)\,A_{\text{old}}(x,y)\right)\right].$$

$$A^{\pi}(x,y) \;=\; Q^{\pi}(x,y) - V^{\pi}(x).$$

II) GRPO objective : remove target network from PPO



https://arxiv.org/abs/2402.03300

II) GRPO objective

$$\hat{A}(x,y) = \frac{R_\beta(x,y) - \mu(x)}{\sigma(x) + \varepsilon}, \quad \mu(x) = \mathbb{E}_{y' \sim \pi_{\text{old}}(\cdot|x)}[R_\beta(x,y')], \quad \sigma(x) = \sqrt{\text{Var}_{y' \sim \pi_{\text{old}}(\cdot|x)}[R_\beta(x,y')]}.$$

$$\mathcal{L}_{\text{GRPO-clip}}(\pi_\theta) = -\mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{y \sim \pi_{\text{old}}(\cdot|x)} \left[\min\Big(r(x,y)\,\hat{A}(x,y),\ \text{clip}(r(x,y), 1-\epsilon, 1+\epsilon)\,\hat{A}(x,y) \Big) \right].$$

- Not supposed to converge in offline/online
- Clipping is added for stability
- Biased baseline because of the standard deviation normalisation
- Possible to add Importance Sampling between trainer and sampler to improve stability too.

To go further ? Some reading !

SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training
https://arxiv.org/abs/2501.17161

Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model?
https://arxiv.org/pdf/2504.13837

DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models

https://arxiv.org/abs/2402.03300

Conclusion

- RL is now central in LLM fine tuning.
- It allow better generalization compared to SFT, while very difficult from an infra perspective.
- Online RL allow to self-correct mistakes, as long as we are able to deal with the variance of PG.
- DPO is useful when there is no ground-truth e.g. traduction
- TP : Code DPO, PG PG gradient with baseline, CopG in offline setting !

A small note on the use of AI

This is simply my personal opinion and this does not engage my company :

- AI is great for coding, translation, education, medicine BUT…

- AI is becoming politic and a sovereignty question many countries and we should care more about training a more responsible AI , safety etc…

  We must avoid **mass domestic surveillance, fully autonomous weapons using AI etc…**

  Example of company/institution which do not care at all : Palantir (mass domestic surveillance), xAI (racist, misogynist AI), US Gouvernement.

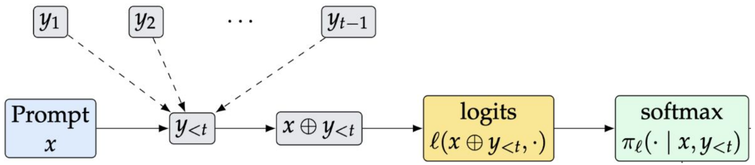
  This is possible to care about this. Anthropic : not using AI for war : here . As future engineer/researcher/assistant professor :

  Train your favorite model with all the tools you can use, but always care also about the data, safety, the finality of your model and not only pure performances…

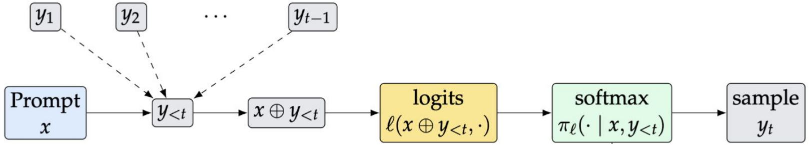
Thank You

Questions?

Online Supervised Fine-tuning



$$\mathcal{L}_{\text{SFT}} = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \left[-\sum_{t=1}^{T_y} \log \pi_\ell(y_t \mid x, y_{<t}) \right].$$



$$\mathcal{L}_{\text{SFT}}^{\text{on-policy}} = \mathbb{E}_{(x,y^*)\sim\mathcal{D}_{\text{SFT}}} \, \mathbb{E}_{\tilde{y}_{1:T}\sim\pi_\ell(\cdot\mid x)} \left[-\sum_{t=1}^{T_{y^*}} \log \pi_\ell(y_t^* \mid x, \tilde{y}_{<t}) \right].$$

Pros

- **Reduces exposure bias**: the model is trained on the states it actually visits (its own prefixes), not only on expert prefixes.
- **More robust to its own errors** : learns how to recover from suboptimal earlier tokens.
- **Still a supervised loss**: no reward/reward model, gradients are well-behaved cross-entropy.
- **Conceptually between SFT and RL**: closer to on-policy learning without full Reinforcement Learning complexity.

Cons

- **More expensive** : you must sample rollouts from the current model
- **Mismatch with reference**: if sampled prefixes diverge a lot from the reference answer, the targets can become incoherent or very low-probability, increasing variance.
- **Still limited by labels** : you're not exploring arbitrary behaviors; you're still trying to imitate given references.

Still limited by labels ? ➡ Use reward function and not ground-truth!

Mismatch with reference ? ➡ Use negative reward on bad sampled trajectory and not only good completion !

RL algorithms

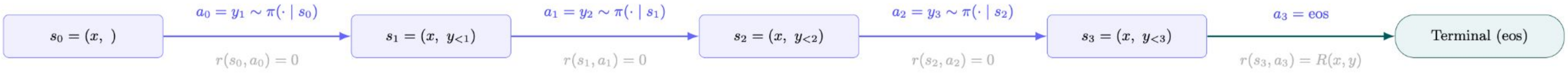
	Classical RL	**RL for LLMs**
Policy Gradient	REINFORCE, PPO, TD3 etc..	PPO, CoPG, GRPO etc…
Bellman Residual/Bootstrapping approaches	SARSA, Q- learning based method ? DQN, etc…	**?**

Could we derive algorithm based on Bellman equation for LLMs,

that take in to account LLMs specific characteristics? A method with better credit assignment ?

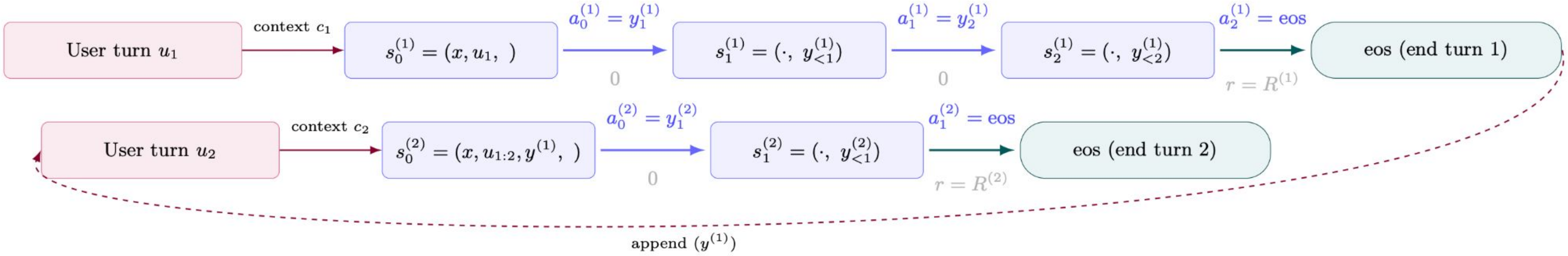
In RL you observe a *delayed* scalar signal (reward). Credit assignment is the problem of figuring out **which past state–action choices "deserve credit or blame"** for that reward.

LLM as Markov Decision Processes



State: $s_t = (x, y_{<t})$
Action: $a_t = y_t$ (next token)
Policy: $\pi(a_t \mid s_t)$
Transition: $s_{t+1} = (x, y_{<t} \oplus a_t)$
Reward: $r(s_t, a_t) = \begin{cases} R(x, y), & a_t = \text{eos or } t = T_{\max}, \\ 0, & \text{otherwise.} \end{cases}$
Return: $R(x, y) = \sum_{t=1}^{|y|} \gamma^{t-1} r(s_t, a_t)$ (typically $\gamma = 1$).

Multi-turn RL



Turn context: $c_k = (x, u_{1:k}, y_{1:k-1})$ **Within-turn state:** $s_t^{(k)} = (c_k,\ y_{<t}^{(k)})$
Action: $a_t^{(k)} = y_t^{(k)}$; token reward 0 except at eos **Between turns:** user supplies u_{k+1}
Episode return: $\sum_{k=1}^{K} R^{(k)}$ or $R(x, u_{1:K}, y_{1:K})$

RL Recall on Credit assignment

State: $s_t = (x, y_{<t})$
Action: $a_t = y_t$ (next token)
Policy: $\pi(a_t \mid s_t)$
Transition: $s_{t+1} = (x, y_{<t} \oplus a_t)$
Reward: $r(s_t, a_t) = \begin{cases} R(x,y), & a_t = \text{eos or } t = T_{\max}, \\ 0, & \text{otherwise.} \end{cases}$
Return: $R(x,y) = \sum_{t=1}^{|y|} \gamma^{t-1} r(s_t, a_t)$ (typically $\gamma = 1$).

TD learning = Bellman credit signal

$$\textbf{State-value Bellman: } v_\pi(s_t) = \mathbb{E}_\pi\left[r(s_t, a_t) + \gamma\, v_\pi(s_{t+1}) \mid s_t\right].$$

Monte Carlo (full-return target)

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r(s_{t+k}, a_{t+k}), \qquad V(s_t) \leftarrow V(s_t) + \alpha\left(G_t - V(s_t)\right).$$

TD error (credit signal)

$$\delta_t = r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t), \qquad V(s_t) \leftarrow V(s_t) + \alpha\, \delta_t.$$

$\mathbb{E}[\delta_t \mid s_t, a_t] = q_\pi(s_t, a_t) - v_\pi(s_t) = A_\pi(s_t, a_t)$, so TD error is an unbiased per-timestep advantage/credit si

$$\text{SARSA: } Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right],$$

$$\text{Q-learning: } Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right].$$

Unbiased but higher variance; no one-step bootstrapping. TD uses the Bellman one-step target and its residual as the immediate credit signal that propagates reward backward.

Back to basics : KL regularised Bellman equation

Theorem 1

Let $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ be the unique function satisfying, for any admissible (s_t, a_t, s_{t+1}),

2

$$\left\{ q(s_t, a_t) = r(s_t, a_t) + \gamma\beta \ln \sum_{a' \in \mathcal{A}} \pi_{ref}(a'|s_{t+1}) \exp \frac{q(s_{t+1}, a')}{\beta}. \right\}$$

Then, the unique optimal policy maximizing RL objective satisfies

$$\pi_*(a_t|s_t) = \frac{\pi_{ref}(a_t|s_t) \exp \frac{q(s_t,a_t)}{\beta}}{\sum_{a \in \mathcal{A}} \pi_{ref}(a|s_t) \exp \frac{q(s_t,a)}{\beta}}.$$

Naive L2 residual approach ?

$$\rightarrow \quad L_{\text{try1}}(q) = \mathbb{E}_{x,y\sim\mathcal{D}}\left[\sum_{s_t,a_t \in (x,y)} \left(r(s_t, a_t) + \gamma\beta \ln \sum_{a' \in \mathcal{A}} \pi_{\text{ref}}(a'|s_{t+1}) \exp \frac{q(s_{t+1}, a')}{\beta} - q(s_t, a_t) \right)^2\right]$$

Issues with naive L2 residual approach

$$L_{\text{try1}}(q) = \mathbb{E}_{x,y\sim\mathcal{D}}\left[\sum_{s_t,a_t\in(x,y)}\left(r(s_t,a_t) + \gamma\beta\ln\sum_{a'\in\mathcal{A}}\pi_{\text{ref}}(a'|s_{t+1})\exp\frac{q(s_{t+1},a')}{\beta} - q(s_t,a_t)\right)^2\right]$$

- **Sampling/Inference problem**: need to load and infer reference and policy networks for sampling

$$\pi(a_t|s_t) \propto \exp\frac{q(s_t,a_t) + \beta\ln\pi_{ref}(a_t|s_t)}{\beta}.$$

→ **Easing sampling**: eliminates the need to load and infer from the reference models.

Mathematically : a change of variable/shift

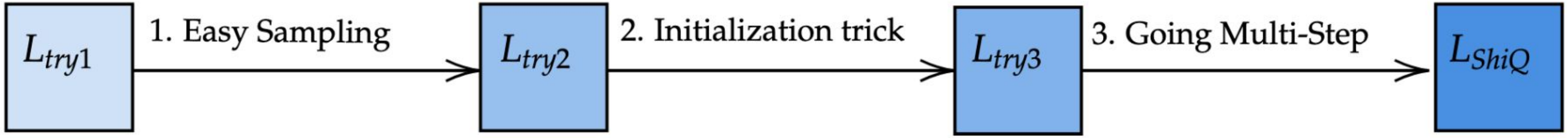
- **Bad initialisation**: the loss is not zero at the reference policy with no rewards.

→ **Initialization trick**: leverages the pretrained policy for a smarter Q-learning start.

Mathematically : a second change of variable/shift

- **Sparse rewards**: difficult to learn from only final rewards/sparce rewards

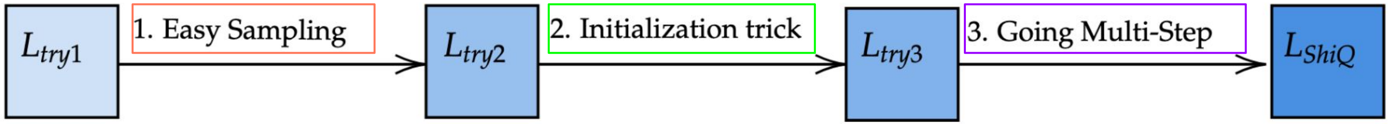
→ **Going Multi-step**: propagates rewards more effectively across multiple steps.



ShiQ for Shifted Q !

$$L_{\text{try1}}(q) = \mathbb{E}_{x,y\sim\mathcal{D}}\left[\sum_{s_t,a_t\in(x,y)}\left(r(s_t,a_t) + \gamma\beta\ln\sum_{a'\in\mathcal{A}}\pi_{\text{ref}}(a'|s_{t+1})\exp\frac{q(s_{t+1},a')}{\beta} - q(s_t,a_t)\right)^2\right]$$

change of variable/shift : $\pi(a_t|s_t) = \exp\frac{q(s_t,a_t)+\beta\ln\pi_{ref}(a_t|s_t)}{\beta}.$



$$L_{\text{ShiQ}}(\ell) = \mathbb{E}_{(x,y)\in\mathcal{D}}\left[\sum_{t=1}^{|y|}\left(\sum_{k=t}^{|y|}\gamma^{k-t}\left(r(s_k,a_k) - \beta\ln\frac{\pi_\ell(a_k\mid s_k)}{\pi_{\text{ref}}(a_k\mid s_k)}\right) - \beta\left(v_\ell(s_t) - v_{\text{ref}}(s_t)\right)\right)^2\right].$$

Going multi-step

Add a term, for good initialization

Relation between policy, logits and value function in regularised bellman equation :

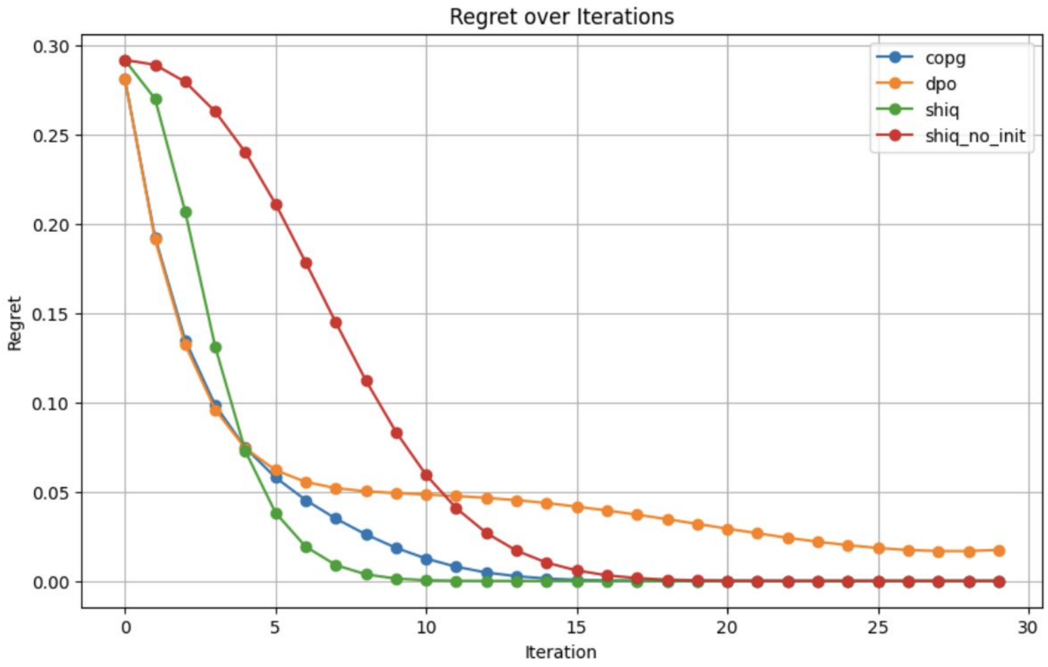
$$\pi_\ell(y_t|x,y_{<t}) = \exp(\ell(x\oplus y_{<t},y_t) - v_\ell(x\oplus y_{<t})) \text{ with } v_\ell(x\oplus y_{<t}) = \ln\sum_{w\in\mathcal{V}}\exp\ell(x\oplus y_{<t},w)$$

Theorem (informal):

*"Given offline data generated with same support than the reference policy, the minimizer of ShiQ loss is the **same** as the classical RL objective"*

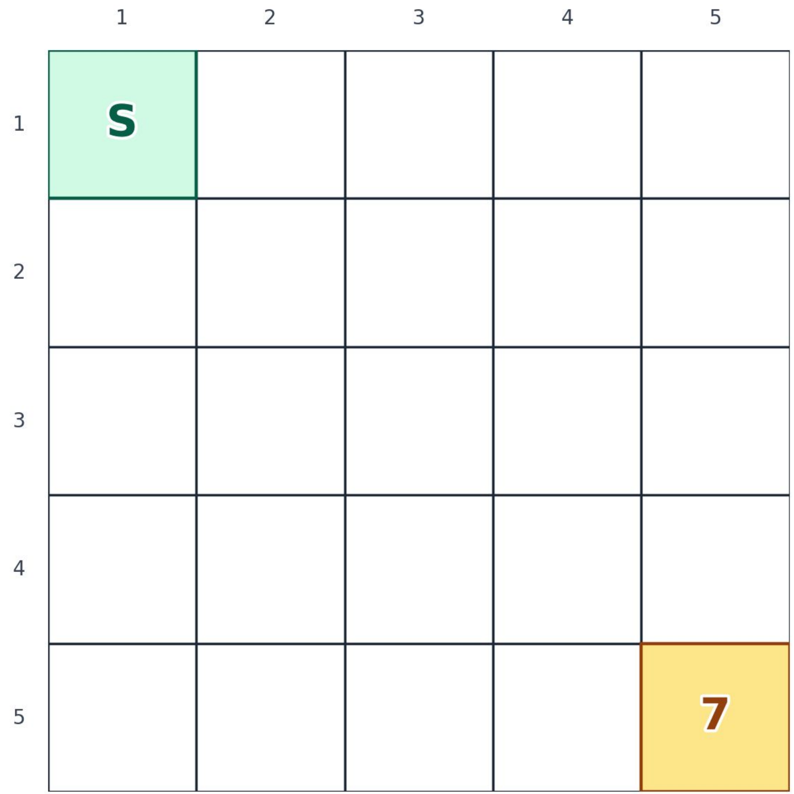
3-armed Bandit setting

- **Setting** : 3-armed bandit with rewards (2.5,2,1) starting from uniform reference policy.
- **Optimal policy :** $\pi_*(y) \propto \exp\left(R(y)/\beta\right)$
- **Metric** : $\text{regret} = J(\pi_*) - J(\hat{\pi}), \quad J(\pi) = \mathbb{E}_{y\sim\pi}[R(y)] - \beta\,\mathrm{KL}(\pi\|\pi_{\text{ref}})$

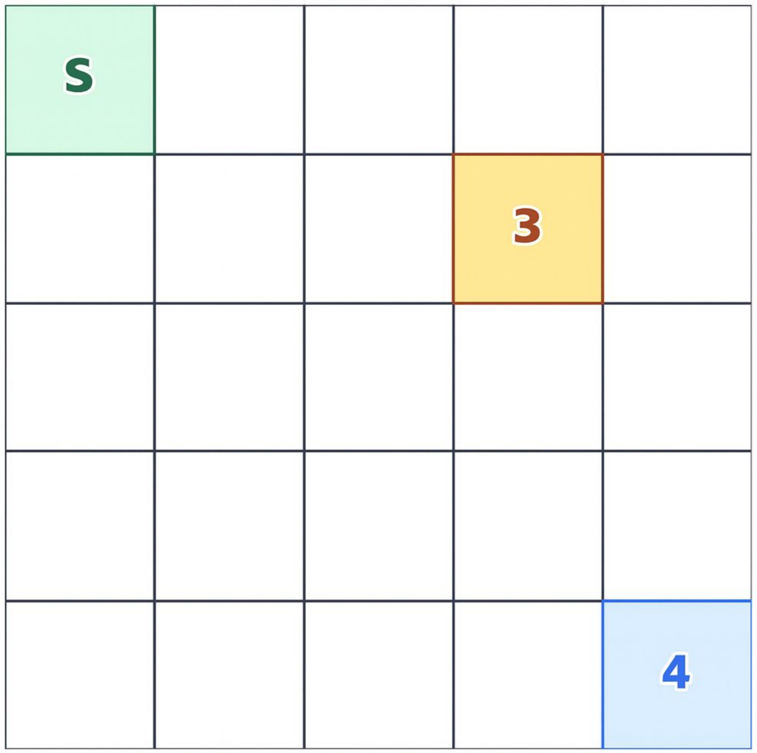


5 x 5 grid experiments

1:

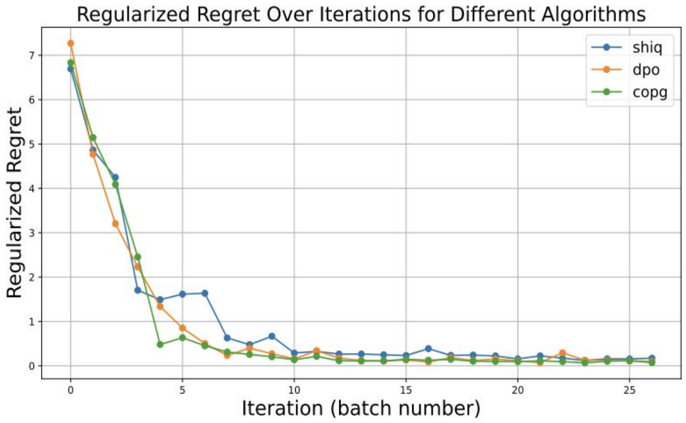


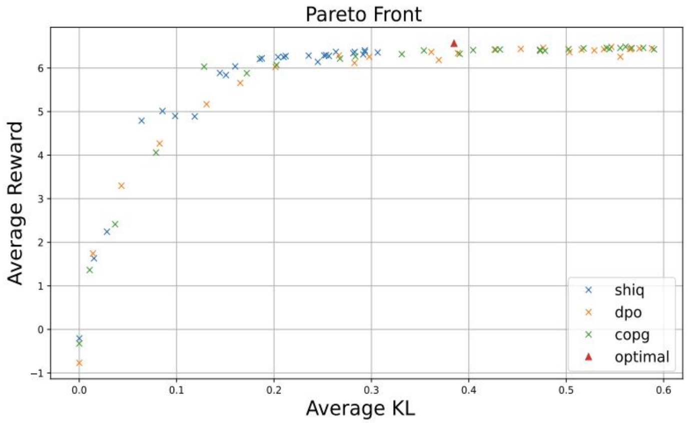
2:



Final reward setting

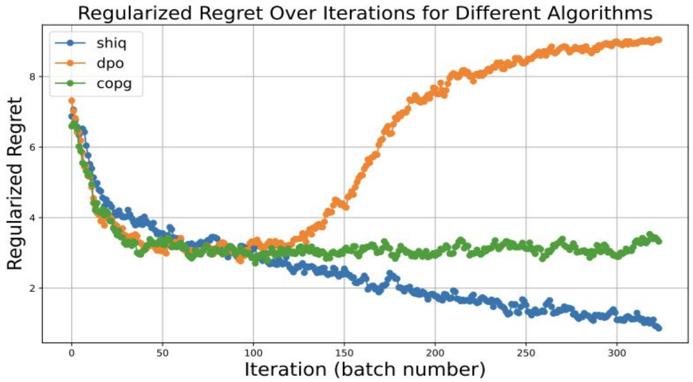
1:

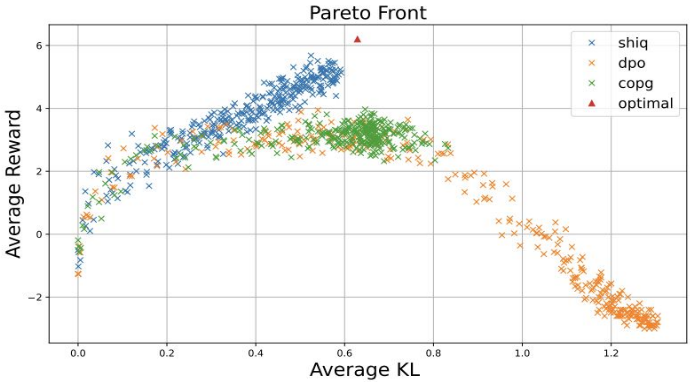




Fine-grained reward setting

2:



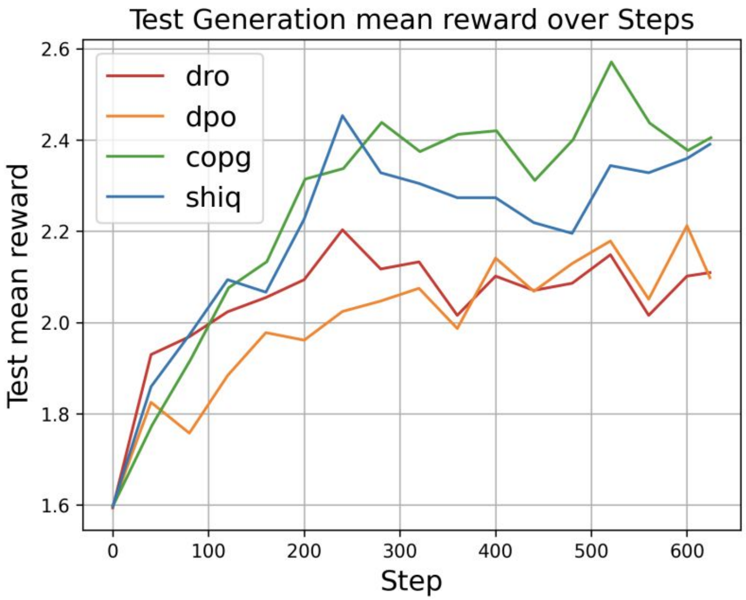


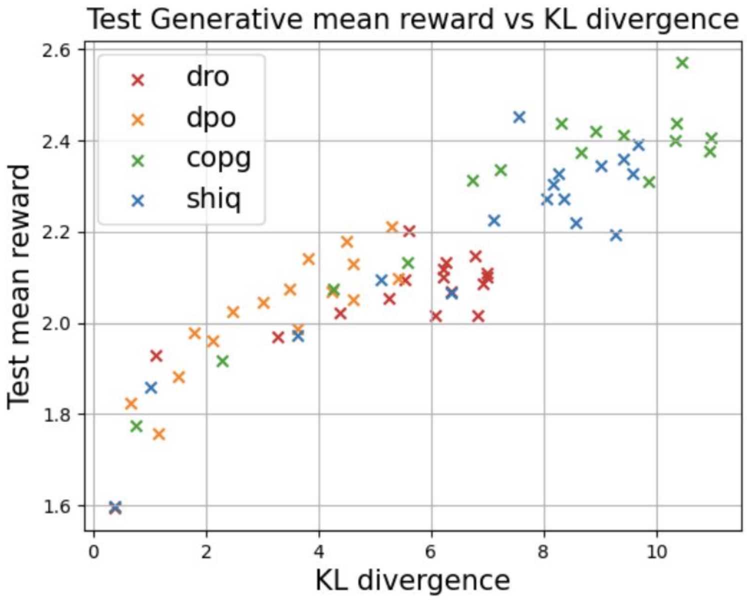
Multi-turn DPO , Multi-turn CoPG Losses as a baseline.

$$\mathcal{L}_{\mathrm{DPO}}(\pi;\mathcal{D}) = -\,\mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\log\sigma\left(\sum_{t=0}^{N-1}\beta\log\frac{\pi(a_t^1 \mid s_t^1)}{\pi_{\mathrm{ref}}(a_t^1 \mid s_t^1)} \;-\; \sum_{t=0}^{M-1}\beta\log\frac{\pi(a_t^2 \mid s_t^2)}{\pi_{\mathrm{ref}}(a_t^2 \mid s_t^2)}\right)\right].$$

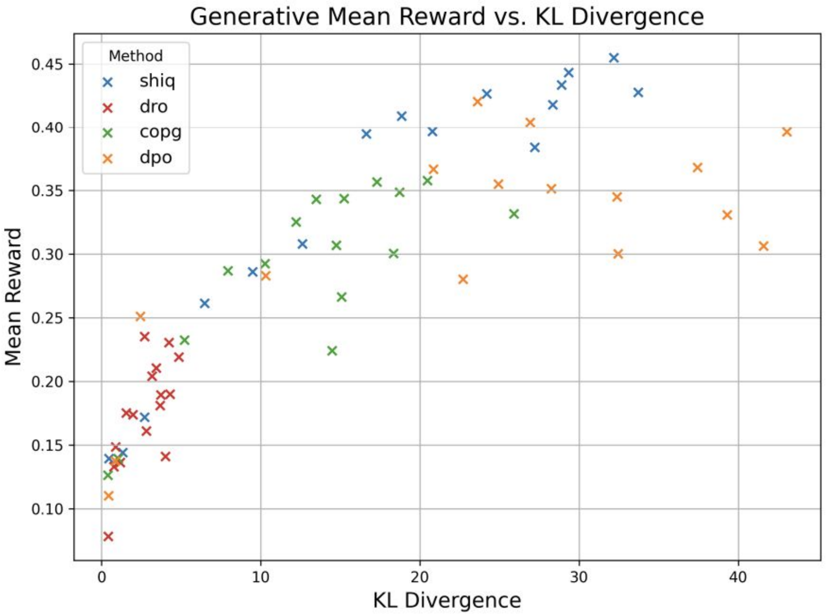
$$\mathcal{L}_{\mathrm{CoPG}}(\pi;\mathcal{D}) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\left(\sum_{t=0}^{N-1}\beta\log\frac{\pi(a_t^1 \mid s_t^1)}{\pi_{\mathrm{ref}}(a_t^1 \mid s_t^1)} \;-\; \sum_{t=0}^{M-1}\beta\log\frac{\pi(a_t^2 \mid s_t^2)}{\pi_{\mathrm{ref}}(a_t^2 \mid s_t^2)} \;+\; \left(R(\tau_2) - R(\tau_1)\right)\right)^2\right].$$

Single-turn experiments on HH datasets with 7B models





Multi-turns experiments on BFCL-v3 datasets with 7B models



Conclusion on ShiQ algorithm (for LLMs)

- **Improved initialization**: leverages pre-trained policy for a *smarter Q-learning initialization.*
- **Multi-step extension** : ShiQ propagates rewards more effectively across multiple steps.
- **Single Network Requirement:** Using the LLM as the policy *without a separate value network.*
- **Single Trajectory Requirement**: *Eliminating the need for pairs of completions.*
- **ShiQ can adapt to both single and multi-turn RL.** *Fine grained allocations.*
- **ShiQ could be used on other field/application ?**
    - **Classical Offline RL?**
    - **Robotics ?**
    - **For Distillation ?**

Ablations on HH