UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN

Escuela de
Ciencias Aplicadas
e Ingeniería

Artificial Intelligence

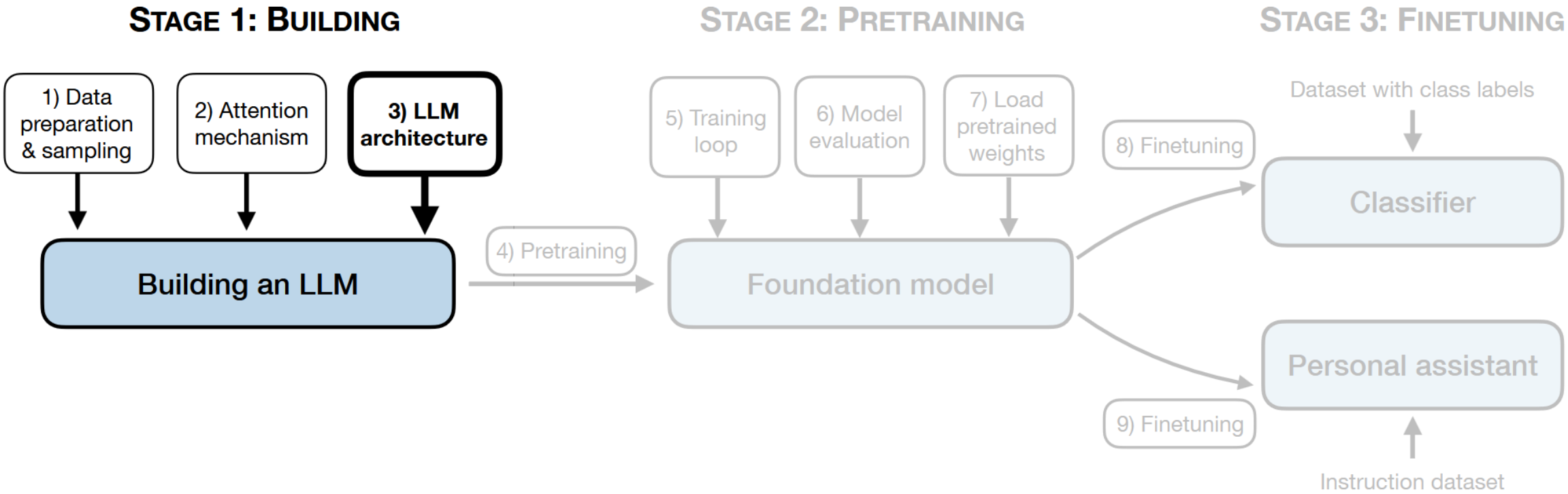
Lecture08 – RLHF

Agenda

UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN
Escuela de Ciencias Aplicadas e Ingeniería

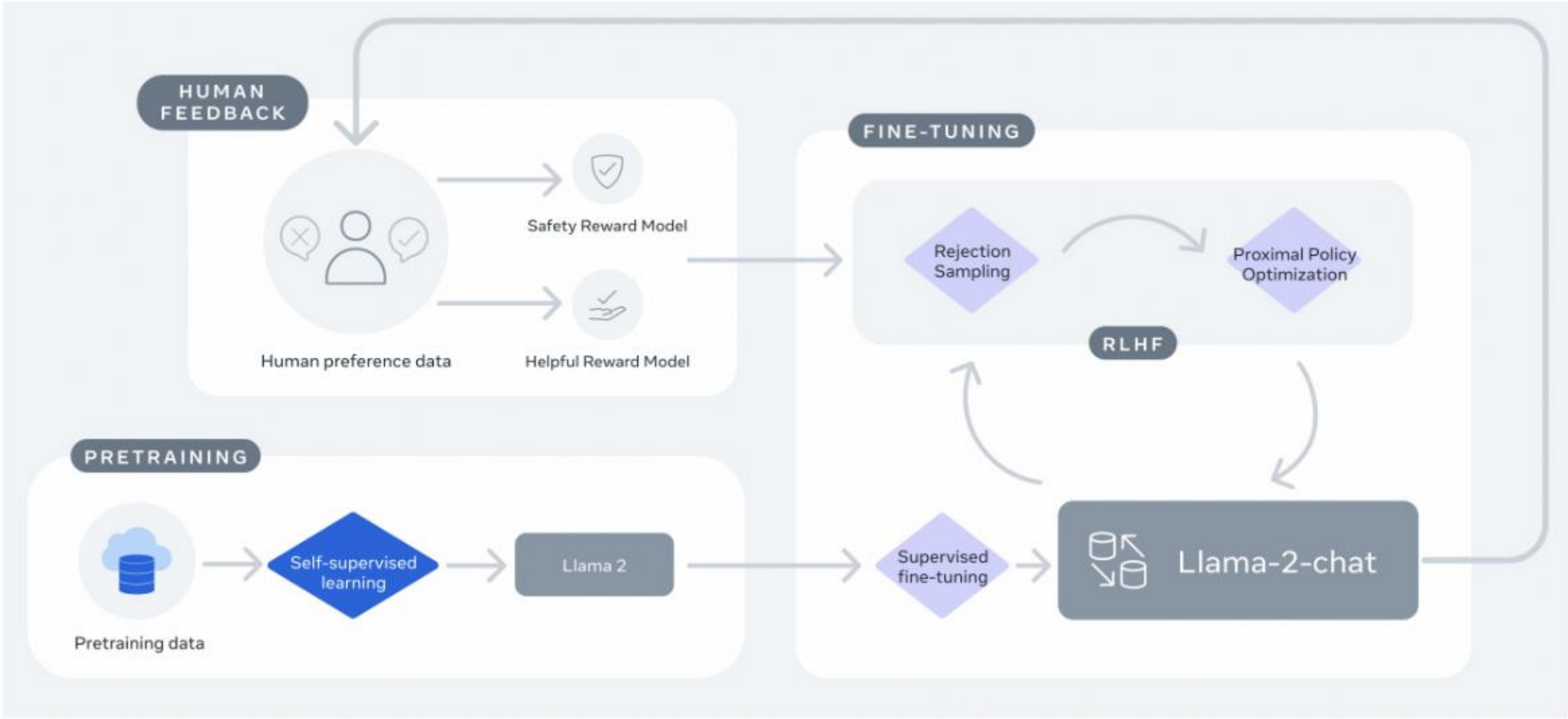
1. Summary of LlaMA architecture
2. Summary of SFT
3. RLHF
4. PPO
5. GRPO (Deep Seek)
6. DPO (Llama)

Building LLMs

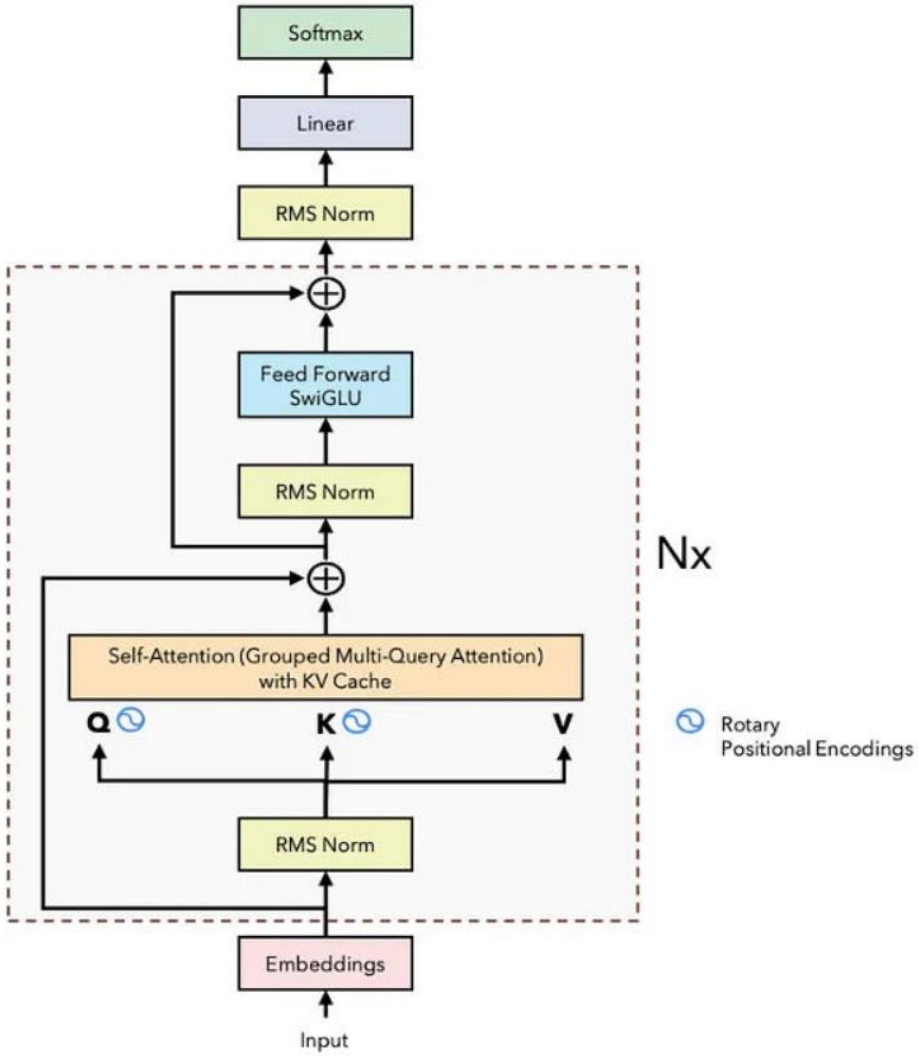


Overview of LLMs Training

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



LlaMA



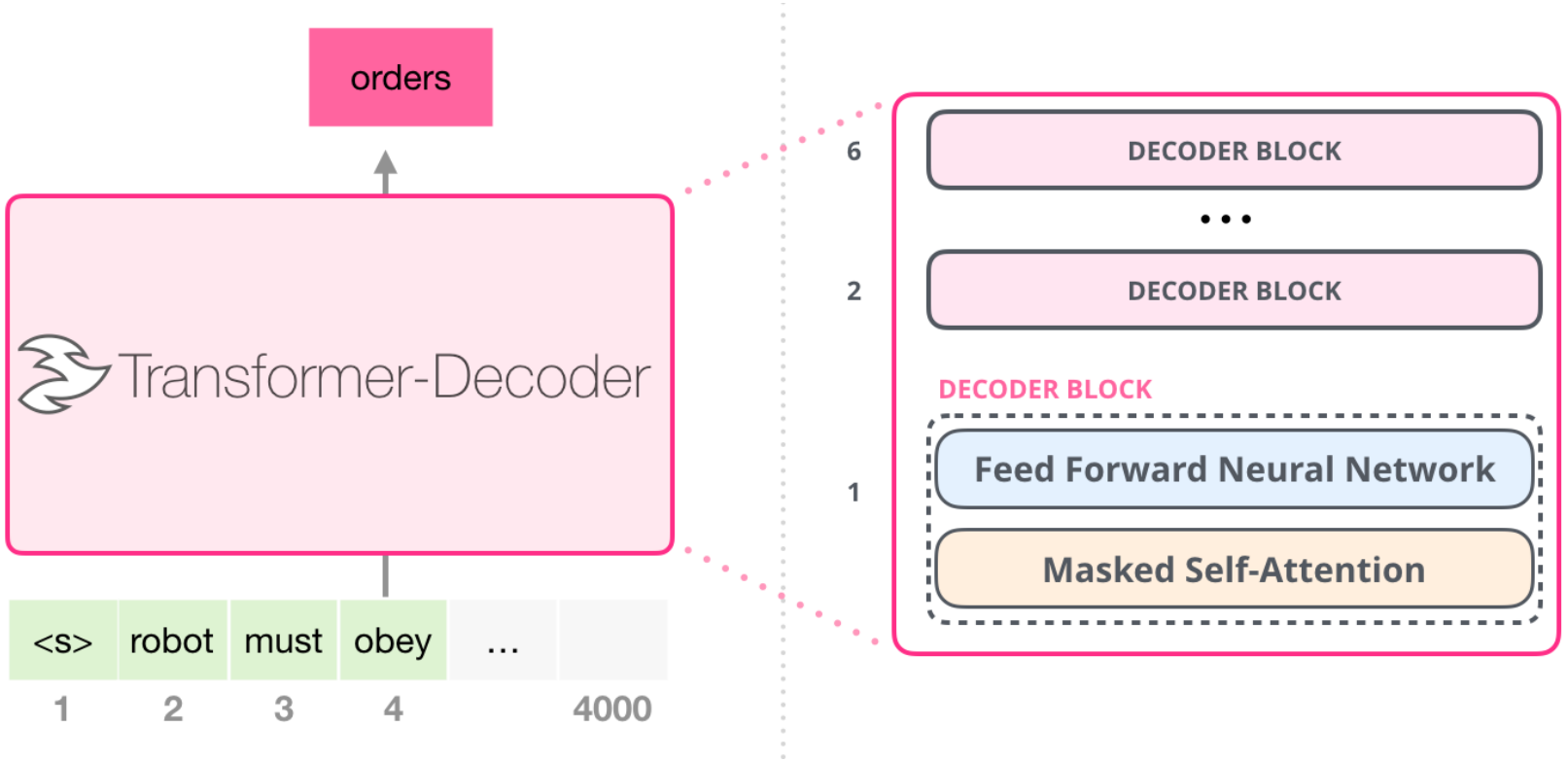
LLaMA

	Llama (Feb 2023)				Llama 2 (Jul 2023)				Llama 3 (Apr 2024)			
# params	7B	13B	33B	65B	7B	13B	34B	70B	8B			70B
# training tokens	1T	1T	1.4T	1.4T	2T	2T	2T	2T	15T			15T
hidden embed dim	4096	5120	6656	8192	4096	5120		8192	4096			8192
# attn heads	32	40	52	64	32	40		64	32			64
# attn layers	32	40	60	80	32	40		80	32			80
attention	MHA	MHA	MHA	MHA	MHA	MHA	GQA	GQA	GQA			GQA
# kv heads	32	40	52	64	32	40		8	8			8
mlp intermediate size	11008	13824	17920	22016	11008	13824		28672	14336			28672
context	2048				4096				8192			
tokenizer	BPE sentencepiece				BPE sentencepiece				BPE tiktoken			
token vocabulary	32000				32000				128256			
fine-tuned models	-				Llama-2-Chat (Jul 2023) Code Llama (Aug 2023)				Llama-3-Instruct (Apr 2024)			

BPE: Byte Pair Encoding Not released by Meta
MHA: Multi-Head Attention
GQA: Grouped-Query Attention

https://devopedia.org/llama-llm

Transformer decoder



Embeddings

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

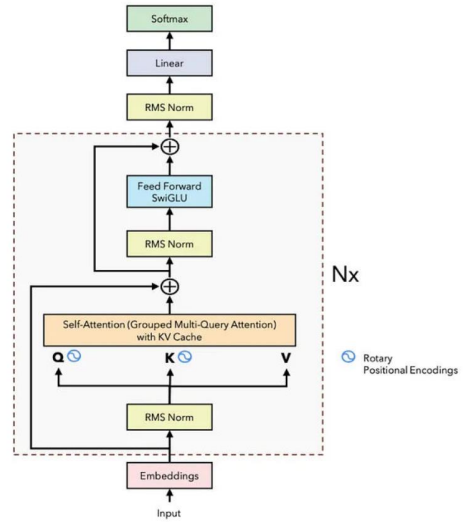
1️⃣ Understanding the Learnable Parameters in the Embedding Layer

For a vocabulary of **32,000** tokens and an embedding dimension of **5120**, the **embedding matrix** has the shape:

$$E \in \mathbb{R}^{32000 \times 5120}$$

Each token is represented as a **5120-dimensional vector**, meaning the total number of parameters in the embedding layer is:

$$32000 \times 5120 = 163,840,000$$



LLaMA

2️⃣ Where Are These Parameters Used?

- **At the input**: Converts token indices into dense vectors before being processed by the Transformer layers.
- **At the output**: The final hidden states are mapped back to vocabulary logits using the same matrix or a separate **output projection**.

In some models, the embedding matrix is **tied** with the output projection to reduce parameter count, but in LLaMA-2, they are **separate**, meaning **another 163.84M parameters** for output.

RMSNorm

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1 Understanding RMSNorm

Given an input x of shape (B, T, D), where:

- B = Batch size
- T = Sequence length
- D = Embedding dimension (**5120** in LLaMA-2)

RMSNorm computes the normalized activation:

$$\hat{x} = \frac{x}{\text{RMS}(x)}$$

where the **Root Mean Square (RMS)** is:

$$\text{RMS}(x) = \sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2 + \epsilon}$$

This ensures that the activations have unit norm across dimensions.

2 Learnable Parameters in RMSNorm

Instead of an **affine transformation (scale + shift) like LayerNorm**, RMSNorm **only applies a learnable scaling factor** γ:

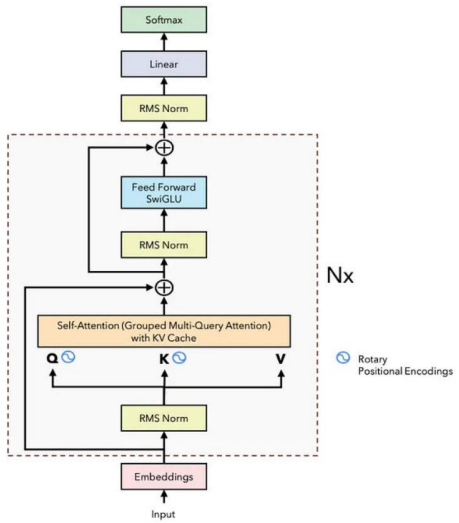
$$y = \gamma \odot \hat{x}$$

where:

- γ is a **learnable parameter vector** of size D (**5120** in LLaMA-2).
- There is **no bias term** (unlike LayerNorm).

Thus, RMSNorm has **one learnable parameter per dimension**, meaning:

$$\text{Number of parameters} = D = 5120$$



LLaMA

Attention

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1️⃣ Understanding the Learnable Parameters in Attention

For an input x of shape (B, T, D), where:

- B = Batch size
- T = Sequence length
- D = Embedding dimension (**5120** in LLaMA-2)
- **Number of attention heads** = **40**
- **Head dimension** = **128** (so $40 \times 128 = 5120$)

Each attention layer has **four projection matrices**:

Step 1: Q, K, V Projections

Each token in the sequence is **projected** to three different spaces (Q, K, V) using independent weight matrices:

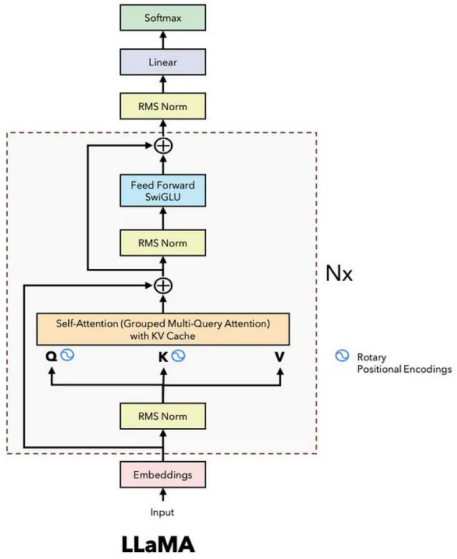
$$Q = xW_Q, \quad K = xW_K, \quad V = xW_V$$

where:

- W_Q, W_K, W_V each have shape $(5120, 5120)$.

Thus, total parameters for Q, K, and V:

$$3 \times (5120 \times 5120) = 3 \times 26,214,400 = 78,643,200$$



LLaMA

Step 2: Output Projection

Once self-attention is computed, the result is **projected back** into the embedding dimension:

$$O = \text{Attention}(Q, K, V)W_O$$

where:

- W_O has shape $(5120, 5120)$.
- Total parameters for the output projection:

$$5120 \times 5120 = 26,214,400$$

SwiGLU

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1 Step 1: First Up Projection (Expand to 13824)

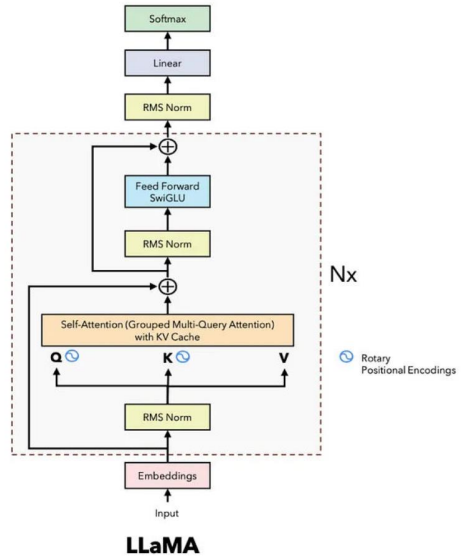
- Expands embedding from **5120 → 13824**.
- Uses **two linear projections**:
  - $W_1 : (5120, 13824)$ → Generates h_1.
  - $W_2 : (5120, 13824)$ → Generates h_2.
- Each has a bias term.

2 Step 2: SwiGLU Activation

- Applies **Swish activation** on h_1:
  $\text{Swish}(x) = x \cdot \sigma(x)$, where $\sigma(x)$ is the sigmoid function.
- Performs **element-wise multiplication** with h_2:
  $h = \text{Swish}(h_1) \odot h_2$.
- Introduces **gating mechanism** for better expressivity.

3 Step 3: Down Projection (Reduce to 5120)

- Projects **back to 5120** using:
  $W_3 : (13824, 5120)$.
- Returns output to **original embedding size**.



LLaMA

4 Why SwiGLU?

- **Improves gradient flow** (smoother than ReLU).
- **Gating mechanism** increases model expressivity.
- Used in **PaLM, LLaMA-2, and other modern Transformers**.

Classification layer

The final hidden state of the Transformer decoder (H) has shape:

$$(B, T, D)$$

where:

- B = Batch size
- T = Sequence length
- D = Embedding dimension (5120 in LLaMA-2 13B)

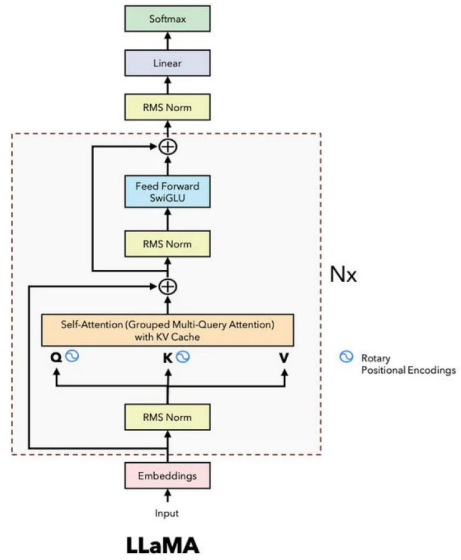
To predict the next token, LLaMA applies a **single linear transformation**:

$$\text{logits} = HW_{\text{out}} + b$$

where:

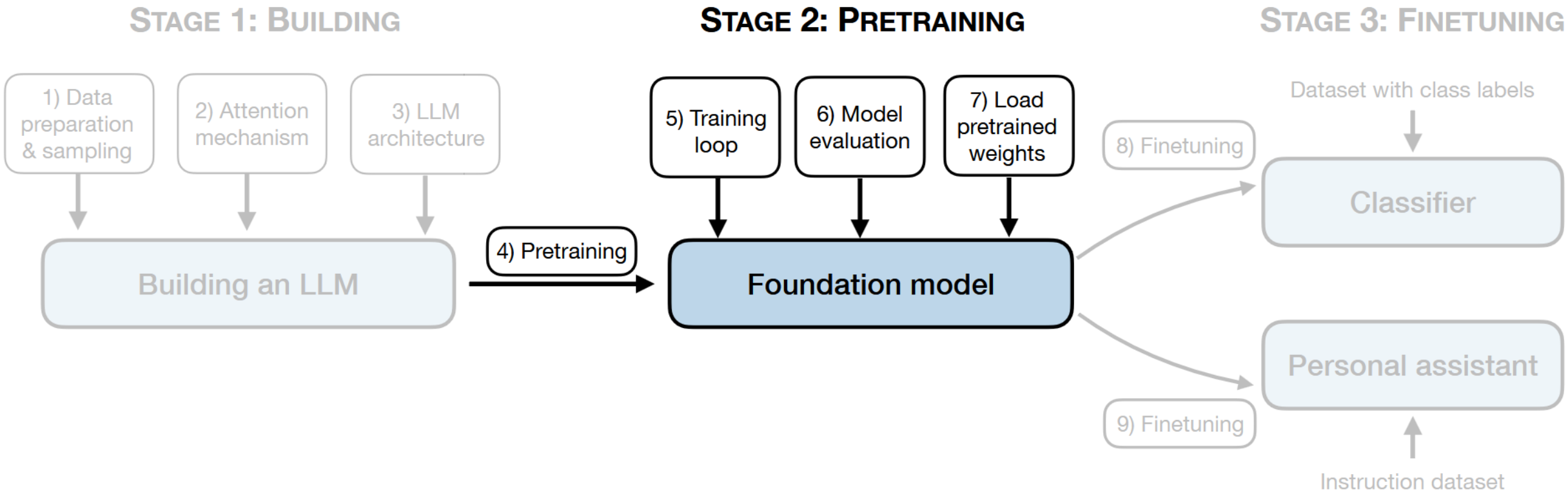
- W_{out} is the **output projection matrix** of shape $(5120, 32000)$.
- b is a **bias vector** of shape (32000) (if used).
- The output shape is $(B, T, 32000)$, giving the **logits over the vocabulary**.

Thus, the classification layer is **just one linear layer**, mapping from the **embedding space (5120)** to the **vocabulary size (32000)**.

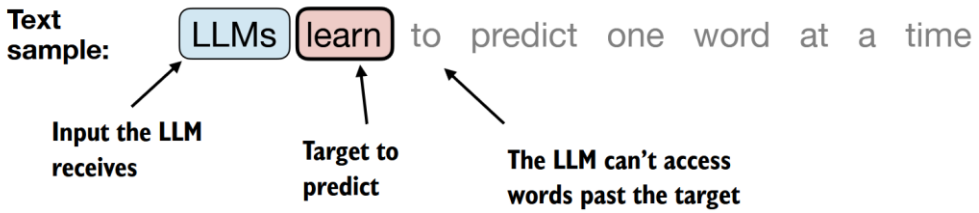


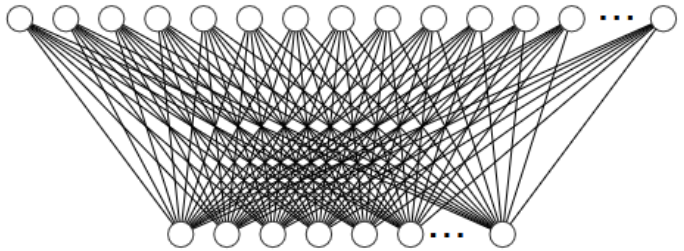
LLaMA

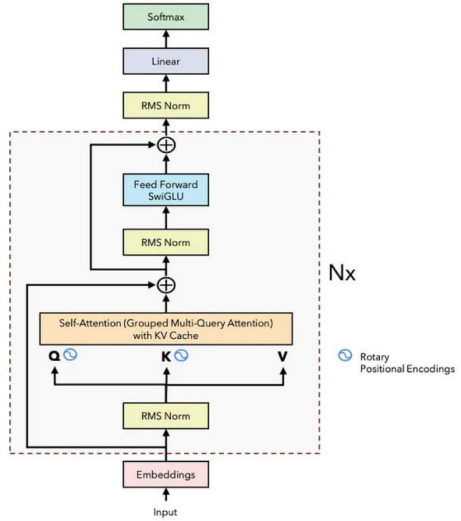
Building LLMs



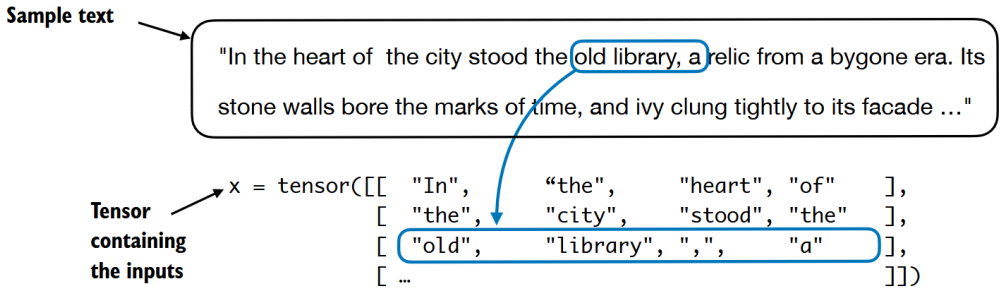
Pre-training







Sample 1	LLMs	learn	to	predict	one	word	at	a time
Sample 2	LLMs	learn	to	predict	one	word	at	a time
Sample 3	LLMs	learn	to	predict	one	word	at	a time
Sample 4	LLMs	learn	to	predict	one	word	at	a time
Sample 5	LLMs	learn	to	predict	one	word	at	a time
Sample 6	LLMs	learn	to	predict	one	word	at	a time
Sample 7	LLMs	learn	to	predict	one	word	at	a time
Sample 8	LLMs	learn	to	predict	one	word	at	a time



(Common input lengths are >1024)

Pre-training

Llama 3

"To train the best language model, the curation of a large, high-quality training dataset is paramount. In line with our design principles, we invested heavily in pretraining data. Llama 3 is pretrained on over 15T tokens that were all collected from publicly available sources."

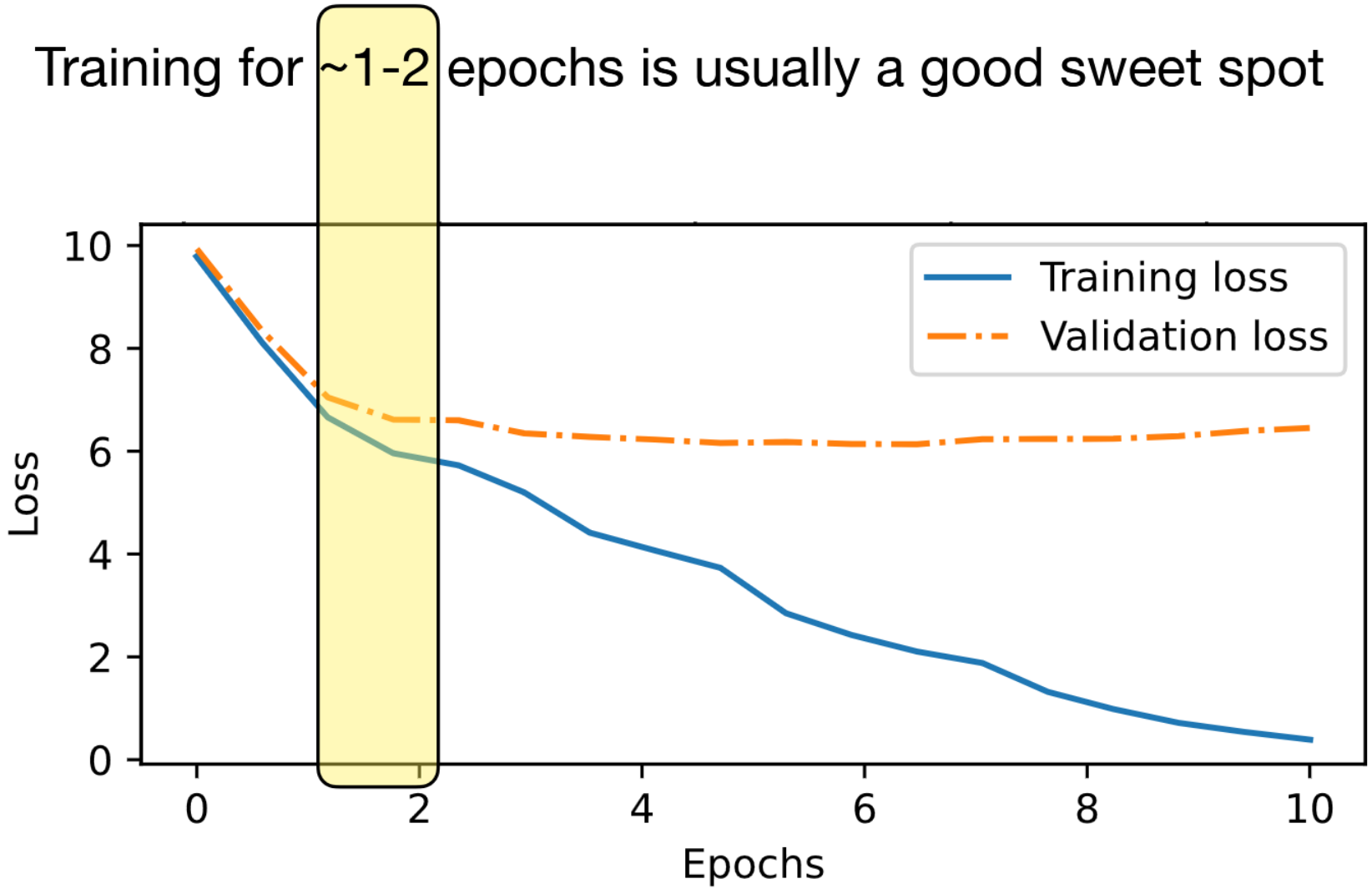
Introducing Meta Llama 3: The most capable openly available LLM to date (2024), https://ai.meta.com/blog/meta-llama-3/

Quantity vs Quality

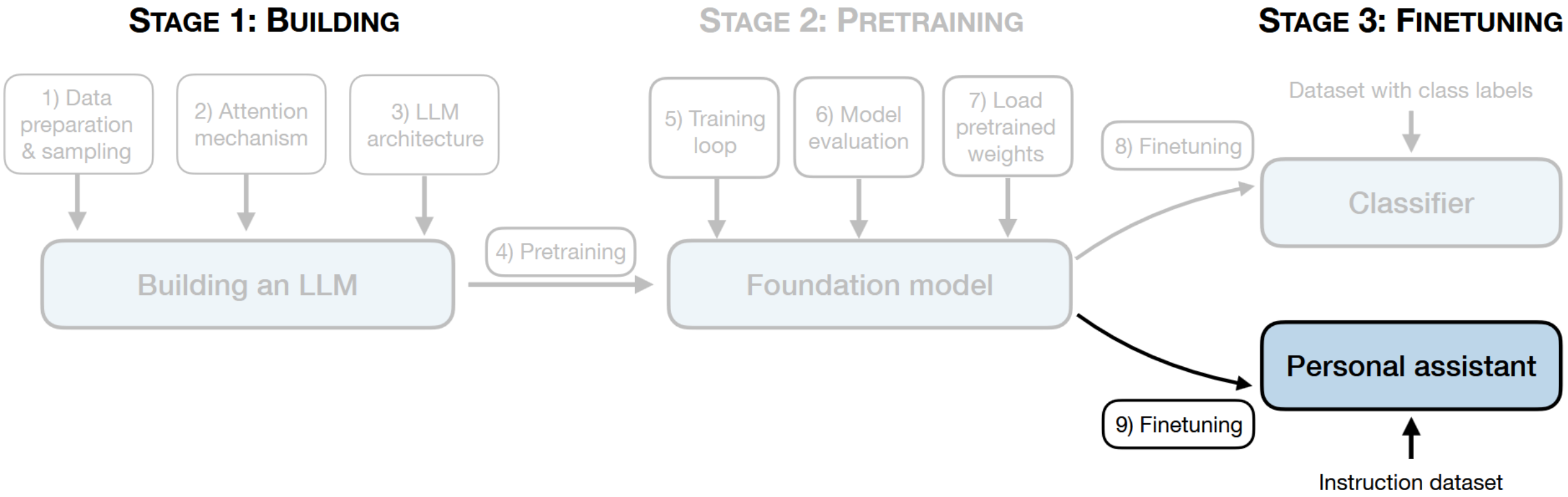
"we mainly focus on the quality of data for a given scale. We try to calibrate the training data to be closer to the "data optimal" regime for small models. In particular, we filter the publicly available web data to contain the correct level of "knowledge" and keep more web pages that could potentially improve the "reasoning ability" for the model. As an example, the result of a game in premier league in a particular day might be good training data for frontier models, but we need to remove such information to leave more model capacity for "reasoning" for the mini size models.

Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone (2024), https://arxiv.org/abs/2404.14219

Pre-training



Building LLMs



Instruction dataset


```
{
    "instruction": "Rewrite the following sentence using passive voice.",
    "input": "The team achieved great results.",
    "output": "Great results were achieved by the team."
},
```

Instruction dataset


```
{
    "instruction": "Rewrite the following sentence using passive voice.",
    "input": "The team achieved great results.",
    "output": "Great results were achieved by the team."
},
```

Apply prompt style template (for example, Alpaca-style)

```
Below is an instruction that describes a task. Write a response
that appropriately completes the request.

### Instruction:
Rewrite the following sentence using passive voice.

### Input:
The team achieved great results.

### Response:
Great results were achieved by the team.
```

Pass to LLM for supervised instruction finetuning

Problems of SFT

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1. Some mistakes are worse than others: In the end, we want good outputs

- Some mistaken predictions hurt more than others, so we'd like to penalize them appropriately
- Please send this package to Pittsburgh
- Please send a package to Pittsburgh
- Please send this package to Tokyo
- ****ing send this package to Pittsburgh

2. Bias in data: Corpora are full of outputs that we wouldn’t want a language model reproducing!

- For instance:
- Toxic comments in reddit
- Disinformation
- Translations from old machine translation systems

3. Exposure bias: The model is not exposed to mistakes during training, and cannot deal with them at test

Optimizing for human preferences

Let's say we were training a language model on some task (e.g. summarization).

For an instruction x and a LM sample y, imagine we had a way to obtain a human reward of that summary: $R(x,y) \in \mathbb{R}$, higher is better.

x

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

y_1

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

$R(x, y_1) = 8.0$

y_2

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

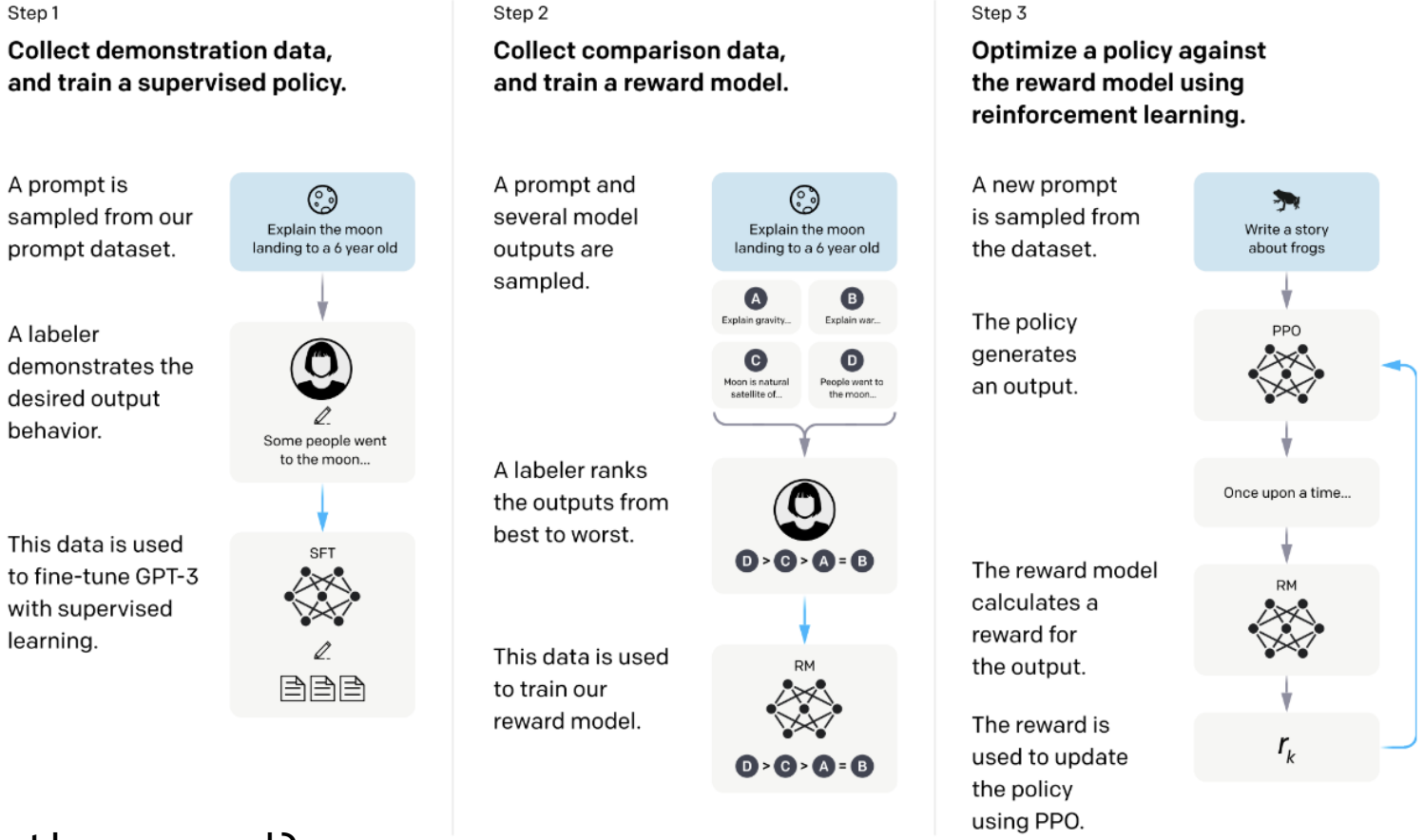
$R(x, y_2) = 1.2$

Now we want to maximize the expected reward of samples from our LM:

$$\mathbb{E}_{\hat{y} \sim p_\theta(y \mid x)}[R(x, \hat{y})]$$

Reinforcement learning with human feedback - RLHF

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



How to maximize the reward?

How do we get rewards?

Problem 1: human-in-the-loop is expensive!
Solution: instead of directly asking humans for preferences, model their preferences as a separate (NLP) problem! [Knox and Stone, 2009]

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

$R(x, y_1) = 8.0$

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

$R(x, y_2) = 1.2$

Train a $RM_\phi(x, y)$ to predict human reward from an annotated dataset, then optimize for RM_ϕ instead.

How do we get rewards?

Problem 2: human judgments are noisy and miscalibrated!
Solution: instead of asking for direct ratings, ask for pairwise comparisons, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.

$$R(x, y_3) = \begin{matrix} y_3 \\ 4.1? \end{matrix} \quad \begin{matrix} \\ 6.6? \end{matrix} \quad \begin{matrix} \\ 3.2? \end{matrix}$$

How do we get rewards?

Problem 2: human judgments are noisy and miscalibrated!
Solution: instead of asking for direct ratings, ask for pairwise comparisons, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

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

y_1

$>$

A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.

1.2 y_3

$>$

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

y_2

How do we get rewards? Bradley-Terry

$$J_{RM}(\phi) = -\mathbb{E}_{(x, y^w, y^l)\sim D}\left[\log \sigma(RM_\phi(x, y^w) - RM_\phi(x, y^l))\right]$$

"winning" sample

"losing" sample

y^w should score higher than y^l

$$P(y_w > y_l) = \frac{e^{r^\phi(x,y_w)}}{e^{r^\phi(x,y_w)} + e^{r^\phi(x,y_l)}}$$

$$\frac{e^A}{e^A + e^B} \implies \sigma(A - B)$$

Reinforcement Learning

Agent: the cat

State: the position of the cat (x, y) in the grid

Action: at each position, the cat can move to one of the 4-directionally connected cells. If a move is invalid, the cell will not move and remain in the same position. Every time the cat makes a move, it results in a new state and a reward.

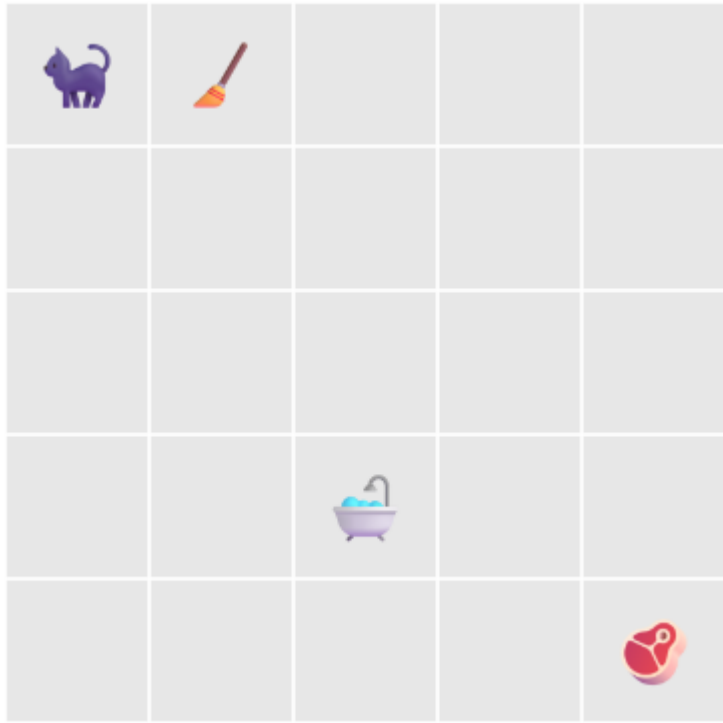
Reward model:

- A move to another empty cell results in a reward of 0.
- A move towards the broom, will result in a reward of -1.
- A move towards the bathtub will result in a reward of -10 and the cat fainting (episode over). The cat will be respawned at the initial position again.
- A move towards the meat will result in a reward of +100.

Policy: a policy rules how the agent selects the action to perform given the state it is in:

$$a_t \sim \pi(\cdot \mid s_t)$$

The goal in RL is to select a policy that maximizes the expected return when the agent acts according to it.



Reinforcement Learning

Agent: the language model itself

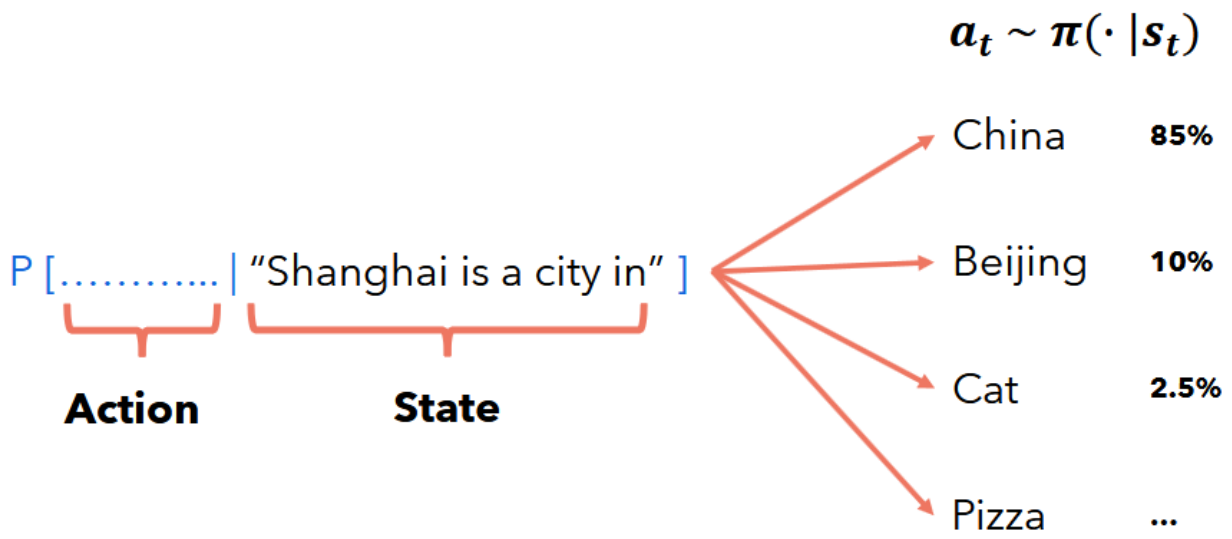
State: the prompt (input tokens)

Action: which token is selected as the next token

Reward model: the language model should be rewarded for generating "good responses" and should not receive any reward for generating "bad responses".

Policy: In the case of language models, the policy is the language model itself! Because it models the probability of the action space given the current state of the agent:

$$a_t \sim \pi(\cdot \mid s_t)$$



Reinforcement Learning

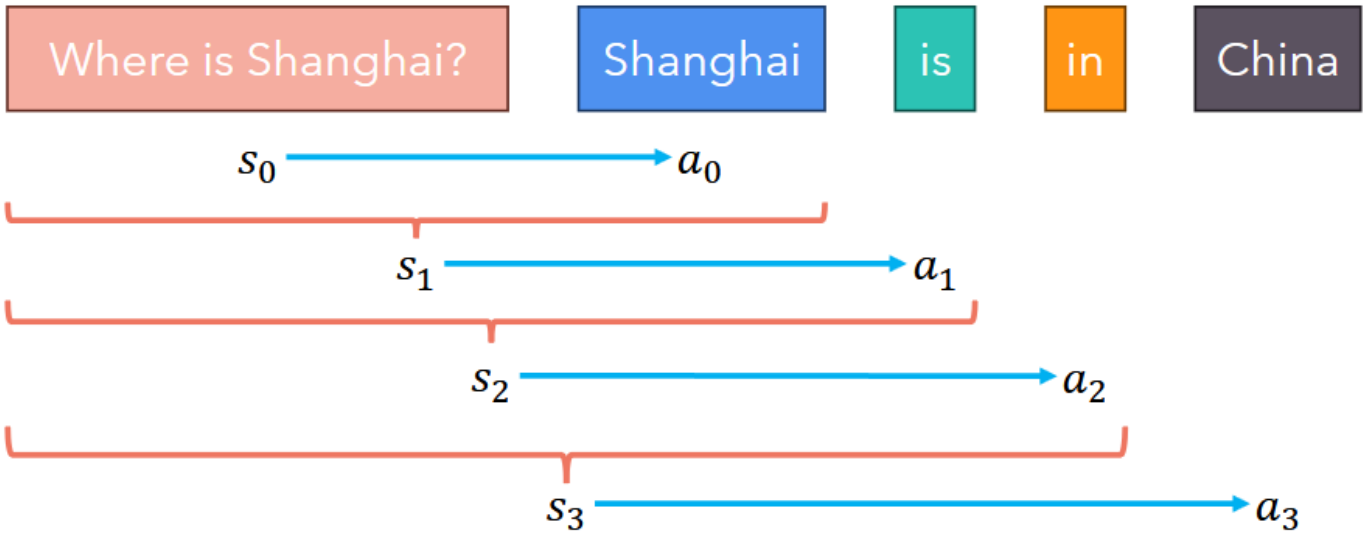
State and Actions in Language Modeling

- The **state** (s_t) is the partial sentence generated so far. - The **action** (a_t) is the next word the model chooses.

Example:

- s_t = "The best way to"
- a_t = "learn"
- New state after action: s_{t+1} = "The best way to learn"

$$\tau = (s_0, a_0, s_1, a_1, \ldots).$$



RLHF

We have the following:

- A pretrained (possibly instruction-finetuned) LM (policy) $\pi_\theta(\cdot|s_t)$
- A reward model $RM_\phi(\tau)$ that produces scalar rewards for LM outputs, trained on a dataset of human comparisons, where $\tau = (s_0, a_0, a_1, a_2, \ldots, a_T)$

Now to do RLHF:

- Copy the policy $\pi_\theta^{RL}(a_t \mid s_t)$, with parameters θ we would like to optimize
- We want to optimize:

$$\mathbb{E}_{\tau \sim \pi_\theta^{RL}(a_t|s_t)}[RM_\phi(\tau)]$$

RLHF

Why is computing the expectation problematic?

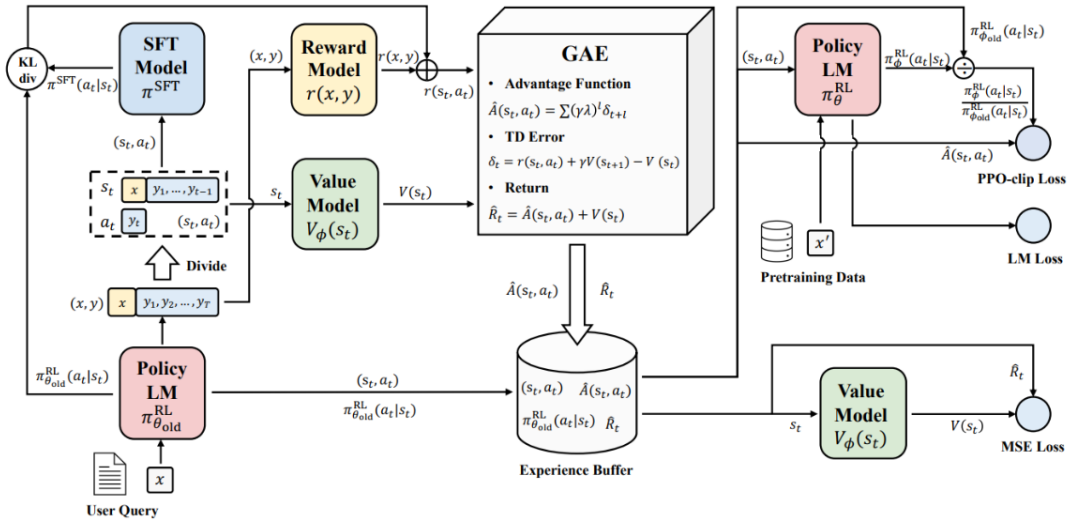
$$\mathbb{E}_{\tau \sim \pi_\theta^{RL}(a_t|s_t)}[RM_\phi(\tau)]$$

Issues:

- Sum over all possible outputs is intractable
- Language models operate over an exponentially large space
- Exact computation requires iterating over all possible sentences

Solution: Use policy gradient methods like PPO to estimate and optimize this expectation efficiently.

Proximal Policy Optimization - PPO

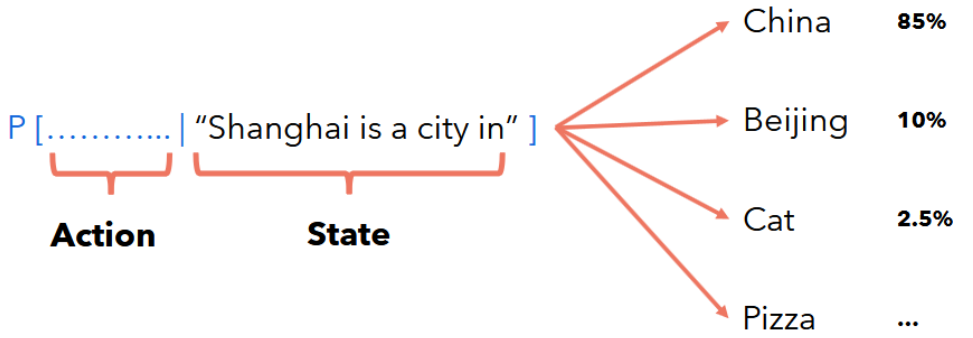


[Secrets of RLHF. Zheng et al. 2023]

Key Components:

- **Policy** (π_θ): The LLM that has been pre-trained / SFT'ed.
- **Reward model** (R_ϕ): A trained and frozen network that provides a scalar reward given a complete response to a prompt.
- **Critic** (V_γ): Also known as the value function, a learnable network that takes in a partial response to a prompt and predicts the scalar reward.

$$a_t \sim \pi(\cdot\,|s_t)$$



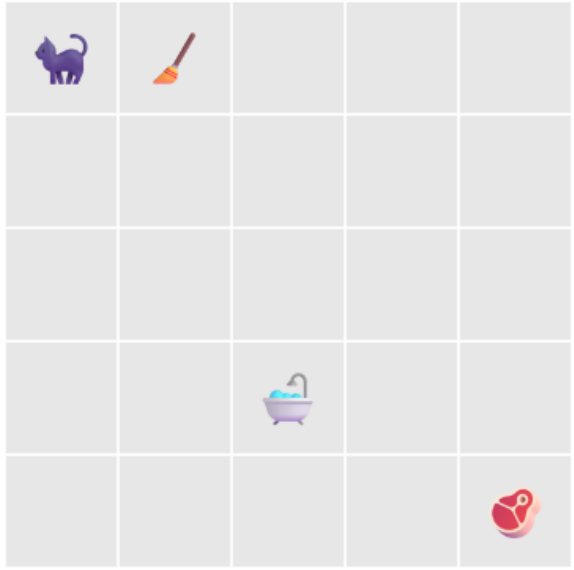
RLHF – Advantage Function

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

- Measures how much better an action a_t is compared to the average action.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

- Where:
  - $Q(s_t, a_t)$ = Expected total reward for taking a_t.
  - $V(s_t)$ = Expected reward over all possible actions.



Expected Value $V(s_t)$

- Represents the **average expected reward** from state s_t.
- Computed as:

$$V(s_t) = \sum_{a_t} \pi(a_t \mid s_t) Q(s_t, a_t)$$

where:

- $\pi(a_t \mid s_t)$ = Probability of taking action a_t.
- $Q(s_t, a_t)$ = Expected return for action a_t.

Example:

- State: s_t = "The best way to"
- Actions:
  - a_1 = "learn" → $Q(s_t, a_1) = 0.9$
  - a_2 = "sleep" → $Q(s_t, a_2) = 0.2$
- If $V(s_t) = 0.6$:
  - $A(s_t, \text{"learn"}) = 0.9 - 0.6 = 0.3$ (good)
  - $A(s_t, \text{"sleep"}) = 0.2 - 0.6 = -0.4$ (bad)

RLHF – General Advantage Estimation

UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN

Escuela de
Ciencias Aplicadas
e Ingeniería

- **Monte Carlo Sampling:**
  - Uses the reward of the full trajectory (entire response).
  - Evaluation Method: Computes reward only after the full response is generated.
  - Granularity: Sentence/response level rather than token level.
  - Advantage: Low bias since it accurately models the final reward.
  - Drawback: High variance due to sparse rewards. Cost: Expensive to sample enough responses from the LLM for effective optimization.

- Temporal Difference (TD):
  - Uses one-step trajectory reward.
  - Evaluation Method: Measures how good the newly generated word is given the prompt.
  - Granularity: Computes reward at the token level.
  - Advantage: Reduces variance in reward estimation.
  - Drawback: Increases bias since it cannot accurately anticipate the final reward from a partial response.

Bias and Variance in PPO

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Low Bias, High Variance

- Computes the advantage using the full sum of all future rewards. - Since it fully considers future rewards, it is an **unbiased** estimate. - However, rewards can be noisy, so advantage values fluctuate a lot (**high variance**).

Example (High Variance): Imagine training a chatbot to answer a question:

- The model generates an entire sentence before getting feedback.
- The final score depends on the full response.
- Small changes in earlier words may drastically change the total reward.

Since rewards depend on far-future tokens, the advantage values vary a lot.

High Bias, Low Variance

- Estimates the advantage using only **one-step lookahead** (TD(0)). - This makes the estimate very **stable** (**low variance**). - However, it ignores future rewards, leading to a **biased** estimate.

Example (High Bias):

- Suppose the chatbot predicts: **"Reinforcement learning is used for"**
- It only looks at the **next word** to estimate the value of the sentence.
- This is incorrect because the full sentence matters!

Since it ignores long-term effects, the advantage estimate is **biased**.

The critic value function

What is the Critic?

- The critic estimates the expected future reward from a given state.
- It is trained alongside the language model to predict the reward model's output.

Objective:

- The critic learns to approximate the true value function:

$$V_\gamma(s_t) \approx RM_\phi(\tau)$$

- This helps stabilize training by reducing variance in advantage estimation.

Loss Function:

$$L_r = \mathbb{E}_t \left[\left(V_\gamma(s_t) - RM_\phi(\tau) \right)^2 \right]$$

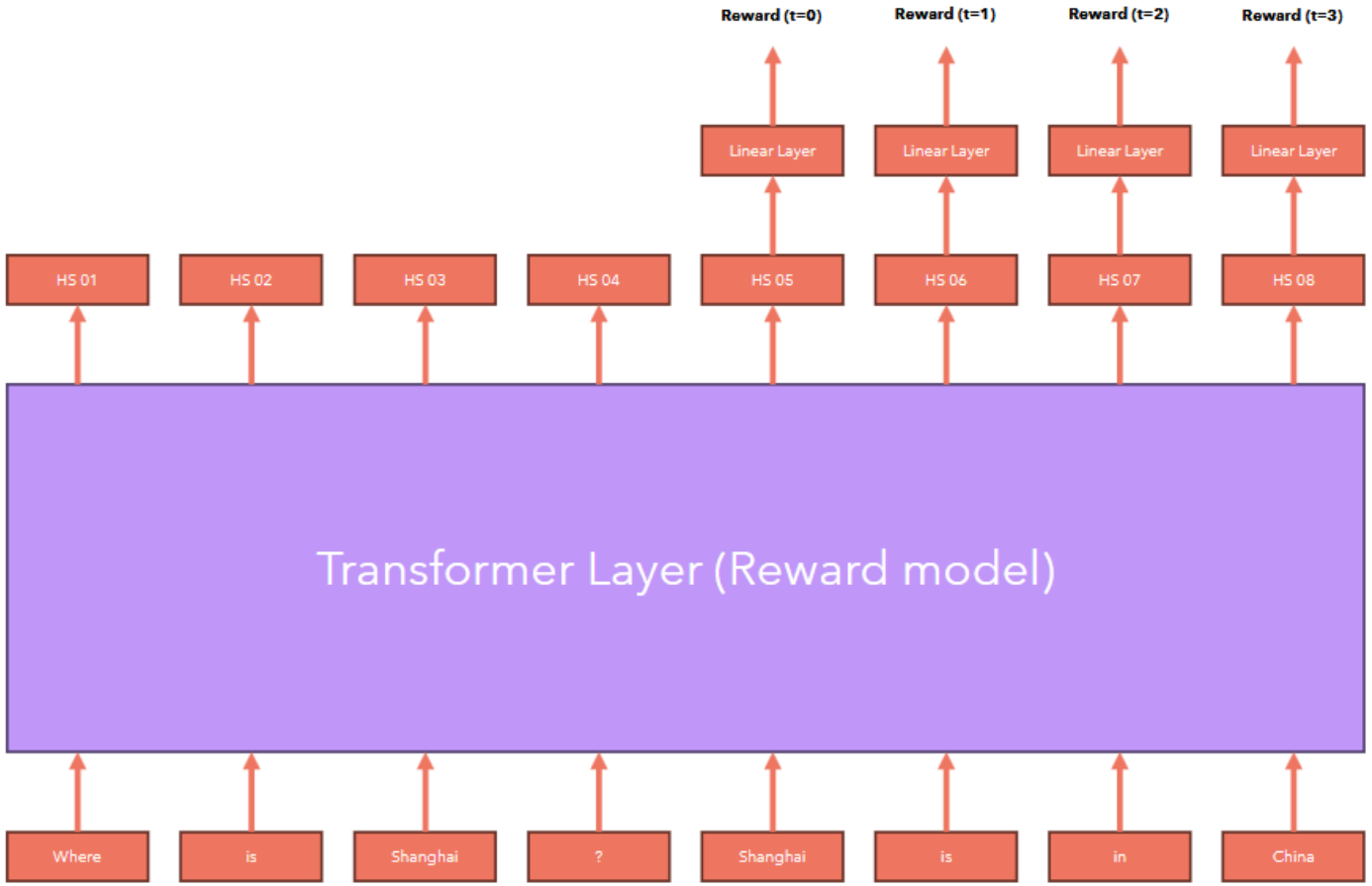
where:

- $V_\gamma(s_t)$ is the critic's estimated value of state s_t.
- $RM_\phi(\tau)$ is the reward model output.

Why is this Useful?

- The critic provides a baseline for computing advantage estimates.
- It reduces high variance in policy updates, making training more stable.

The critic value function



Linear layer with only one output feature

GAE

From Value Function to Advantage Function

- The critic estimates $V(s_t)$, the expected reward from a state.
- To improve policy updates, we need to compute the advantage function:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

- Since $Q(s_t, a_t)$ is unknown, we approximate it using the Temporal Difference (TD) Residual:

$$\delta_t = r_t + \gamma V(s_{t+\mathbf{1}}) - V(s_t)$$

Generalized Advantage Estimation (GAE)

- GAE smooths advantage estimation by accumulating discounted TD residuals:

$$A_t^{GAE(\lambda)} = \sum_{l=\mathbf{0}}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

- The parameter λ balances bias vs. variance:
  - $\lambda = \mathbf{1}$ → More future information, higher variance.
  - $\lambda = \mathbf{0}$ → Immediate rewards, lower variance but higher bias.

Why GAE?

- Reduces high variance in advantage estimates.
- Provides a stable signal for policy updates.
- Improves training efficiency in reinforcement learning.

PPO Cost Function

- The clipped surrogate objective prevents overcommitment to a single action by limiting updates.
- The objective is defined as:

$$\mathcal{L}^{\text{clip}}(\theta) = \mathbb{E}_t \left[\min \left(c_t(\pi_\theta) A_t^{GAE}, \text{clip} \left(c_t(\pi_\theta), \mathbf{1} - \epsilon, \mathbf{1} + \epsilon \right) A_t^{GAE} \right) \right]$$

- Here, the probability ratio is defined as:

$$c_t(\pi_\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

where ϵ controls the clipping range.

- Suppose an LLM assigns the word "unlimited" the following probabilities:
  - **Before update:** 0.1
  - **After update:** 0.3
- The probability ratio is computed as:

$$c_t = \frac{0.3}{0.1} = 3$$

- If we take $\epsilon = 0.2$, the clipped value is 1.2.
- The final clipped surrogate loss is:

$$\mathcal{L}^{\text{clip}}(\pi_\theta) = 1.2 A_t^{GAE}$$

- Prevents overconfidence in policy updates.
- Ensures stable learning by bounding updates.

PPO Cost Function

KL Divergence Penalty:

- Prevents the current policy θ from deviating too far from the original model θ_{orig}.
- The KL term is computed as:

$$KL(\theta) = \mathbb{E}_{s_t}\left[D_{KL}\left(\pi_{\theta_{\text{orig}}}(\cdot|s_t)||\pi_\theta(\cdot|s_t)\right)\right]$$

- Estimated by taking the average over sequence and batch.

Entropy Bonus:

- Encourages exploration by penalizing low entropy.
- Defined as:

$$H(\theta) = -\mathbb{E}_{a_t}\left[\log \pi_\theta(a_t|s_t)\right]$$

- Helps prevent premature convergence by maintaining diverse policy outputs.

PPO Cost Function

Final PPO Objective:

- The PPO objective combines multiple terms to balance reward maximization, entropy, and stability.
- The full loss function is:

$$\mathcal{L}_{PPO}(\theta, \gamma) = \mathcal{L}_{\text{clip}}(\theta) + w_1 H(\theta) - w_2 KL(\theta) - w_3 \mathcal{L}(\gamma)$$

where:

- $\mathcal{L}_{\text{clip}}(\theta)$: Maximizes rewards while preventing overcommitment.
- $H(\theta)$: Encourages exploration by maximizing entropy.
- $KL(\theta)$: Penalizes deviations from the reference policy to maintain stability.
- $\mathcal{L}(\gamma)$: Minimizes error in value predictions (critic L2 loss).

Summary of PPO Terms:

Term	Purpose
$\mathcal{L}_{\text{clip}}(\theta)$	Maximize rewards for high-advantage actions (clipped to avoid instability).
$H(\theta)$	Maximize entropy to encourage exploration.
$KL(\theta)$	Penalize deviations from the reference policy (stability).
$\mathcal{L}(\gamma)$	Minimize error in value predictions (critic L2 loss).

Group Relative Policy Optimization (GRPO)

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Why GRPO?

- Eliminates the need for a separate value function estimator (critic).
- Reduces memory and compute overhead by approximately 50%.

Key Idea:

- Instead of relying on a learned value function, GRPO estimates advantages directly using a group of sampled actions.
- For a given state s, sample multiple actions $a_1, \ldots, a_G$.
- Compute the group-relative advantage:

$$A^{\text{GRPO}}(s, a_j) = \frac{R(s, a_j) - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of rewards in the sampled group.

Group Relative Policy Optimization (GRPO)

Three Key Steps in GRPO:

1. Sample multiple actions $a_1, \ldots, a_G$ for each state s_t.
2. Normalize rewards using the sampled group:

$$\mu = \frac{1}{G} \sum_{j=1}^{G} R(s_t, a_j), \quad \sigma^2 = \frac{1}{G} \sum_{j=1}^{G} (R(s_t, a_j) - \mu)^2$$

3. Compute and optimize policy loss using group-relative advantage:

$$A^{GRPO}(s_t, a_j) = \frac{R(s_t, a_j) - \mu}{\sigma}$$

Why These Steps Matter:

- Ensures policy updates are stable.
- Uses a relative comparison instead of absolute values, reducing variance.
- Efficiently updates the policy without needing a separate critic.

GRPO Cost Function

Objective in GRPO

- Similar to PPO, GRPO uses a **clipped surrogate loss** and a **KL penalty**.
- The clipping mechanism stabilizes updates, while KL divergence prevents the policy from diverging too far from the original.

Clipped Surrogate Loss:

$$\mathcal{L}_{\text{clip}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \min \left(\frac{\pi_\theta(r_i|p)}{\pi_{\theta_{old}}(r_i|p)} A_i, \text{clip} \left(\frac{\pi_\theta(r_i|p)}{\pi_{\theta_{old}}(r_i|p)}, 1-\epsilon, 1+\epsilon \right) A_i \right)$$

Final GRPO Objective:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathcal{L}_{\text{clip}}(\theta) - w_1 D_{KL}(\pi_\theta || \pi_{\text{orig}})$$

Why This Works:

- The clipped loss maximizes reward while avoiding large updates.
- The KL penalty ensures policy consistency with the original model.
- GRPO avoids the need for a separate value function, reducing computational cost.

Direct Preference Optimization (PPO)

Can we simplify RLHF? Towards Direct Preference Optimization **Current pipeline is as follows:**

- Train a reward model $RM_\phi(x, y)$ to produce scalar rewards for LM outputs, trained on a **dataset of human comparisons**.
- Optimize pretrained (possibly instruction-finetuned) LM $p^{PT}(y|x)$ to produce the final RLHF LM $p_\theta^{RL}(\hat{y}|x)$.

What if there was a way to write $RM_\phi(x, y)$ in terms of $p_\theta^{RL}(\hat{y}|x)$?

- Derive $RM_\theta(x, y)$ in terms of $p_\theta^{RL}(\hat{y}|x)$.
- Optimizing parameters θ by fitting $RM_\theta(x, y)$ to the preference data instead of $RM_\phi(x, y)$.

How is this possible? The only external information to the optimization comes from the preference labels.

Direct Preference Optimization (PPO)

Objective to Maximize:

$$\mathbb{E}_{\hat{y}\sim p_\theta^{RL}(\hat{y}|x)}\left[RM(x,\hat{y}) - \beta \log\left(\frac{p_\theta^{RL}(\hat{y}|x)}{p^{PT}(\hat{y}|x)}\right)\right]$$

Closed-Form Solution:

$$p^*(\hat{y}|x) = \frac{1}{Z(x)} p^{PT}(\hat{y}|x) \exp\left(\frac{1}{\beta} RM(x,\hat{y})\right)$$

Rearrange the Terms:

$$RM(x,\hat{y}) = \beta \log \frac{p^*(\hat{y}|x)}{p^{PT}(\hat{y}|x)} + \beta \log Z(x)$$

For Arbitrary LMs:

$$RM_\theta(x,\hat{y}) = \beta \log \frac{p_\theta^{RL}(\hat{y}|x)}{p^{PT}(\hat{y}|x)} + \beta \log Z(x)$$

Direct Preference Optimization (PPO)

Start with the Optimal Policy Equation:

$$p^*(y|x) = \frac{1}{Z(x)} p^{PT}(y|x) \exp\left(\frac{1}{\beta} RM(x,y)\right)$$

Step 1: Take the Natural Logarithm

$$\log p^*(y|x) = \log p^{PT}(y|x) + \frac{1}{\beta} RM(x,y) - \log Z(x)$$

Step 2: Solve for $RM(x,y)$

$$RM(x,y) = \beta \log \frac{p^*(y|x)}{p^{PT}(y|x)} + \beta \log Z(x)$$

Interpretation:

- The reward function measures how much more likely the optimized policy is compared to the pretrained policy.
- The term β scales how strongly the reward influences policy updates.
- The partition function $Z(x)$ acts as a normalization constant.
- In DPO, we focus on **reward differences**, removing $Z(x)$.

Direct Preference Optimization (PPO)

Fitting the Reward Model:

$$J_{RM}(\phi) = -\mathbb{E}_{(x,y^w,y^l)\sim D}\left[\log \sigma(RM_\phi(x, y^w) - RM_\phi(x, y^l))\right]$$

Difference Between Rewards:

$$RM_\theta(x, y^w) - RM_\theta(x, y^l) = \beta \log \frac{p_\theta^{RL}(y^w|x)}{p^{PT}(y^w|x)} - \beta \log \frac{p_\theta^{RL}(y^l|x)}{p^{PT}(y^l|x)}$$

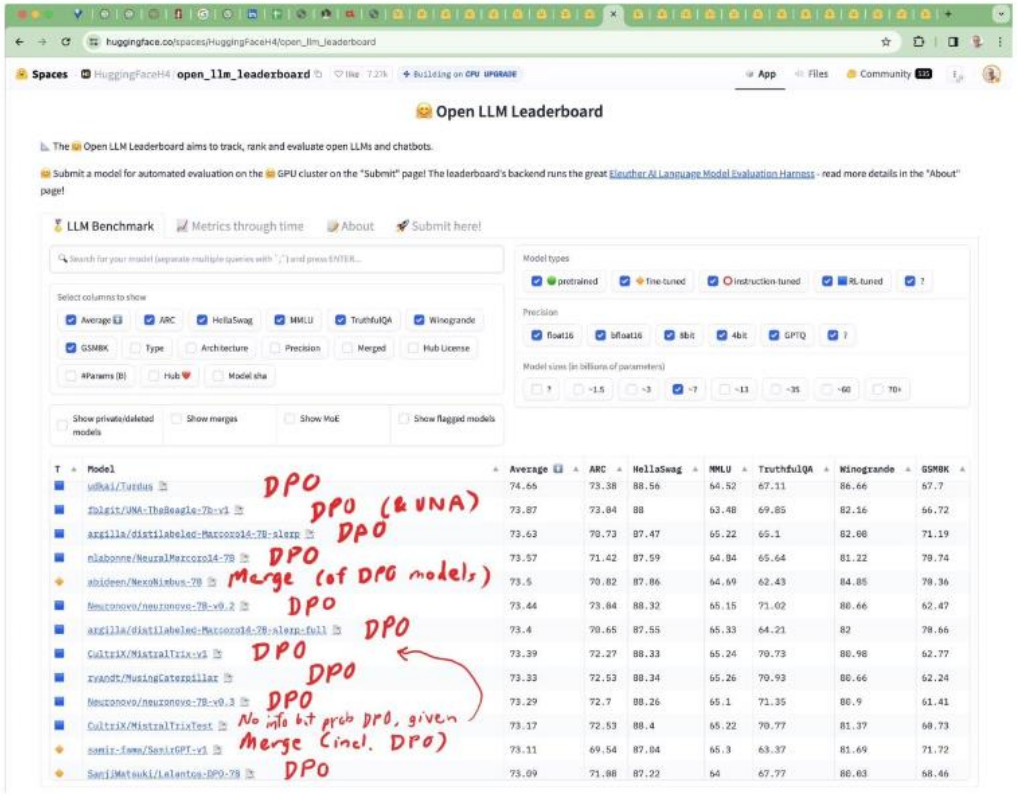
Final DPO Loss Function:

$$J_{DPO}(\theta) = -\mathbb{E}_{(x,y^w,y^l)\sim D}\left[\log \sigma(RM_\theta(x, y^w) - RM_\theta(x, y^l))\right]$$

Key Insight:

- A simple classification loss connects preference data to language model parameters directly!

Direct Preference Optimization (PPO)



Open source LLMs now almost all just use DPO (and it works well!)

	GPT - 3.5	Mistral Small	Mistral Medium
MT Bench (for Instruct models)	8.32	8.30	**8.61**

https://mistral.ai/news/mixtral-of-experts/

Instruction fine-tuning

pretrained models in chat use cases, we innovated on our well. Our approach to post-training is a combination of tion sampling, proximal policy optimization (PPO), and PO). The quality of the prompts that are used in SFT and sed in PPO and DPO has an outsized influence on the me of our biggest improvements in model quality came nd performing multiple rounds of quality assurance on notators.

Learning from preference rankings via PPO and DPO also greatly improved the performance of Llama 3 on reasoning and coding tasks. We found that if you ask a model a reasoning question that it struggles to answer, the model will sometimes produce the right reasoning trace: The model knows how to produce the right answer, but it does not know how to select it. Training on preference rankings enables the model to learn how to select it.

RLHF

Instruction: What are the five most common causes of stress among people?

Alpaca (Finetuned w/ 10k data): The five most common causes of stress among people are work, money, relationships, health, and family

Alpaca + PPO: The five most common causes of stress among people are:

1. Pressure at work: Whether it is due to tight deadlines, high expectations, or an overwhelming workload, pressure at work can be a major source of stress
2. Money problems: Financial issues can be a significant source of stress, from not being able to make ends meet to struggling with debt.
3. Unhealthy lifestyle: Poor diet, lack of exercise, and sleep deprivation can all contribute to stress.
4. Unhappiness at home: Problems in relationships, conflicts with family members, and other domestic issues can be a major source of stress.
5. Uncertainty: Excessive worry and anxiety about the future, health issues, or other unknowns can be a major cause of stress

Reasoning models

https://sebastianraschka.com/blog/2025/understanding-reasoning-llms.html

Mixture of Experts (MoEs)

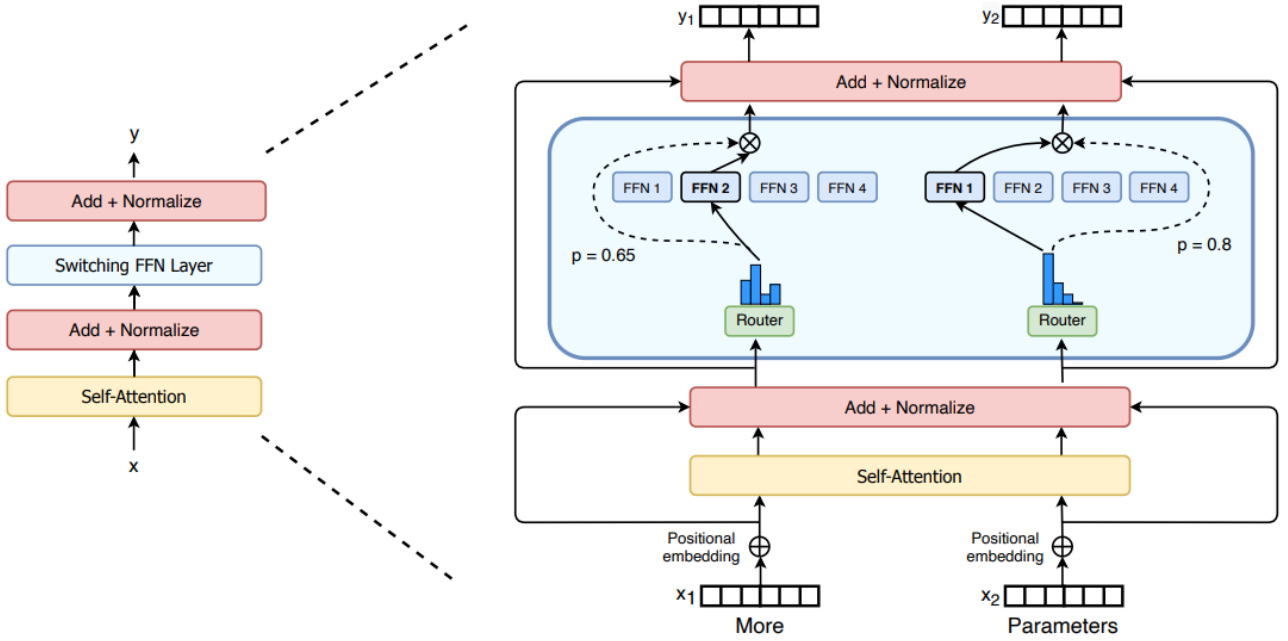
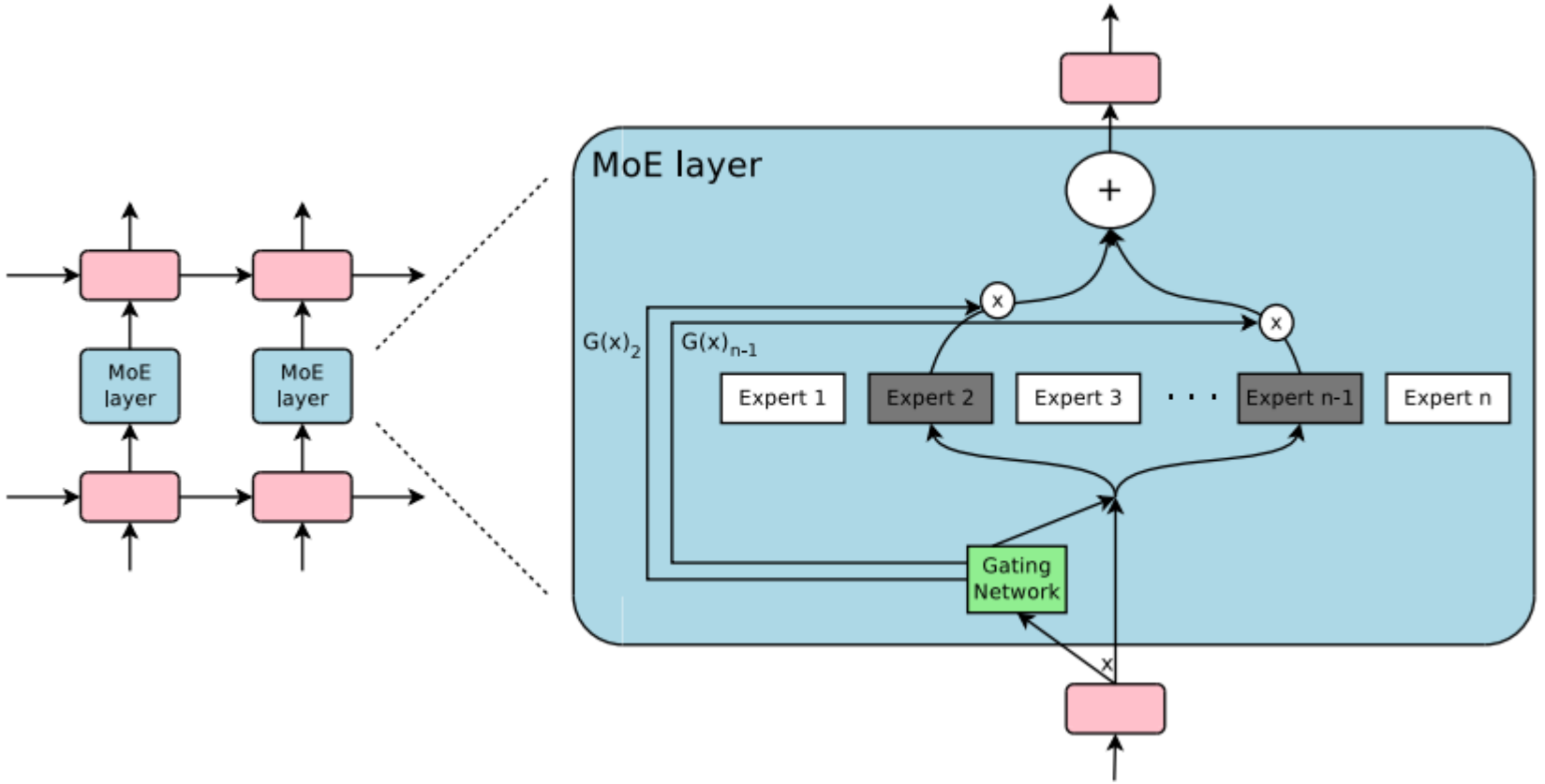


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

Mixture of Experts (MoEs)



Mixture of Experts (MoEs)

Mixture of Experts Setup:

- Each expert is weighted by a gating function $G(x)$.
- The output is computed as:

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

- If $G(x) = 0$, that expert is not used, saving computation.

Gating Function: Softmax Selection

$$G_\sigma(x) = \text{Softmax}(x \cdot W_g)$$

Noisy Top-k Gating (Shazeer's Approach)

1. Add tunable noise:

$$H(x)_i = (x \cdot W_g)_i + \text{StandardNormal}() \cdot \text{Softplus}((x \cdot W_{\text{noise}})_i)$$

2. Keep only the top-k values:

$$\text{KeepTopK}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in top k elements of } v, \\ -\infty & \text{otherwise.} \end{cases}$$

3. Apply the softmax:

$$G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k))$$