🔥 LLM FINE TUNING (REVISION) 📚

Sample question- What is the Capital of France?

(This question is used most of the times as refrence)

1. What is the diff bw bf16 and fp16?

- **Sign Bit**: Determines if the number is positive or negative.
- **Exponent Bits**: Controls the range (how big or small the number can be).
- **Mantissa Bits**: Controls precision (how much detail the number has).

Feature	**BF16 (Brain Float 16)**	**FP16 (Half Precision)**
Bits Total	16	16
Sign Bits	1	1
Exponent Bits	8	5
Mantissa Bits	7	10
Range	~5.96e-38 to ~3.39e38	~5.96e-8 to 65,504
Precision	Lower (less detail in small numbers)	Higher (more detail in small numbers)
Exponent Range	-126 to +127	-14 to +15
Memory Usage	2 bytes	2 bytes
Best For	Large-scale training, bigger models	Smaller models, older hardware
Hardware Support	Newer GPUs (e.g., A100, H100)	Broad support (e.g., V100, T4)
Risk of Overflow	Lower (wider range)	Higher (narrower range)

2. Having less trainging data what can that cause?

1. Model might be overfitting on training data if given more epochs
2. And model might learn chat template and in answer we can get to see special tokens like [INST],<s> etc

3. Diff bw top_k and top_p during Inference of LLM?

1. Top-K vs. Top-P Sampling (Inference Notes)

- Purpose: Control randomness in text generation, balance creativity & coherence.
- My Context: Fine-tuning a pirate model (250 examples) to respond in dialect.

Top-K Sampling:

- Def: Picks top k most likely tokens (e.g., k=50), samples from them.
- How: Sort probs, take top k, renormalize, sample.
- Pros: Fixed diversity, good for consistent tone (e.g., "grog," "arr").
- Cons: Misses tokens if k too small, dilutes if too big.
- Ex: k=30 → "Swabbed the deck," repetitive with small dataset.

Top-P (Nucleus) Sampling:

- Def: Picks tokens until cumulative prob > p (e.g., p=0.9), samples from nucleus.
- How: Sort probs, keep tokens until p reached, sample.
- Pros: Adaptive diversity, creative (e.g., "Wrestled a sea beast").
- Cons: Less predictable, can get too wild if p too high.
- Ex: p=0.9 → "Bartered with a mermaid," varied output.

Differences:

- Top-K: Fixed k tokens, static control.
- Top-P: Dynamic nucleus, adapts to prob distribution.
- Control: k for number, p for prob mass.

My Pirate Model:

- Issue: Extra [INST] pairs, repeats training data.
- Current: top_k=30, top_p=0.9, temp=0.1 → Too rigid, overfit.
- Fix: top_k=50 (alone) or top_p=0.95 (alone), temp=0.7 → Less repetition, more flair.

Talking Points:

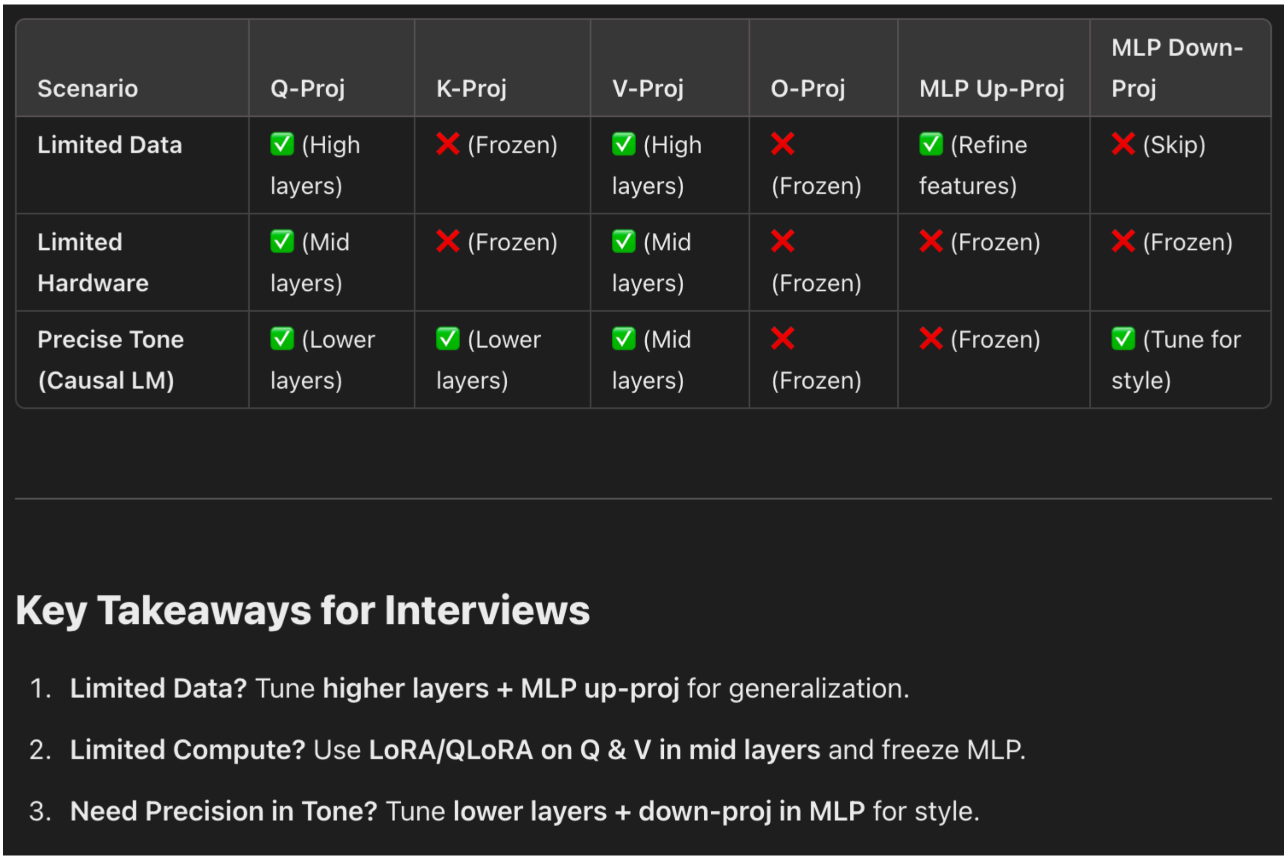
- "Top-K is picking my top 50 pirate words; Top-P grabs 90% of my lingo’s juice."
- "Top-K for tight pirate vibe, Top-P for wild tales."
- "I’d use top_p=0.95 to avoid parroting my small dataset."

4. How do you decide between top_k and top_p?

   1. If I need a tight stricter tone—like ‘Arr, swab the deck!’—I’d pick top_k=30 for consistency. But for a wild tale like fighting a sea beast, I’d go top_p=0.95 to let the model stretch its creativity, especially with my small 250-example dataset.

5. **What is common heuristic when deciding r and alpha(lora_alpha) in LORA?**
   1. alpha = 2*r

6. **When to use which layers from (q_proj, k_proj, v_proj) and mlp layers (up_proj,down_proj) and when to choose which for fine-tuning using LORA ?**



Scenario	Q-Proj	K-Proj	V-Proj	O-Proj	MLP Up-Proj	MLP Down-Proj
Limited Data	✅ (High layers)	❌ (Frozen)	✅ (High layers)	❌ (Frozen)	✅ (Refine features)	❌ (Skip)
Limited Hardware	✅ (Mid layers)	❌ (Frozen)	✅ (Mid layers)	❌ (Frozen)	❌ (Frozen)	❌ (Frozen)
Precise Tone (Causal LM)	✅ (Lower layers)	✅ (Lower layers)	✅ (Mid layers)	❌ (Frozen)	❌ (Frozen)	✅ (Tune for style)

Key Takeaways for Interviews

1. **Limited Data?** Tune **higher layers + MLP up-proj** for generalization.
2. **Limited Compute?** Use **LoRA/QLoRA on Q & V in mid layers** and freeze MLP.
3. **Need Precision in Tone?** Tune **lower layers + down-proj in MLP** for style.

7. **What is q_proj, k_proj, v_proj and o_proj in most of decoder based LLM?**
   1. **Example Sentence: "My name is Sahib I am studying"**
      - **Goal**: Turn into pirate style, e.g., "Arr, I be Cap’n Sahib, learnin’ the seas!"

   Roles in Attention

   - **q_proj (Query)**: Asks, "What’s this about?" for each word.
     - Ex: For "Sahib", asks, "Who’s this?" → Focuses on "name".
   - **k_proj (Key)**: Labels each word for matching.
     - Ex: "name" labeled as "identity", "is" as "connector".

- **v_proj (Value)**: Stores info to retrieve.
    - Ex: "Sahib" stores "I'm the name", "studying" stores "I'm the action".
- **o_proj (Output)**: Combines focused info into output.
    - Ex: "Sahib" + "name" → "Sahib is the name, ready for pirate flair."

When to Fine-Tune with LoRA?

- **q_proj**: To adjust focus (e.g., focus on "Sahib" as subject → "Cap'n Sahib").
- **k_proj**: If focus matching fails (e.g., ignores "name" for "Sahib"—rare).
- **v_proj**: To adjust retrieved info (e.g., "Sahib" retrieves pirate identity).
- **o_proj**: If output lacks coherence (e.g., "Sahib" + "name" → messy—less common).

8. Explain me TrainingArguments for training an LLM?

The Parameters Explained:

1. `per_device_eval_batch_size=2`
    - **What it is**: How many samples (data points) each device (e.g., GPU) processes at once during evaluation (testing the model's performance).
    - **Why 2?**: Small batch size saves memory, useful if your GPU can't handle more during evaluation.

2. `per_device_train_batch_size=4` `# Increased from 2`
    - **What it is**: How many samples each device processes at once during training (updating the model).
    - **Why 4?**: Bigger than 2 (previous value) means faster training since more data is processed per step, but still fits in memory.

3. `save_strategy="epoch"` `# Save checkpoints`
    - **What it is**: When to save the model. "Epoch" means save after each full pass through the data (one epoch).
    - **Why?**: Checkpoints let you resume training or use the model later.

4. `save_total_limit=2` `# Keep last 2`
    - **What it is**: Limits how many saved checkpoints to keep. Only the last 2 are stored.
    - **Why?**: Saves disk space while keeping recent backups.

5. `gradient_accumulation_steps=4` `# Reduced from 16`
   - **What it is**: Number of small batches to combine before updating the model. Instead of updating with a batch of 4 (from `per_device_train_batch_size`), it waits until 4 batches (4 × 4 = 16 samples) are processed.
   - **Why reduced?**: Smaller value (from 16) means updates happen more often, which can speed up learning but uses less memory.

6. `gradient_checkpointing=True`
   - **What it is**: A trick to save memory by not storing all intermediate calculations during training. It recalculates them when needed.
   - **Why?**: Lets you train bigger models on limited GPU memory.

7. `gradient_checkpointing_kwargs={"use_reentrant": False}`
   - **What it is**: Extra setting for gradient checkpointing. `use_reentrant=False` makes it faster and more memory-efficient on modern systems.
   - **Why?**: Improves performance without breaking anything.

8. `learning_rate=2.0e-04`
   - **What it is**: How big of a step the model takes when adjusting its weights. Here, it's 0.0002.
   - **Why?**: Small steps help the model learn steadily without overshooting.

9. `log_level="info"`
   - **What it is**: Controls how much info is printed during training. "Info" shows useful updates (not too quiet, not too chatty).
   - **Why?**: Keeps you informed about progress.

10. `logging_steps=7`
    - **What it is**: How often (every 7 steps) to log progress (e.g., loss, accuracy).
    - **Why?**: Frequent updates help you monitor training.

11. `logging_strategy="steps"`
    - **What it is**: Tells it to log based on steps (not epochs or other intervals).
    - **Why?**: Ties logging to `logging_steps` for consistent tracking.

12. `lr_scheduler_type="cosine"`
    - **What it is**: How the learning rate changes over time. "Cosine" starts high, gradually

lowers it, then tapers off smoothly.
- **Why?**: Helps the model learn fast early on and fine-tune later.

13. `max_steps=-1`
    - **What it is**: Maximum number of training steps. -1 means no limit; it uses epochs instead.
    - **Why?**: Lets `num_train_epochs` control training duration.

14. `num_train_epochs=3` `# Reduced from 7`
    - **What it is**: How many full passes through the data to train for. Reduced from 7 to 3.
    - **Why?**: Shorter training time, maybe enough for the task or to test something quickly.

How They're Connected and Used:

Imagine training a model as teaching a kid to ride a bike:

- **Batch Sizes (**`per_device_train_batch_size`**,** `per_device_eval_batch_size`**)**: How many examples you show the kid at once. Small batches (4 for training, 2 for testing) keep it manageable for the GPU (like not overwhelming the kid).
    - **Gradient Accumulation (**`gradient_accumulation_steps`**)**: You don't correct the kid's balance after every wobble (batch of 4). You wait until 4 wobbles (16 samples) to give one big correction. Reducing from 16 to 4 means more frequent corrections, adapting faster.
    - **Memory Saving (**`gradient_checkpointing`**)**: Instead of remembering every move the kid makes, you only note key moments and figure out the rest later. This lets you teach a "bigger kid" (larger model) with the same bike (GPU).
    - **Learning Rate (**`learning_rate`**,** `lr_scheduler_type`**)**: You start with bigger nudges (0.0002) to get the kid moving, then ease off (cosine schedule) so they don't fall as they get better.
    - **Epochs and Steps (**`num_train_epochs`**,** `max_steps`**)**: You decide to practice 3 full laps (epochs) around the park, not 7, and don't cap the little steps (max_steps=-1) within those laps.
    - **Saving (**`save_strategy`**,** `save_total_limit`**)**: After each lap, you take a photo (checkpoint) but only keep the last 2 to save space.
    - **Logging (**`log_level`**,** `logging_steps`**,** `logging_strategy`**)**: Every 7 pedals, you

shout how they’re doing (info-level logs) to keep track.

Big Picture:

These settings balance speed, memory, and progress:

- Small batches + gradient accumulation = effective learning without crashing the GPU.
- Checkpoints + logging = you can pause, resume, and monitor without losing track.
- Learning rate + scheduler + epochs = steady improvement over 3 rounds, tailored to the task.

9. Explain me the connection bw steps and epochs ?

Definitions:

- **Epoch**: One full pass through the dataset (e.g., 1,000 samples).
- **Step**: One model update after processing an "effective batch size" of data.
- **If Steps are given than steps will be given more importance than Epochs.**

Key Parameters (from your setup):

- `per_device_train_batch_size=4` (samples per device per step)
- `gradient_accumulation_steps=4` (batches to accumulate before an update)
- `num_train_epochs=3` (full dataset passes)
- Assume 1 device, dataset size = 1,000 samples.

How Steps Are Calculated:

1. **Effective Batch Size** = `per_device_train_batch_size` × `gradient_accumulation_steps`
   = 4 × 4 = 16 samples per update.
2. **Steps per Epoch** = Total Samples ÷ Effective Batch Size
   = 1,000 ÷ 16 = 62.5 ≈ 63 steps (rounded up for leftovers).
3. **Total Steps** = Steps per Epoch × Number of Epochs
   = 63 × 3 = 189 steps.

Connection:

- **Epochs** (3) set how many full dataset cycles: 3 × 1,000 = 3,000 samples processed.
- **Steps** (189) are the updates within those cycles: 63 steps per epoch.

- **Batch Processing**: 4 samples processed per step, 16 samples per update (due to accumulation).

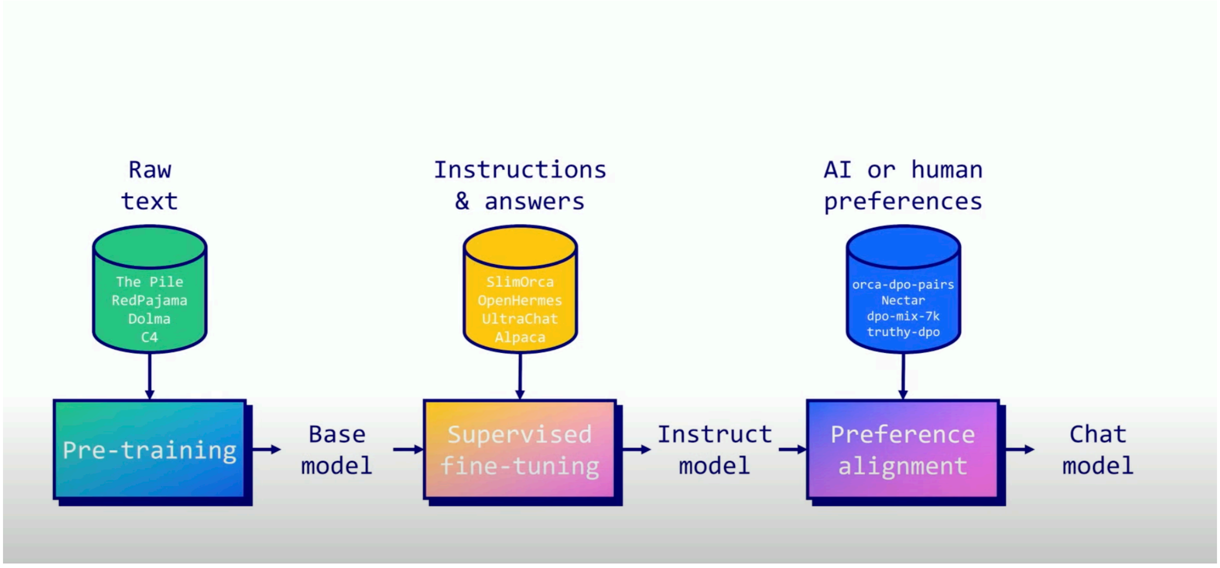
With Your Parameters:

- **Saving**: `save_strategy="epoch"` → Save after 63, 126, 189 steps. `save_total_limit=2` → Keep checkpoints from steps 126 and 189.
- **Logging**: `logging_steps=7` → Log every 7 steps (~9 logs per epoch: 63 ÷ 7 ≈ 9).
- **Learning Rate**: `lr_scheduler_type="cosine"` → Adjusts over 189 steps.

Analogy:

- **Dataset**: 1,000 cookies.
- **Batch Size (4)**: Oven holds 4 cookies.
- **Accumulation (4)**: Tweak recipe after 4 batches (16 cookies).
- **Steps per Epoch**: 1,000 ÷ 16 ≈ 63 tweaks.
- **Epochs (3)**: Bake all cookies 3 times = 189 tweaks.

10. Explain from 1000m perspective how fine-tuning life-cycle looks like?

1. 

11. What is the primary task on which SFT is done?

1. SFT also uses next-token prediction as training objective with goal to minimize Cross entropy loss as loss function.
   Simple perpective is to minimize the difference between the model's predicted outputs and the ground truth (the expected responses in the dataset)

12. Is it “Normal” Next token prediction and How is Next Token Prediction different from Pre-Training and SFT?

1. Yes it “Normal” —> Next Token prediction.

   Pre-training: During pre-training, the model learns from a massive, diverse corpus (e.g., web texts, books) to predict the next token in a general language modeling task. The goal is to build a broad understanding of language.

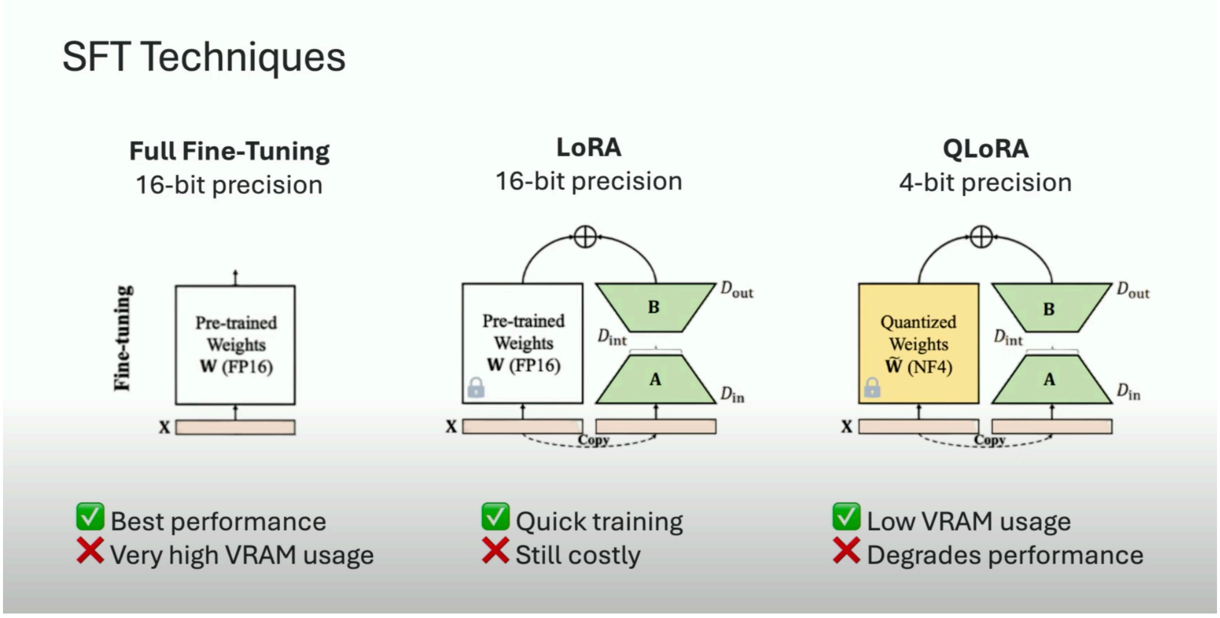
   SFT: In SFT, the dataset is much more specific—typically pairs of instructions and desired responses (as shown in the image). The model still uses next-token prediction, but now it’s optimizing to generate responses that align with the supervised examples, such as answering questions or following instructions in a particular style.

13. What is Preference Alignment in LLM fine-tuning, and what is its goal?

1. Preference Alignment fine-tunes an LLM using human feedback, training on datasets with instructions, chosen answers (preferred), and rejected answers (less desirable). The goal is to increase the model’s probability of generating preferred responses, making it more helpful and aligned with human preferences (e.g., using methods like DPO).
   Ex - DPO, PPO, KTO, GRPO (DeepSeek) etc.

14. Different SFT technqiues?

1.



SFT Techniques

Full Fine-Tuning
16-bit precision

LoRA
16-bit precision

QLoRA
4-bit precision

Fine-tuning
Pre-trained Weights W (FP16)
x
Pre-trained Weights W (FP16)
B
A
D_{out}
D_{int}
D_{in}
Copy
x
Quantized Weights $\bar{W}$ (NF4)
B
A
D_{out}
D_{int}
D_{in}
Copy
x

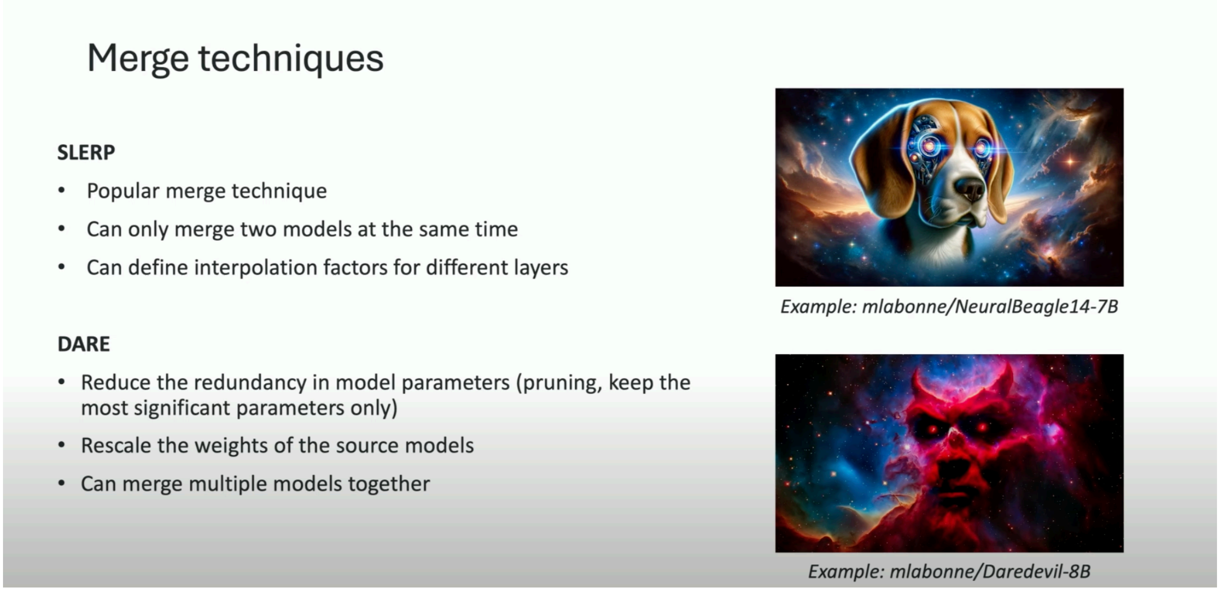
✅ Best performance
❌ Very high VRAM usage

✅ Quick training
❌ Still costly

✅ Low VRAM usage
❌ Degrades performance

15. What is Model merging and different technqiues?

1.



Merge techniques

SLERP

- Popular merge technique
- Can only merge two models at the same time
- Can define interpolation factors for different layers

Example: mlabonne/NeuralBeagle14-7B

DARE

- Reduce the redundancy in model parameters (pruning, keep the most significant parameters only)
- Rescale the weights of the source models
- Can merge multiple models together

Example: mlabonne/Daredevil-8B

2. **SLERP (Spherical Linear Interpolation)** is a model merging technique that combines two pre-trained LLMs by interpolating their parameters in a high-dimensional space, treating the model weights as points on a hypersphere. Unlike simple linear interpolation (LERP), which can lead to a loss of performance by averaging weights directly, SLERP ensures a smooth transition by following a spherical path, preserving the geometric properties and unique characteristics of each model.

3. **DARE (Drop And Rescale)** is a technique used to prepare task vectors for model merging by reducing interference between parameters of multiple models. It's often combined with other methods like TIES (Trim, Elect Sign, Merge) to merge multiple LLMs into a single model, enhancing their combined strengths.

16. How Transformers work at time of trainingand Inference. —> here by Neils Rogge

17. What is "Policy" in Reinforcemnet Learning?

1. a **policy** is like a strategy or a set of rules that an agent (the learner) uses to decide what actions to take in different situations. Think of it as a guide that tells the agent, "When you're in this spot, do this."

18. Explain the Idea of "Agent","Policy","Action","Reward","State" interms of language model?

- **State**: The input prompt or the current sequence of tokens (e.g., "Shanghai is a city in").
- **Agent**: The language model itself, which generates text.
- **Action**: The next token the model generates (e.g., "China").
- **Reward Model**: A system that scores the model's outputs, giving high scores for good text (e.g., correct and coherent) and low scores for bad text (e.g., incorrect or nonsensical).

- **Policy**: The model’s strategy for picking the next token, represented as a probability distribution over possible tokens (e.g., 85% for "China," 10% for "Beijing").

19. What is KL diveregence? How it aligns in LLM Alignment phase?

1. **Definition:** KL divergence is a statistical measure of how different two probability distributions are. In the context of LLMs, it compares the model’s output probabilities before and after an update (e.g., old policy vs. new policy).
2. **Very Important** It is useful in technqiues like PPO and GRPO where it acts as a regularization term in the loss function because at times the model might over-optimize for the current batch of rewards, drifting too far from its original behavior and losing generalization. KL divergence keeps updates conservative, preserving the model’s prior knowledge while aligning it with new goals.
3. **Example:** If “4” gets a high reward for “2+2,” the policy might over-focus on always outputting “4” unless KL divergence penalizes excessive shifts.

20. Taking Same exmaple where Input was “ Where is Shanghai? ” —> We got “ Shanghai is in ”

Now to calculate next token Langugae model will first generate **Logits** and then those will be normalised to **Probabilities**. So now the question what’s the diff bw logits and probabilities?

What is Probabilities?

1. **Definition**: Probabilities are normalized values that represent the likelihood of each possible outcome (e.g., each token in the vocabulary). In a language model, probabilities are derived from logits and are used to decide which token to generate next.
   How They’re Calculated: Probabilities are typically obtained by applying a **softmax function** to the logits. The softmax function converts the raw logits into a probability distribution, ensuring that:
   - All probabilities are between 0 and 1.
   - The probabilities sum to 1 across all possible outcomes (tokens).

What are Logits?

1. **Definition**: Logits are the raw, unnormalized scores output by a model before they are converted into probabilities. In a language model, logits represent the model’s "confidence"

or preference for each possible next token (word or subword) in the vocabulary, based on the current input (state).

How They're Calculated: In a neural network like a language model, the logits are typically the output of the final layer before any normalization. For example, in a transformer-based language model (like GPT), the model processes the input prompt (state) and produces a vector of logits, where each value in the vector corresponds to a token in the vocabulary.

Characteristics:

- Logits are **unnormalized**, meaning they can be any real number (positive, negative, or zero).
- They don't directly represent probabilities; they're more like raw scores that indicate the model's preference for each token.
- Higher logit values indicate a stronger preference for a particular token, but the values themselves don't have a bounded range(e.g., they're not between 0 and 1).

Using the logits from the earlier example:

Logits: " China ": 4.2, "Beijing": 1.5, "Cat": -2.0, "Pizza": -3.5

Apply softmax:

- e^4.2 ≈ 66.69
- e^1.5 ≈ 4.48
- e^-2.0 ≈ 0.14
- e^-3.5 ≈ 0.03

Sum of exponentials: 66.69 + 4.48 + 0.14 + 0.03 = 71.34

Probabilities:

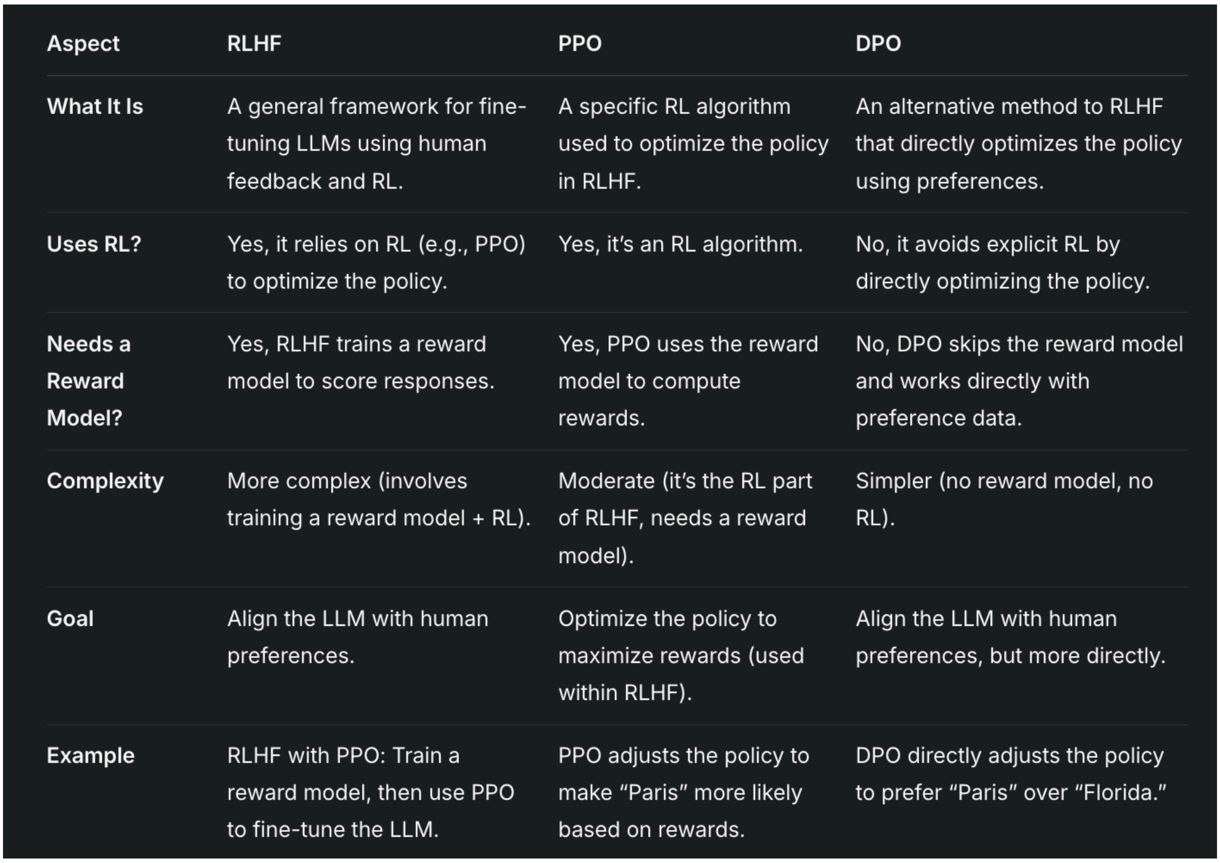
- "China": 66.69 / 71.34 ≈ 0.935 (93.5%)
- "Beijing": 4.48 / 71.34 ≈ 0.063 (6.3%)
- "Cat": 0.14 / 71.34 ≈ 0.002 (0.2%)
- "Pizza": 0.03 / 71.34 ≈ 0.0004 (0.04%)

These probabilities sum to 1, and they represent the likelihood of each token being the next one. This is the policy π(s_t) mentioned in the slide from your previous question, where the model samples the next token (action) based on these probabilities.

21. What are the major difference bw RLHF,PPO and DPO?

These are the differences

- RLHF is the overarching framework that often uses PPO as its RL algorithm.
- PPO is a specific tool (an RL algorithm) used within RLHF to optimize the model's policy.
- DPO is an alternative to RLHF that achieves the same goal (aligning the model with human preferences) but in a different, simpler way without using RL or a reward model.

- 

Aspect	RLHF	PPO	DPO
What It Is	A general framework for fine-tuning LLMs using human feedback and RL.	A specific RL algorithm used to optimize the policy in RLHF.	An alternative method to RLHF that directly optimizes the policy using preferences.
Uses RL?	Yes, it relies on RL (e.g., PPO) to optimize the policy.	Yes, it's an RL algorithm.	No, it avoids explicit RL by directly optimizing the policy.
Needs a Reward Model?	Yes, RLHF trains a reward model to score responses.	Yes, PPO uses the reward model to compute rewards.	No, DPO skips the reward model and works directly with preference data.
Complexity	More complex (involves training a reward model + RL).	Moderate (it's the RL part of RLHF, needs a reward model).	Simpler (no reward model, no RL).
Goal	Align the LLM with human preferences.	Optimize the policy to maximize rewards (used within RLHF).	Align the LLM with human preferences, but more directly.
Example	RLHF with PPO: Train a reward model, then use PPO to fine-tune the LLM.	PPO adjusts the policy to make "Paris" more likely based on rewards.	DPO directly adjusts the policy to prefer "Paris" over "Florida."

28. What is PPO (Proximal Policy optimisation)?

PPO (Proximal Policy Optimization) is a reinforcement learning (RL) algorithm used to fine-tune a model (like an LLM) by optimizing its policy (the probability distribution over actions, such as token generation in an LLM) to maximize rewards while ensuring stability. In the context of LLMs, PPO is often used in RLHF (Reinforcement Learning from Human Feedback) to align the model with human preferences.

How PPO Works (Step by Step):

- **Step 1: Start with an instruction-tuned model (e.g., after Supervised Fine-Tuning, SFT).**
    - Begin with an LLM pre-trained and fine-tuned on instruction data, ready to generate responses based on its current policy.
- **Step 2: Generate responses to prompts using the current policy.**
    - **Policy in LLM Terms:** The policy is the **probability distribution** over possible next tokens given a prompt, controlled by the model's weights. For example, for "What's a good travel spot?" the policy might assign 60% probability to "Paris" and 40% to "Florida." The model samples or selects a response (e.g., "Paris") based on these

probabilities.

  - This step collects data for optimization by rolling out the current policy.
- **Step 3: Evaluate those responses with a reward model to get a reward score.**
  - A separate reward model (trained on human feedback, e.g., preference pairs) scores each response. For example, “Paris” might get 0.9 (highly preferred), while “Florida” gets 0.1 (less preferred).
  - The reward model provides the raw feedback signal, distinct from the critic’s role (see Step 4).
- **Step 4: Use the critic model to estimate the advantage for each response.**
  - The critic (a separate neural network) predicts the **value** (expected future reward) of the state (e.g., the prompt or prompt+response).
  - **Advantage Calculation:** Advantage = Reward + Discounted Future Value - Critic’s Value Estimate (A = R + γV(next) - V(current)). For simplicity, if no future steps are considered (single response), it’s often just Reward - Critic’s Baseline.
    - Example: If “Paris” has reward 0.9 and the critic’s baseline is 0.5, advantage = 0.9 - 0.5 = +0.4 (good response). “Florida” with 0.1 gets -0.4 (poor response).
- Advantage tells the policy how much better (or worse) each response is compared to the average expectation, guiding updates.
- **Step 5: Update the policy to favor high-reward responses, using a clipped objective and KL penalty for stability.**
  - **Policy Update:** The actor (the LLM’s policy) adjusts its weights to increase the probability of high-advantage responses. For example, “Paris” might go from 60% to 80% probability, shifting the distribution.
    - **Clipped Objective:** PPO uses a clipped loss function (e.g., min(ratio * A, clip(ratio, 1-ε, 1+ε) * A)) to limit how much the policy changes per update, where “ratio” is the probability ratio between new and old policies, and ε (e.g., 0.2) caps extreme shifts.
    - **KL Penalty:** Adds KL divergence (measuring difference between old and new policy distributions) to the loss, penalizing large deviations to maintain stability.
  - Result: The model learns gradually, avoiding catastrophic forgetting or overfitting to noisy rewards.
- **Step 6: Repeat the process iteratively.**
  - Generate new responses with the updated policy, score them, compute advantages, and refine further, improving alignment over multiple iterations.

29. What is the difference of Actor,Critic and Reward Model ?

1. **Actor (The Policy Model - The LLM Itself)**
    - **What It Is:** The actor is the LLM—the model generating responses. It's called the "actor" because it takes actions (outputs tokens or responses) based on its current policy.
    - **Policy:** The policy is the actor's strategy, defined as a **probability distribution** over possible next tokens for a given prompt. For example, for "What's a good travel spot?" it might assign 60% to "Paris" and 40% to "Florida."
    - **Role:** Decides *what to say*. It's the part being trained to improve based on feedback.
        - **Example:** You ask the LLM, "Solve 2+2." The actor outputs "4" (70% chance) or "5" (30% chance) based on its current weights.
        - **Goal:** Adjust its probabilities (via weight updates) to favor better responses over time.
2. **Reward Model (The Scorer)**
    - **What It Is:** A separate model (often a smaller neural network, sometimes an LLM) trained on human feedback (e.g., preference pairs like "Paris > Florida") to assign scores to responses.
    - **Role:** Evaluates *how good* a response is, giving a raw reward score. It's like a judge saying, "This answer gets a thumbs-up or thumbs-down."
    - **Example:** For "Solve 2+2," the reward model scores "4" as 1.0 (correct) and "5" as 0.2 (wrong). For "What's a good travel spot?" it might give "Paris" 0.9 and "Florida" 0.1 based on human preferences.
    - **Key Point:** It only scores the response—it doesn't care about what happens next or how good the actor's overall strategy is. It's static feedback.
3. **Critic (The Value Estimator)**
    - **What It Is:** Another separate neural network in PPO that predicts the **value** of a state or action—essentially, how good the situation is in terms of expected future rewards.
    - **Role:** Provides a baseline to measure *how much better or worse* the actor's response is compared to the average. It's like a coach saying, "This move was smarter than your usual play."
    - **Example:** For "Solve 2+2," the critic might estimate the value of the prompt state (before answering) as 0.5 (average expected reward). If "4" gets a reward of 1.0, the critic helps calculate that it's better than average.
    - **Key Point:** Unlike the reward model, the critic looks at the bigger picture (future potential), not just the immediate score. It's trained alongside.
4. ## How They Work Together in PPO (Simple Analogy)

    Imagine training an LLM to recommend travel spots:

    - **Actor (LLM):** Suggests “Paris” or “Florida” based on its current mood (policy probabilities).
    - **Reward Model (Judge):** Gives “Paris” a 0.9 (“Great choice!”) and “Florida” a 0.1 (“Meh”).
    - **Critic (Coach):** Says, “On average, your suggestions are worth 0.5. ‘Paris’ at 0.9 is a win (+0.4 advantage), ‘Florida’ at 0.1 is a loss (-0.4 advantage).”
    - **Result:** The actor learns to suggest “Paris” more often, guided by the critic’s advice and the reward model’s scores.

30. What is advantage in PPO?

1. Advantage estimation calculates how much better or worse an action is compared to a baseline (e.g., average reward or expected value). In RL, it’s typically “reward minus baseline” (A = R - V). Positive advantage means the action is worth reinforcing; negative means it should be discouraged.
2. Simple terms - How well your action was as compared to average action taken here

31. What is the need of Separate Critic model? (Specifically Reward - Critic’s Baseline in PPO)

- The advantage calculation —> (Reward - Critic’s Baseline) is all about figuring out *how much better or worse* a response is compared to what the model typically expects. Here’s the intuition:
- **Critic’s Baseline (Value):** The critic predicts the **value** of a state—essentially, the average expected reward if the actor follows its current policy from that point. For a prompt like “What’s a good travel spot?” the critic might say, “On average, responses here are worth 0.5.” It’s a benchmark of “normal” performance.
- **Reward:** The reward model gives the actual score for a specific response (e.g., “Paris” gets 0.9). This is the real outcome.
- **Advantage (Reward - Baseline):** Subtracting the baseline from the reward tells us if the response beat expectations (positive advantage) or fell short (negative advantage).
    - Example: Reward 0.9 - Baseline 0.5 = +0.4 → “Paris” is better than average, so reinforce it.
    - Reward 0.1 - Baseline 0.5 = -0.4 → “Florida” is worse, so discourage it.
- **Why This Matters:**
    - If we just used the raw reward (0.9 for “Paris”), we’d always push the model toward

        higher rewards, even if they're already decent. The advantage focuses training on what's *exceptional* (above average) or *poor* (below average), making learning more efficient.
    - Without a baseline, small differences in rewards might over-influence updates. The critic's value normalizes this, grounding the process in context.
- **When No Future Steps Are Considered:**
    - In your note, you mentioned "if no future steps are considered (single response), it's often just Reward - Critic's Baseline." This simplifies the full formula (A = R + γV(next) - V(current)) because, for a single-turn task (like answering a question), there's no "next state" to predict. The critic just estimates the value of the current state (prompt or prompt+response), and the advantage becomes a direct comparison of reward to that baseline.
    - So, A = R - V(current) is the practical version here.

32. Pls explain the whole crtic, actor , reward , advantage terms by taking an example?

Example Prompt - What is the capital of france?

Step 1: Actor Generates a Response

- Prompt: "What is the capital of France?"
- Actor's policy: 70% chance "Paris," 30% chance "Florida" (bad initial guess).
- Output: "Paris" (sampled from the distribution).

Step 2: Reward Model Scores It

- Reward model (trained on correctness or human feedback): "Paris" = 1.0 (correct), "Florida" = 0.1 (wrong).
- Reward for "Paris" = 1.0.

Step 3: Critic Estimates Value

- **State:** The prompt "What is the capital of France?" (before the response).
- Critic's job: Predict the average expected reward for this prompt based on the current policy. Suppose the actor's policy is shaky (sometimes says "Florida"), so the critic estimates V(current) = 0.6 (average of good and bad answers).
- **No Future States Here:** This is a single-turn task—once the actor answers "Paris," the interaction ends. There's no "next state" like in a game (e.g., chess moves). So, γV(next) = 0, and the advantage simplifies to R - V(current).

Step 4: Calculate Advantage

- Reward (R) = 1.0 for "Paris."
- Critic's Baseline (V(current)) = 0.6.
- Advantage = 1.0 - 0.6 = +0.4.

- Meaning: “Paris” is 0.4 units better than the actor’s average performance for this prompt.

Step 5: Update Policy

- The actor adjusts its weights to increase the probability of “Paris” (e.g., from 70% to 85%), guided by the +0.4 advantage, using PPO’s clipped loss and KL penalty.

33. Can you explain what’s the meaning of “ Future states ” in the above example?

- **Single-Turn Task (example abive):** There are no future states! The task ends with one response (“Paris”), so the critic only evaluates the current state (the prompt or prompt+response). The advantage is just R - V(current) because there’s no sequence of actions to consider.
- **Multi-Turn Example (For Contrast):** If the task were a dialogue—“What’s the capital of France?” → “Paris” → “What’s its population?”—the critic would estimate V(next) for the next state (after “Paris”), and the full formula (R + γV(next) - V(current)) would apply. But for our above example question, it’s one-and-done.
- **Why “Future States” in the Formula?** The full advantage formula comes from RL problems like robotics or games, where actions lead to new states with more rewards later. In LLM single-response tasks, we simplify it because there’s no “later.”

34. What is “ ref_model ” in PPOTrainer class in trl?

1. The ref_model argument in the PPOTrainer (from the trl library) refers to a **reference model** that is used to compute the **KL divergence** (a measure of how much two probability distributions differ) between the current policy (the model being fine-tuned) and a baseline policy (the reference model). Let’s break this down in simple terms.
2. **Role in PPO**:

- In PPO, when you fine-tune a language model (LLM), you’re updating its policy (the probability distribution over tokens, like 60% for “Paris” and 40% for “Florida”).
- The reference model represents the *original policy* (the model’s behavior before fine-tuning) or a *baseline policy* (e.g., the model after supervised fine-tuning, SFT).
- PPO uses the reference model to calculate how much the current policy has changed during fine-tuning. This is done by computing the KL divergence between the current policy’s probabilities and the reference model’s probabilities.

Why Is This Important?

- PPO wants to make sure the policy doesn't change too much in one step, as big changes can make the model unstable or cause it to forget what it already knows (a problem called **catastrophic forgetting**).
- The KL divergence acts as a "penalty" in PPO's objective function. If the current policy deviates too far from the reference model, the penalty reduces the update size, keeping the fine-tuning process stable.

In Simple Terms:

- Think of the reference model as a "teacher" who reminds the student (the model being fine-tuned) not to stray too far from what it originally knew.
- For example, if the reference model gives "Paris" a 60% probability, but the fine-tuned model starts giving "Paris" a 99% probability, the KL divergence will be large, and PPO will reduce the update to prevent such a drastic change.

Setting it **"None"** means PPOTrainer will automatically set a copy of Model loaded for generating answers to User's Prompt.

30. (Important) How does PPO ensure stability during fine-tuning? or What is the use of KL divergence penalty or clipping used for?

1. **Clipping**: PPO clips the probability ratio (new policy probability / old policy probability) to a range (e.g., 0.8 to 1.2 with cliprange=0.2), preventing large policy updates that could destabilize the model.

2. **KL Penalty**: PPO adds a KL divergence penalty to its objective, penalizing the policy if it deviates too far from a reference model (ref_model).Now why it is important because think of a way where our LLM has found to spoof the reward model and generate high rewards which inturn are changing the policy of LLM and LLM has started throwing Gibrrish or same words again and again to cheat the model.

31. What is DPO?

1. DPO is a fine-tuning method for LLMs that directly optimizes a model's policy based on human preferences, bypassing the need for a separate reward model or reinforcement learning complexities like those in PPO. It's designed to align LLMs with human feedback

more efficiently by using a simple classification-like approach, where the model learns to prefer “better” responses over “worse” ones based on preference pairs.

How DPO Works (Step by Step):

- Start with an instruction-tuned model (e.g., after Supervised Fine-Tuning, SFT).
- Collect pairs of responses to the same prompt: one “preferred” (chosen) and one “less preferred” (rejected), based on human feedback (e.g., “Paris” as chosen, “Florida” as rejected for “What’s a great travel spot?”).
- Model the preference as a binary choice: the goal is to increase the probability of the preferred response and decrease the probability of the rejected one.
- Use a loss function (derived from a Bradley-Terry model) that compares the model’s predicted probabilities for both responses, adjusting the policy to favor the preferred one.
- Update the model directly using this loss, without needing an external reward model or iterative RL steps—keeping it computationally simpler and stable.

32. Does DPO uses Reinforcemnet learning ? —>

No, DPO does *not* use reinforcement learning in the traditional sense. While DPO is inspired by RL concepts and achieves a similar goal to RLHF (aligning a language model with human preferences), it avoids explicit RL by directly optimizing the model using a supervised learning-like approach

What is RL ➔ a type of learning mechanism where a agent, changes it’s state based on action in a given environment and then receives a **reward** from a reward model.

1. **Policy**: The strategy the agent uses to choose actions (e.g., the language model’s probability distribution over tokens, as we discussed with logits and probabilities).
2. **Reward**: A numerical signal that tells the agent how good or bad its action was.
3. **Optimization**: RL algorithms (like PPO) iteratively update the policy by exploring the environment, collecting rewards, and adjusting the policy to maximize the expected reward.

What actually DPO Does?

1. DPO fine-tunes a language model (e.g., an instruction-tuned LLM) using human preference data, such as pairwise comparisons (e.g., Response A: “Paris” is preferred over Response B: “Florida” for the prompt “What’s the capital of France?”).
2. Instead of using RL, DPO directly optimizes the model’s policy (its probability distribution

over tokens) using a loss function that's designed to increase the likelihood of preferred responses and decrease the likelihood of dispreferred responses.

3. This optimization is done in a single step (or a series of supervised learning steps), similar to how you'd fine-tune a model with supervised learning, rather than the iterative exploration-and-reward process of RL.

Why It's Not RL:

- **No Exploration**: In RL, the agent explores the environment by trying different actions (e.g., generating various responses) and learning from the rewards. DPO doesn't do this—it works with a fixed dataset of preference pairs and directly optimizes the model based on that data.
- **No Reward Model or Iterative Updates**: RLHF uses a reward model and an RL algorithm (like PPO) to iteratively update the policy based on rewards. DPO skips the reward model and RL algorithm, instead using a closed-form loss function to update the model in a supervised manner.
- **Supervised Learning-Like Approach**: DPO's optimization process is more akin to supervised fine-tuning (SFT). It uses a dataset of preferences and directly adjusts the model's parameters to align with those preferences, without the trial-and-error process of RL.

33. Code example of DPO working ? —> From Maximme Labonne

34. Difference bw prefrence data of DPO and PPO?

- **DPO (Direct Preference Optimization)**:
  - Imagine training a chatbot. You give it two responses to a question like "How's the weather?"
  - Response A: "It's sunny and warm!"
  - Response B: "Dunno, check outside."

  A human labels "A is better than B." DPO directly adjusts the model to prefer generating responses like A over B, no reward function needed.

- **PPO (Proximal Policy Optimization)**:

  Same scenario, but instead of direct preferences, you first train a reward model. The reward model might assign:
  - Response A: +1 (positive, helpful)
  - Response B: -0.5 (unhelpful)

  PPO then uses these rewards to tweak the chatbot's policy, ensuring it gradually leans toward responses like A while keeping updates stable.

DPO skips the reward step and learns from the "A > B" preference alone, while PPO builds and optimizes around the reward scores.

33. A sample code to show working of PPO ?

1. Link —> here

34. What's the difference between SFT, RLHF, and DPO in the context of LLM fine-tuning?

- SFT (Supervised Fine-Tuning): Fine-tunes a pre-trained LLM on a dataset of instruction-response pairs (e.g., "What's the capital of France?" → "Paris") to make it better at following instructions. It's a supervised learning approach.
- RLHF (with PPO): Builds on SFT by using human feedback to train a reward model, then uses PPO (reinforcement learning) to fine-tune the LLM to maximize the reward, aligning it with human preferences.
- DPO: Also builds on SFT but directly fine-tunes the LLM using preference data (e.g., "Paris" is preferred over "Florida") with a supervised learning-like approach, skipping the reward model and RL.

35. What is GRPO?(Group Relative Policy Optimization)

1. It is Reinforcemnet learning technique similar to PPO etc. In this techqniue to optimise **model policy** (here model refers to LLM). It simplifies the process by **eliminating the need for a separate critic (value)** model, as used in PPO, making it more memory-efficient and scalable. The main theme of GRPO is to leverage **relative performance within a group of outputs to guide model improvement**, focusing on efficiency and stability in post-training LLMs.

36. How GRPO works?

1. Start with an instruction-tuned model (e.g., after Supervised Fine-Tuning, SFT).
2. For each prompt, generate multiple responses (a "group") using the current model policy (e.g., for "Solve 2+2," generate answers like "4," "5," "3").

3. Score each response using a reward signal, which could come from a reward model (trained on human preferences) or a rule-based metric (e.g., accuracy, format correctness).
    1. **Note:** GRPO's innovation lies in simplifying RL by eliminating the critic model (unlike PPO), using group averages instead. This makes it flexible and practical to use either a reward model (LLM-based or otherwise) or lightweight rule-based metrics, depending on the task, without the overhead of a separate value estimator.
4. Calculate the average reward across the group as a baseline (e.g., if scores are 1.0, 0.2, 0.5, the average is 0.57).
5. Compare each response's score to the group average to estimate its "**advantage**" (e.g., "4" with 1.0 is above average, "5" with 0.2 is below).
6. Update the model policy using a clipped loss function (like PPO) to reinforce high-advantage responses, with **KL divergence** regularization to limit drastic shifts.
    1. **KL Divergence:** Ensures the updated policy doesn't stray too far from the original, maintaining stability.

37. (Important) What is Advantage Estimation in General and Why Is It Used So Many Times in RLHF Techniques?

1. **Definition:** Advantage estimation calculates how much better or worse an action is compared to a baseline (e.g., average reward or expected value). In RL, it's typically "reward minus baseline" (A = R - V). Positive advantage means the action is worth reinforcing; negative means it should be discouraged.

2. **Why It's Used in RLHF:**
    - RLHF (Reinforcement Learning from Human Feedback) aims to align LLMs with human preferences, but raw rewards alone don't tell the full story—some actions might be good but not exceptional. Advantage normalizes this, focusing updates on what's truly better or worse than expected.
    - It's efficient: Instead of blindly maximizing all rewards, RLHF techniques (PPO, GRPO) prioritize actions with high advantage, speeding up learning.
    3. ◆ **In GRPO:** Advantage is estimated simply as the response's reward minus the group average (Step 5). This drives policy updates toward high-quality outputs without needing a critic, a common theme in RLHF for balancing efficiency and alignment.

38. Is GRPO a RL Algorithm?

1. Yes, GRPO is a reinforcement learning algorithm. It optimizes the model's policy to

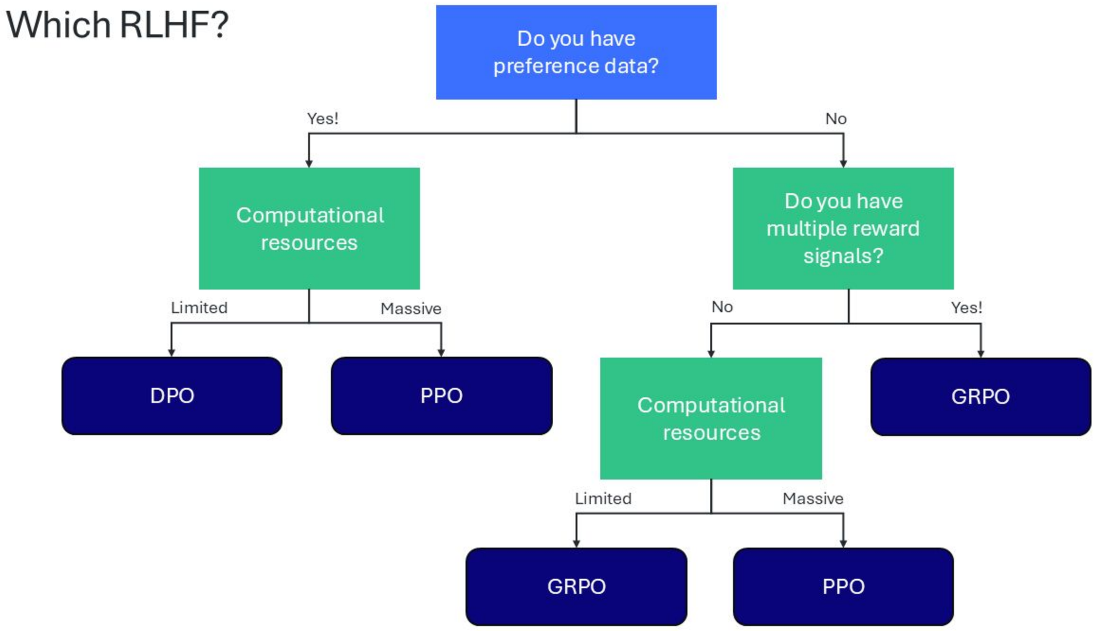
maximize rewards, using techniques like clipped loss and KL divergence (borrowed from PPO), but it innovates by replacing the critic with group-based advantage estimation, making it a streamlined RL approach.

39. For Understanding each and every GRPO component?

1. Start from here from Hf ➜ Here
2. Great practical blog ➜ Mahesh Deshwal

40. When to use GRPO , DPO and PPO? (Great Post)

1.



2. Refer to this post from Maximme Labonne