LLM Fine-Tuning: Techniques and Best Practices

Dr. Pejman Ebrahimi

Deep Learning and Advanced AI Techniques Course

Introduction

- What is fine-tuning and why it matters?
- Common use cases for fine-tuning LLMs
- Fine-tuning vs pre-training: key differences
- The fine-tuning lifecycle overview

Concept

Fine-tuning is the process of taking a pre-trained LLM or SLM and further training it on a specific dataset to adapt it for particular tasks. Unlike pre-training (which requires massive datasets and computational resources), fine-tuning is more efficient as it builds upon existing knowledge. Common use cases include domain specialization (adapting to legal or medical text), instruction following (making models respond to commands), and alignment with human values. The fine-tuning lifecycle typically involves data preparation, training, evaluation, and deployment.

Fine-tuning process

Precision Formats

The choice between these formats affects training stability, memory usage, and hardware compatibility.

BF16 (Brain Float 16)

- 8 exponent bits, 7 mantissa
- Range: ~5.96e-38 to ~3.39e38
- For: Large-scale training
- Hardware: Newer GPUs (A100, H100)

FP16 (Half Precision)

- 5 exponent bits, 10 mantissa
- Range: ~5.96e-8 to 65,504
- For: Smaller models
- Hardware: Broader support (V100, T4)

Floating Point Formats

bfloat16: Brain Floating Point Format

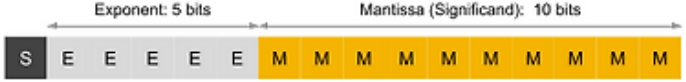
Range: ~1e^{-38} to ~3e^{38}

fp32: Single-precision IEEE Floating Point Format

Range: ~1e^{-38} to ~3e^{38}

fp16: Half-precision IEEE Floating Point Format

Range: ~5.96e^{-8} to 65504



Training Data Considerations

- Limited data can cause overfitting
- Model might learn chat template
- Special tokens leakage ([INST], <s>)
- Quality over quantity is crucial
- Clean, diverse examples work best
- Validation splits help prevent overfitting

```
{
    "instruction" : "What is 2+2?",
    "output"      : "2 + 2 equals 4.",
    "instruction" : "What is 4+4?",
    "output"      : "4 + 4 equals 8.",
    ...
},
```

```
{
    "instruction" : "What is 2+2?",
    "output"      : "2 + 2 equals 4.",
},
{
    "instruction" : "How are you?",
    "output"      : "I'm doing fine!",
},
{
    "instruction" : "Flip a coin.",
    "output"      : "I got heads!",
}
```

Top-K vs Top-P Sampling

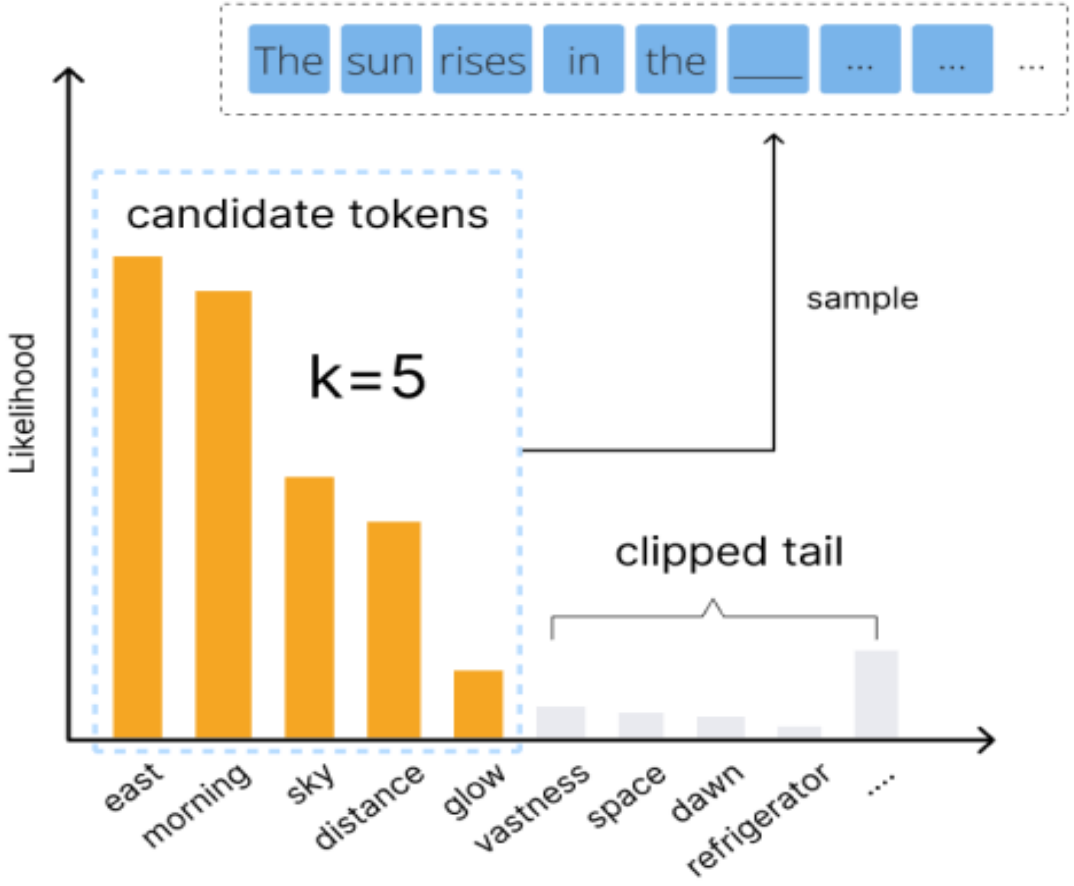
These parameters balance creativity and coherence in generated text. Top-K offers more predictable control, while Top-P adapts to the context's uncertainty (Simply, Top-K or Top-P sampling to make the output more interesting and creative).

Top-P (Nucleus) Sampling

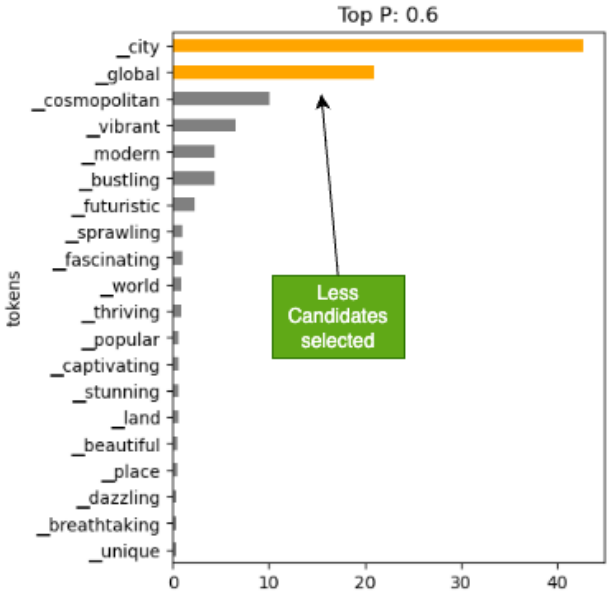
- Samples until prob > p
- Adaptive diversity
- Better for creative outputs
- Example: p=0.9 → 'Bartered with a mermaid'

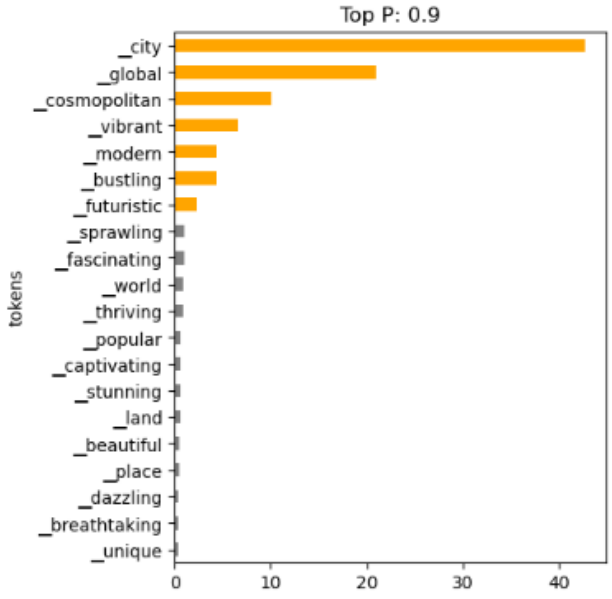
Top-K Sampling

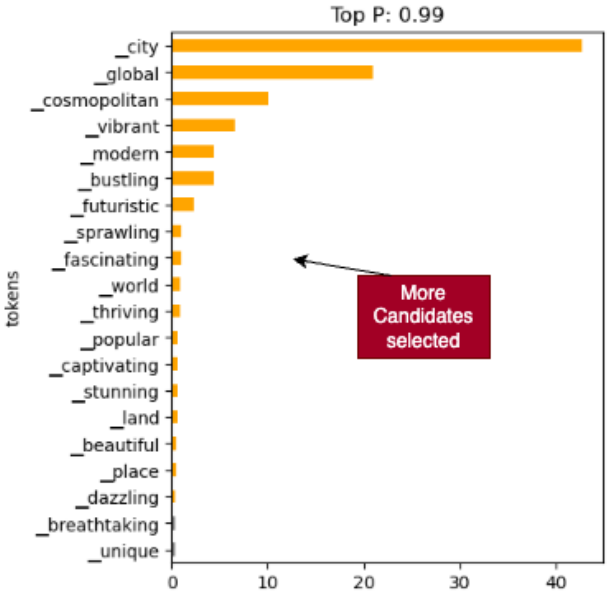
- Picks top k most likely tokens
- Fixed diversity control
- Good for consistent tone
- Example: k=30 → 'Swabbed the deck'



Model Output: Next token Probability Distribution







Low Top P

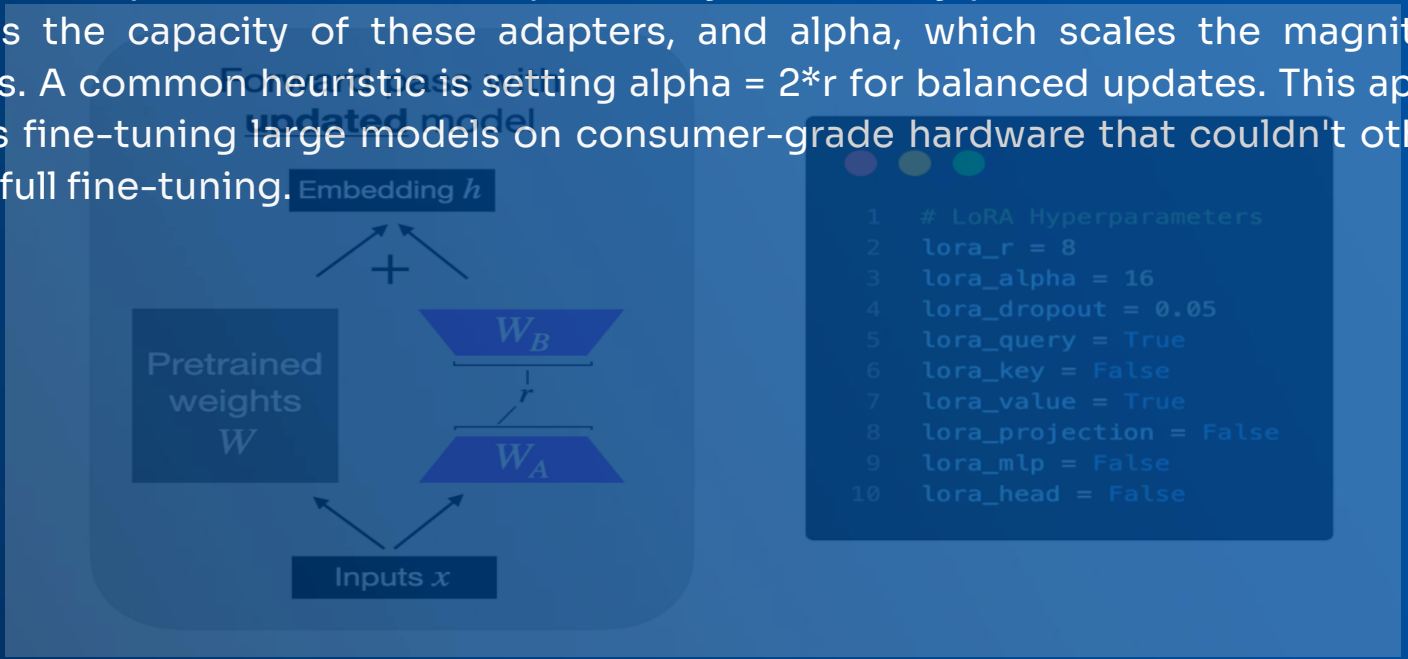
Less candidates selected before random sampling. More confident predictions, but less diverse

High Top P

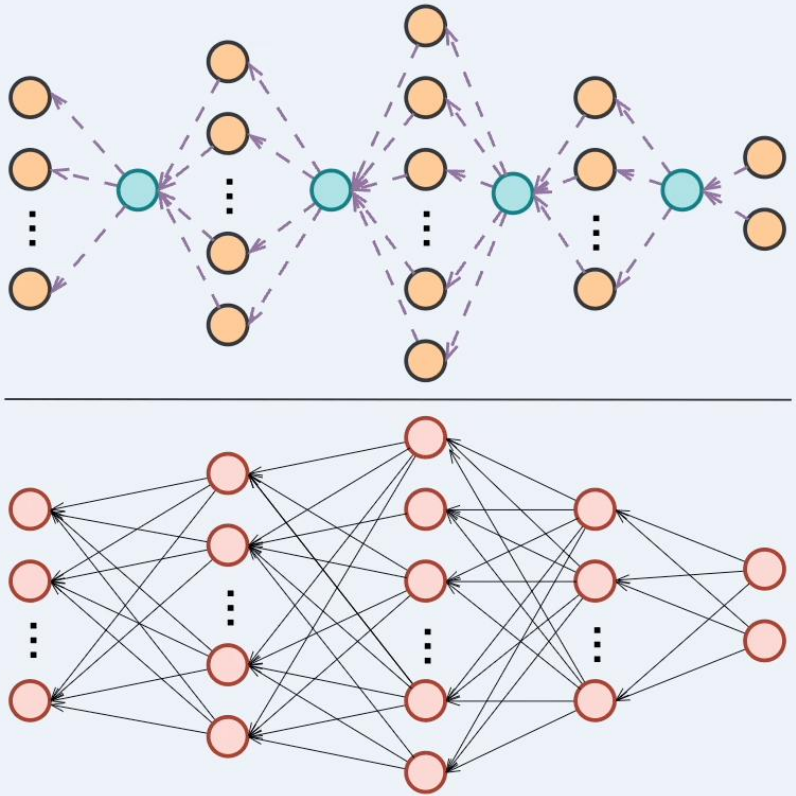
More candidates selected. More diverse generation

LoRA Fine-Tuning Approach

LoRA (Low-Rank Adaptation) is an efficient fine-tuning technique that dramatically reduces memory requirements. Instead of updating all model weights, LoRA inserts small trainable "adapter" matrices into specific layers. The key parameters are rank (r), which controls the capacity of these adapters, and alpha, which scales the magnitude of updates. A common heuristic is setting alpha = 2*r for balanced updates. This approach enables fine-tuning large models on consumer-grade hardware that couldn't otherwise handle full fine-tuning.



LoRA Fine Tuning



Gradient flow

Additional LoRA layers

No gradient flow

Full pre-trained network

Attention & LoRA Targets

- q_proj (Query): Determines what each token is "looking for" in the sequence
- k_proj (Key): Labels each token for matching with queries
- v_proj (Value): Stores the information to be retrieved based on attention scores
- o_proj (Output): Combines the retrieved information into a coherent output

Training Arguments

- Batch sizes control how many examples are processed at once (train=4, eval=2)
- Gradient accumulation (steps=4) allows for effective batch size of 16 (4×4) without requiring more memory
- Learning rate (2.0e-04) with cosine scheduler gradually reduces the update size
- Gradient checkpointing saves memory by recalculating activations instead of storing them
- The relationship between steps and epochs is: Steps = (dataset_size ÷ effective_batch_size) × epochs

These parameters balance training speed, memory usage, and learning effectiveness. Smaller batch sizes with gradient accumulation allow training on limited hardware while maintaining learning quality.

Supervised Fine-Tuning (SFT)

SFT is the process of training a pretrained model on a labeled dataset, where we explicitly show it examples of what the correct outputs should be. It's called "supervised" because we are supervising the model — giving it both inputs and correct answers

RLHF and PPO

01 **RLHF** stands for **Reinforcement Learning from Human Feedback**

02 Why RLHF? Even after supervised fine-tuning (SFT), models: Might still say weird or unsafe things, Don't always follow user preferences. So we train them using human feedback to make them behave better

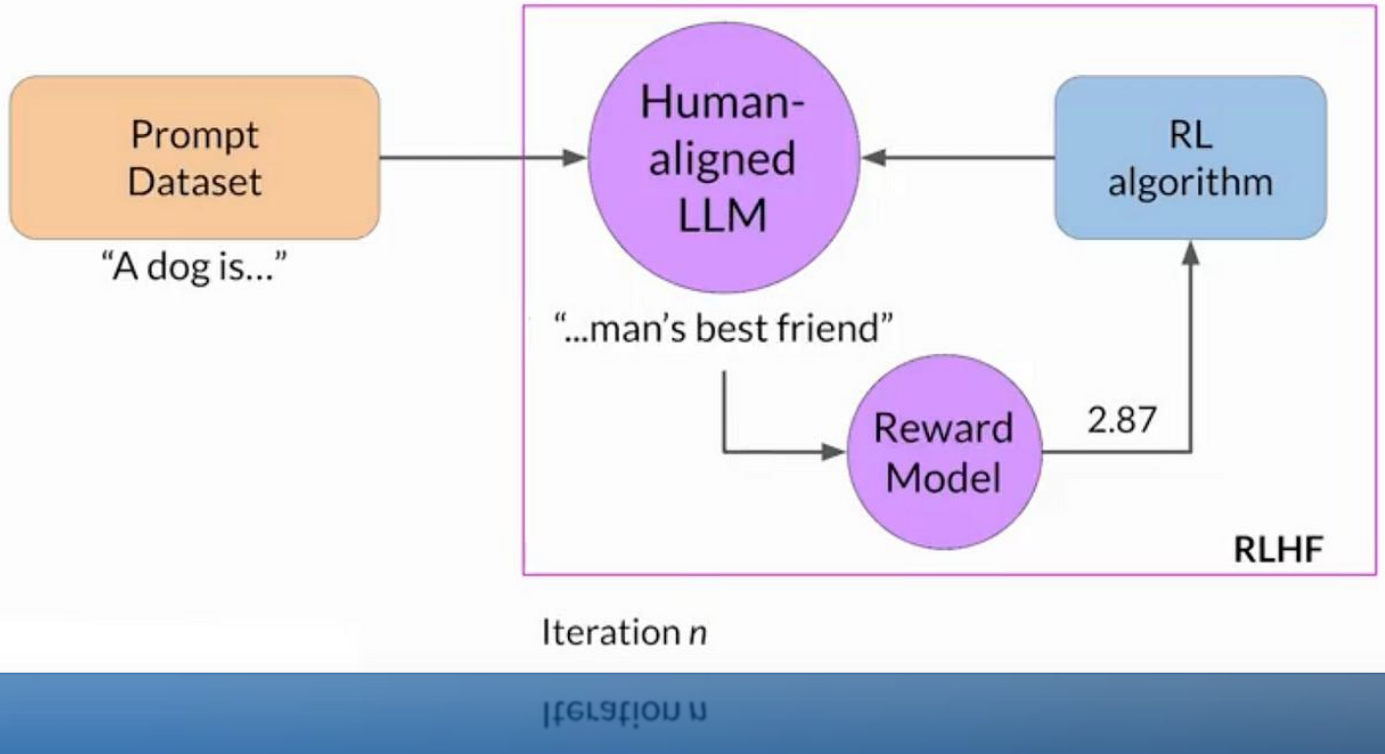
03 **PPO** is Stable (avoids making big changes to the model too quickly) and Efficient (works well with big models like GPT)

04 PPO adjusts the model to make it more likely to generate higher-scoring responses, but not so much that it forgets everything else

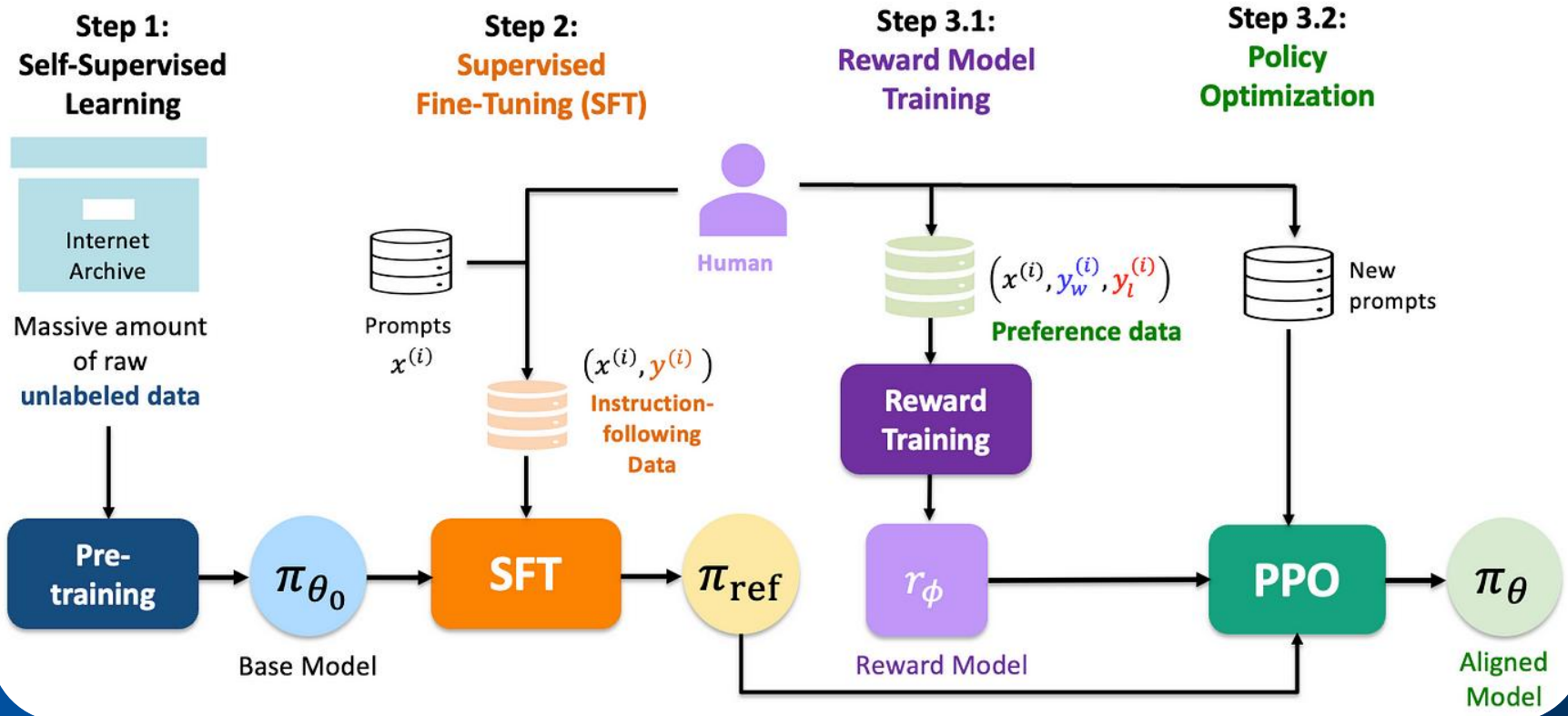
Summary Table

Term	What it Means	Role
RLHF	Reinforcement Learning from Human Feedback	Make models safer, aligned
Reward Model	Learns what good responses look like	Scores responses from the model
PPO	A way to apply reinforcement learning	Tunes model gradually, safely

Use the reward model to fine-tune LLM with RL



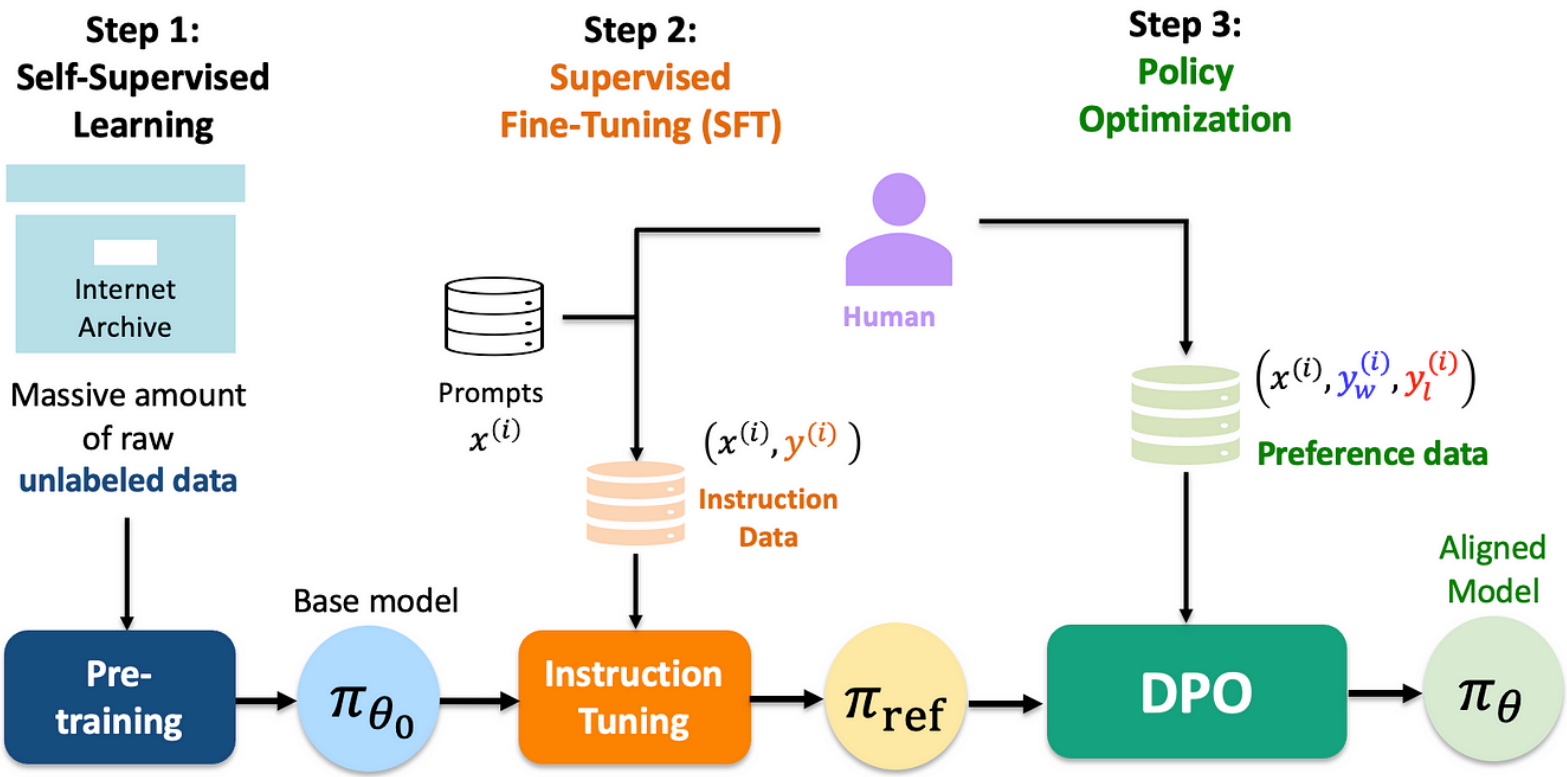
Reinforcement Learning from Human Feedback (RLHF)



Direct Preference Optimization (DPO)

- **DPO is a simpler alternative to RLHF that achieves similar results with less complexity.**
- It eliminates the need for a separate reward model
- It doesn't use reinforcement learning algorithms
- It works directly with preference pairs (preferred vs. rejected responses)
- It's computationally more efficient than RLHF

Direct Preference Optimization (DPO)



Group Relative Policy Optimization (GRPO)

GRPO is a newer alignment technique that offers efficiency improvements:

- It generates multiple responses for each prompt
- It uses the average reward within this group as a baseline
- It eliminates the need for a separate critic model (unlike PPO)
- It's more memory-efficient than traditional RLHF with PPO
- It works with both learned reward models and simple rule-based metrics

GRPO simplifies the RL process by using group statistics instead of a critic model, making it more practical for deployment in resource-constrained environments while maintaining alignment quality.

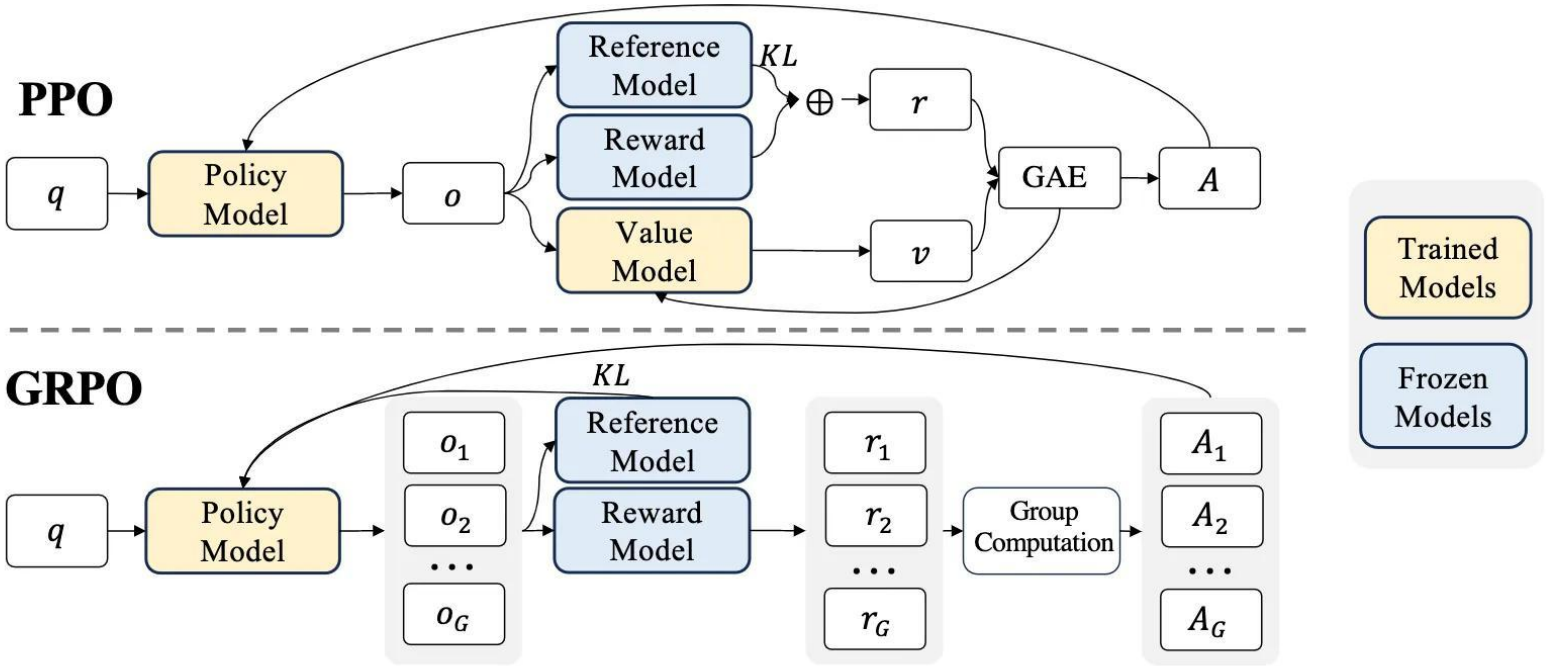


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

Fine-Tuning Method Selection

- SFT is suitable for basic instruction following when high-quality examples are available
- RLHF/PPO is appropriate when human feedback is critical and computational resources are available
- DPO is preferable when preference data exists and simpler implementation is desired
- GRPO works well for efficient alignment with reduced memory requirements

The choice depends on factors like data availability, computational resources, and alignment goals. It's often best to start with simpler approaches (SFT) and scale to more complex methods as needed.

Conclusion & Best Practices

- The fine-tuning lifecycle is iterative, requiring ongoing refinement
- Common pitfalls include overfitting, catastrophic forgetting, and reward hacking
- Efficiency techniques like LoRA and quantization help optimize resource usage
- Monitoring training metrics is essential for detecting issues early
- Thorough testing before deployment ensures quality outputs
- Staying updated on emerging techniques keeps your approach current

These best practices help ensure successful fine-tuning projects that deliver models aligned with your specific needs and values.

```python
from transformers import AutoTokenizer, AutoModelForSequenceClassification

model_path = "arad1367/crypto_sustainability_news_FacebookAI_roberta-large-mnli"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSequenceClassification.from_pretrained(model_path)
```

Fine Tuning Notebook: Crypto News

Today project: Create a Chatbot from scratch and deploy on cloud

Thank You

Do you have any questions?

pejman.ebrahimi@uni.li

UNIVERSITÄT LIECHTENSTEIN