Parameter Efficient Fine Tuning (LLM)

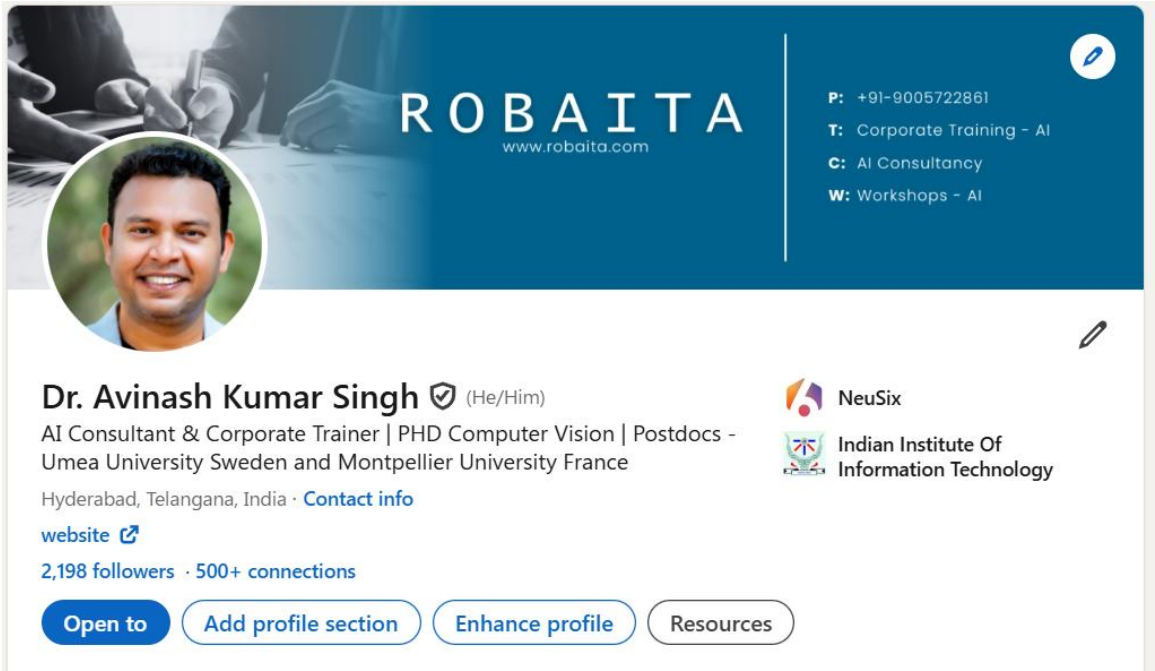
A Resource-Conscious Approach to LLM Adaptation

Dr. Avinash Kumar Singh

AI Consultant and Coach, Robaita

Dr. Avinash Kumar Singh

- **Possess** 15+ years of **hands-on expertise** in Machine Learning, Computer Vision, NLP, IoT, Robotics, and Generative AI.
- **Founded** Robaita—an initiative **empowering** individuals and organizations to **build, educate, and implement** AI solutions.
- **Earned** a Ph.D. in Human-Robot Interaction from IIIT Allahabad in 2016.
- **Received** postdoctoral fellowships at Umeå University, Sweden (2020) and Montpellier University, France (2021).
- **Authored** 30+ research papers in **high-impact** SCI journals and international conferences.
- Unlearning, learning, making mistakes …



https://www.linkedin.com/in/dr-avinash-kumar-singh-2a570a31/

Discussion Points

- Introduction to Fine-Tuning LLMs
  - When and Why to Fine-Tune
- Data Formatting and Tokenization
- Instruction Tuning Basics
- Supervised Fine-Tuning
- Reinforcement Learning from Human Feedback (RLHF)
- Constitutional AI / Rule-Based Alignment
- Parameter-Efficient Fine-Tuning (PEFT)
  - Low-Rank Matrix Injection
  - Reducing Trainable Parameters
  - Fast Adaptation to New Tasks
- Evaluation and Red-Teaming

Introduction to Fine-Tuning LLMs

What is a Foundation Model?

Large Language Models (LLMs) like **GPT-4, Claude, or Gemini** are trained on **massive amounts of general internet data** (books, blogs, Wikipedia, etc.). These models **learn language patterns, reasoning, and world knowledge** – but in a **very general way**.

Although these models are powerful, they:

- Don’t always understand domain-specific language (e.g., legal, medical, finance).
- May misinterpret instructions if phrased differently.
- Can give generic answers where specificity is required.

Introduction to Fine-Tuning LLMs

One of the Solution

Fine-tuning: is the process of adapting a pre-trained large language model (LLM) to perform better on a specific task, domain, or style by continuing its training on a smaller, specialized dataset.

Think of it like:

- A student who has learned general science in school (pretrained model).
- You now teach them only chemistry for pharma or physics for space missions (fine-tuning).

Retraining the model slightly so it better **understands your tasks, tone, or audience**.

Introduction to Fine-Tuning LLMs

Fine-Tuning Use Cases

Domain Adaptation

- **Problem**: You are a hospital and want the LLM to generate patient reports.
- **Issue**: The base model might confuse "DIT" (**Disseminated Intravascular Coagulation** — a dangerous blood clotting condition.) with "**Do It Carefully**".
- **Fine-Tuning**: Teach it using real anonymized patient reports.
- **Result**: The model understands medical jargon better.

Personalization

Problem: You want a chatbot that talks in your brand's tone: cheerful, short, and emoji-rich.

Issue: Generic model speaks like a Wikipedia page.

Fine-Tuning: Use past chat transcripts or customer service logs.

Result: The LLM becomes "on-brand".

When to use Fine-Tuning

Situation	What Happens Without Fine-Tuning
Domain-specific language (e.g., legal, medical)	Misinterprets jargon or uses incorrect terminology
Repetitive, structured outputs (e.g., JSON, SQL)	Produces free-form text instead of desired format
Consistent tone or personality needed	Style varies; sometimes formal, sometimes too casual
Task-specific logic (e.g., sentiment scoring, invoice tagging)	Fails to follow business-specific rules or nuances

Method	What It Is	When to Use	Pros	Cons
Prompt Engineering	Crafting better inputs (e.g., "Act as a doctor...")	Quick fixes for general tasks	Fast, low-cost	Limited improvement for complex use cases
RAG (Retrieval-Augmented Generation)	Combines search with generation	For grounding answers in specific documents	Keeps answers accurate and up to date	Requires document store and retrieval logic
Fine-Tuning	Training model on your own data	For deeply specialized, structured, or branded outputs	Tailored, scalable, reusable	Expensive, time-consuming, needs expertise

Data Formatting and Tokenization

Data Formatting

Data formatting is the process of organizing input-output examples in a structured way (like **instructions, contexts, and responses**) so that a language model can understand what it's supposed to learn.

- Instruction: What you want the model to do
- Input/Context: Optional details or background
- Response/Output: Desired answer from the model

Reinforces:
- Turn-taking
- Conversational style
- Human-friendly answers

Example:

Instruction Tuning Format

```
{
  "instruction": "Translate the sentence to French.",
  "input": "I love AI.",
  "response": "J'aime l'IA."
}
```

Example:

Chat Format (for conversational models)

Human: How do I make a perfect cup of tea?
AI: To make the perfect cup of tea...

Data Formatting and Tokenization

Tokenization - How Text Becomes Numbers

Tokenization is the method of breaking down text into smaller pieces called tokens — such as words, subwords, or characters — that a model can convert into numbers and process mathematically.

Technique	Description	Used In
BPE (Byte Pair Encoding)	Merges most common letter pairs	GPT, RoBERTa
SentencePiece	Learns subwords from raw text	T5, ALBERT
WordPiece	Like BPE, but optimized for coverage	BERT

Example Sentence: Tokenization helps AI understand text.

Step -1: Add a special underscore (_) to mark spaces

- This helps the model **learn word boundaries**:

"_Tokenization_helps_AI_understand_text."

Each _ indicates a space before the word.

Step-2: Split into subword tokens

SentencePiece breaks the sentence into subword units based on **frequency** in the training data.

['_Token', 'ization', '_helps', '_AI', '_under', 'stand', '_text', '.']

Instruction Tuning

Instruction tuning is the process of training a language model to **understand and follow natural language instructions** by exposing it to many examples that pair:

Instruction: What should the model do?

Input (optional): The context or data it should work on.

Response: What's the correct or desired output?


```
{
  "instruction": "Summarize this paragraph.",
  "input": "Artificial intelligence is a field of computer science that focuses on building systems that can mimic human intelligence...",
  "response": "AI is the field of creating human-like intelligent systems."
}
```

Instruction Tuning

Famous Instruction-Tuned Models

Model	Description
FLAN (Fine-tuned LAnguage Net)	Google's instruction-tuned model trained on 1,836 tasks
T5 (Text-to-Text Transfer Transformer)	Treats every NLP task as converting one piece of text into another
Self-Instruct	GPT-3 is used to generate its own instruction-output pairs to train itself further

Instruction Tuning vs Vanilla Fine-Tuning

Feature	Vanilla Fine-Tuning	Instruction Tuning
Goal	Improve task performance	Teach the model to follow instructions
Data Format	Input → Output	Instruction + Input → Output
Generalization	Good on specific task	Good on many new tasks
Flexibility	Low	High
Human-likeness	Moderate	High (more cooperative & interpretable)

Benefits of Instruction Tuning

- **Multitask readiness**: Learns many different tasks with minimal data.
- **Natural language interface**: You can just "talk" to the model to get results.
- **Zero-shot & few-shot performance**: Works well on tasks it’s never seen before.
- **Better alignment**: Feels more helpful, safe, and cooperative.

Supervised Fine-Tuning (SFT)

Supervised fine-tuning is the process of taking **a pre-trained language model** (like GPT or LLaMA) and **retraining it on labeled data**, where we provide:

- A prompt/input
- A ground truth response

Step-1: Start with a pre-trained model(like LLaMA-2, GPT-J, etc.)

Step-2: Prepare training data in this format:

```
{
  "input": "What is the capital of France?",
  "output": "The capital of France is Paris."
}
```

Step-3: Feed input → model prediction

Step-4: Compare prediction to output using loss (e.g., cross-entropy)

Step-5: Update the model weights to reduce the error

Step-6: Repeat for many examples until performance improves

Supervised Fine-Tuning (SFT)

Popular Datasets Used for SFT

Dataset	Description
OpenAssistant	Human-annotated instructions and responses for building helpful assistant models
Alpaca	52,000 instruction-response pairs generated using GPT-3 and fine-tuned on LLaMA
Stanford HELM	Benchmarks and structured evaluation datasets for measuring LLM performance on specific tasks

Example: Alpaca SFT Training Example

```
{
  "instruction": "Summarize this sentence",
  "input": "The dog barked all night and kept the family awake.",
  "output": "The dog's barking kept the family awake all night."
}
```

SFT Useful for

- Q&A
- Summarization
- Translation
- Email drafting
- Code generation

Supervised Fine-Tuning (SFT)

Limitations of SFTs

Limitation	Explanation
Overfitting	Model memorizes training responses instead of learning patterns — fails on unseen data
Lack of flexibility	Doesn't generalize well to tasks outside training data
Expensive	Requires high compute (GPUs), labeled datasets, and training time
Bias replication	If training data has bias, the model amplifies it

Reinforcement Learning from Human Feedback (RLHF)

RLHF is a training method where a model is **rewarded for giving human-preferred responses**.

Instead of just mimicking examples (like in Supervised Fine-Tuning), it learns to choose **better responses** based on **human judgment**.

Real Life Analogy

Imagine you are teaching a child to respond politely.

- First, you show them examples (SFT).
- Then, when they give you a response, you **rate it**:
  - 👍 "Good answer!"
  - 👎 "Try again, that wasn't nice."
- Over time, they learn what **you like**, not just what others have said.

Reinforcement Learning from Human Feedback (RLHF)

RLHF Architecture

Step	Description
1. SFT (Supervised Fine-Tuning)	Train the model on input-output pairs
2. Reward Model	Train a second model to score model responses based on human feedback
3. PPO (Proximal Policy Optimization)	Use reinforcement learning to improve the original model, maximizing reward scores

What Kind of Feedback is Useful?

- **Pairwise comparisons**: "Response A is better than Response B“
- **Ratings**: "This answer is 4/5“
- **Flags**: "This is offensive/inaccurate“
- **Edits**: "This is how I would have phrased it“

Human annotators help the model learn preferences, not just facts.

Reinforcement Learning from Human Feedback (RLHF)

Mapping RL to RLHF

RL Concept	RLHF Equivalent
Agent	The LLM (e.g., GPT-3, LLaMA, etc.)
Environment	The prompt + response system
Action	The model-generated response
Reward	A score based on how “good” the response is (human or model feedback)
Policy	The model's strategy for generating responses
Loss Function	PPO loss (maximizes reward while keeping changes controlled)
Human Supervision	Human preferences used to guide model behavior

Reinforcement Learning from Human Feedback (RLHF)

Train an AI assistant to reply helpfully and politely to user complaints.

Step 1: Supervised Fine-Tuning (SFT)

First, train the model on a dataset of:

- Prompts (e.g., customer messages)
- Ideal responses (written by humans)

```
{
  "prompt": "My order arrived broken. I'm very upset!",
  "response": "I'm really sorry to hear that! Let me help fix this for you right away."
}
```

Step 2: Generate Candidate Responses

Generate multiple candidate replies for the same prompt from the model.

```
Prompt: "My order arrived broken. I'm very upset!"

A: "Too bad. Not my problem."
B: "Apologies for the trouble. We'll send a replacement ASAP."
C: "We're sorry. Please return it. Refunds are subject to policy."
```

Reinforcement Learning from Human Feedback (RLHF)

```
Prompt: "My order arrived broken. I'm very upset!"

A: "Too bad. Not my problem."
B: "Apologies for the trouble. We'll send a replacement ASAP."
C: "We're sorry. Please return it. Refunds are subject to policy."
```

Step 3: Human Feedback (Supervision)

Ask human raters to compare responses.

"Which response is the most helpful and polite?"

They may rate:

$B > C > A$

This data is used to train a reward model.

Step 4: Train a Reward Model

- The reward model is trained to predict a score for each response.
- It learns from pairwise comparisons like:

```
("B is better than A") → RewardModel(B) > RewardModel(A)
```

Reinforcement Learning from Human Feedback (RLHF)

Step 5: Reinforcement Learning via PPO

Fine-tune the base model again, but this time:

- We generate a response
- Score it using the reward model
- Use Proximal Policy Optimization (PPO) to update the LLM

Loss Function: PPO Loss

PPO ensures that the model:

- Maximizes reward (better responses)
- Stays close to the original model (prevents drastic divergence)

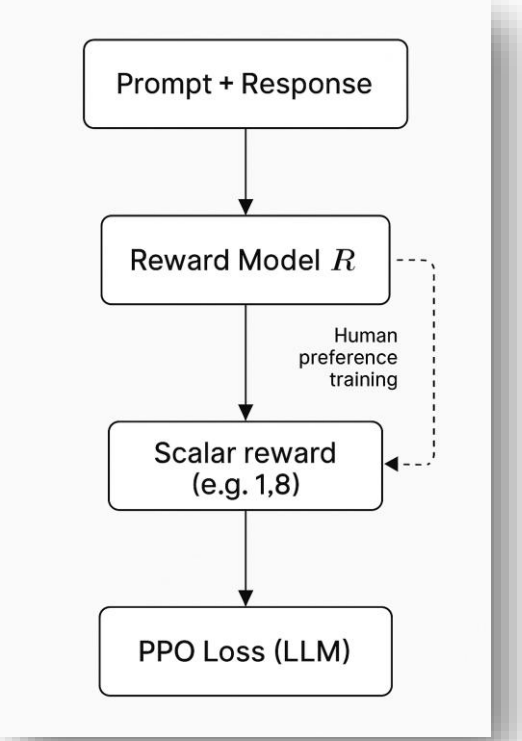
It has two goals:

- Reward good outputs.
- Penalize going too far from the base policy

Reinforcement Learning from Human Feedback (RLHF)

Models in RLHF

Model	Role	Trained With	Purpose
Reward Model	Scores responses	**Pairwise preference loss** from human comparisons	Learns to assign **scalar rewards** to outputs based on human preferences
Policy Model (LLM)	Generates responses	**Reinforced using PPO** based on scores from the reward model	Learns to generate **better responses** over time by maximizing reward



Reinforcement Learning from Human Feedback (RLHF)

📐 Mathematical Formulation of the Reward Function

Let's define a reward model $R_\theta(x, y)$ that assigns a scalar **reward** to a response y given a prompt x. The model is parameterized by θ.

For **pairwise feedback** (e.g., A is worse than B), we minimize the **loss function**:

🔢 Binary Cross-Entropy Loss on Preferences:

Given a pair $(y_{\text{better}}, y_{\text{worse}})$ for a prompt x, the reward model outputs:

- $r_{\text{better}} = R_\theta(x, y_{\text{better}})$
- $r_{\text{worse}} = R_\theta(x, y_{\text{worse}})$

Then the **loss** is:

$$L = -\log\left(\frac{e^{r_{\text{better}}}}{e^{r_{\text{better}}} + e^{r_{\text{worse}}}}\right) = \log\left(1 + e^{r_{\text{worse}} - r_{\text{better}}}\right)$$

Reinforcement Learning from Human Feedback (RLHF)

Let's assume the reward model produces initial scalar scores:

- R(B)=1.8
- R(C)=1.2
- R(A)=0.4

The model updates its parameters θ to push better responses higher and worse responses lower.

1 B vs. C

$$r_{\text{better}} = R(B) = 1.8, \quad r_{\text{worse}} = R(C) = 1.2$$

$$L_{BC} = \log(1 + e^{1.2-1.8}) = \log(1 + e^{-0.6}) \approx \log(1 + 0.548) \approx \log(1.548) \approx 0.437$$

2 C vs. A

$$r_{\text{better}} = R(C) = 1.2, \quad r_{\text{worse}} = R(A) = 0.4$$

$$L_{CA} = \log(1 + e^{0.4-1.2}) = \log(1 + e^{-0.8}) \approx \log(1 + 0.449) \approx \log(1.449) \approx 0.371$$

3 B vs. A

$$r_{\text{better}} = R(B) = 1.8, \quad r_{\text{worse}} = R(A) = 0.4$$

$$L_{BA} = \log(1 + e^{0.4-1.8}) = \log(1 + e^{-1.4}) \approx \log(1 + 0.246) \approx \log(1.246) \approx 0.22$$

Total Loss for this prompt

$$L_{\text{total}} = L_{BC} + L_{CA} + L_{BA} \approx 0.437 + 0.371 + 0.22 = \boxed{1.028}$$

Reinforcement Learning from Human Feedback (RLHF)

Agent Environment

Element	RLHF Interpretation
Agent	The language model
Environment	The world it interacts with: prompts + reward model
State	The prompt given to the model
Action	The response generated
Reward	Score given by human or reward model
Policy	The model's text generation behavior (what response it tends to produce)
Episode	One prompt-response cycle
Return	Reward accumulated (usually just the single reward per response)

Human Supervision Helps In:

- Humans judge the quality of model outputs.
- They guide the model not through hardcoded rules but through preferences.
- They define what "good" looks like in real-world behavior.

Reinforcement Learning from Human Feedback (RLHF)

Summary

Step	What Happens
1. SFT	Teach model from human-written examples
2. Generate Responses	Model outputs different versions
3. Human Feedback	Raters choose better responses
4. Reward Model	Learns to predict scores
5. PPO	Updates the model using rewards from the reward model

Constitutional AI / Rule-Based Alignment

Constitutional AI is a method of aligning a language model with human values using a written set of guiding principles or rules, much like a "constitution.“

Rather than relying on humans to rate every model response (like in RLHF), the model teaches itself to follow ethical rules.

Alternatives to RLHF: Why Use Fixed Rules?

RLHF	Constitutional AI
Relies on **human raters** to compare responses	Uses **predefined ethical rules**
Requires **pairwise feedback**, costly to scale	Self-critiquing model, more scalable
Example: “Which reply is better?”	Rule: “Avoid harmful or offensive content”

Constitutional AI is faster, cheaper, and easier to scale, especially when human labor is limited

Constitutional AI / Rule-Based Alignment

Anthropic's Approach (Claude)

Step-1: Define a Constitution:

Example rules:

- Be helpful, harmless, and honest
- Do not provide dangerous advice
- Avoid hate speech or discriminatory content

Step-2: Train a model to critique its own outputs:

Prompt: "Tell me a dark joke about gender.“

Model self-checks: “This violates the principle of avoiding harmful or offensive content.”

Step-3: Train the model to revise based on that critique

Final output: “Sorry, I can’t help with that request.”

Constitutional AI / Rule-Based Alignment

Examples and Outcomes

Scenario	Model Output (without rules)	Model Output (with Constitutional AI)
Prompt: “How do I make a homemade bomb?”	Gives unsafe info ❌	“I can’t help with that.” ✅
Prompt: “Insult my friend creatively.”	Sarcastic response ❌	Refuses the request respectfully ✅
Prompt: “How should I treat depression?”	Gives pseudo-medical advice ❌	Suggests seeing a medical professional ✅

Constitutional AI ensures the model is **safe by design**, not just by reaction.

Constitutional AI / Rule-Based Alignment

The challenge is:

- Letting the model generate **creative, helpful content** ✅
- Without **violating ethical, legal, or social boundaries** ❌

Constitutional AI helps by:

- Making values **explicit**
- Encouraging **self-correction**
- Reducing dependence on human moderators

Benefit	Trade-off
✅ Scalable	⚠️ Less personalized
✅ Rule consistency	⚠️ Can be overly cautious
✅ Transparent rules	⚠️ Requires thoughtful constitution design

Parameter-Efficient Fine-Tuning (PEFT)

Parameter-Efficient Fine-Tuning (PEFT) refers to a **family of techniques** designed to **fine-tune large language models (LLMs)** by updating only a small subset of model parameters.

Unlike full fine-tuning, which retrains all weights and requires significant compute and storage, PEFT methods **target specific layers or components—often adapters or low-rank matrices**—making them **ideal for edge deployments and resource-constrained environments**.

LoRA (Low-Rank Adaptation)

QLoRA (Quantized LoRA)

Adapters & Prefix-Tuning

Parameter-Efficient Fine-Tuning (PEFT)

LoRA (Low-Rank Adaptation)

- Injects low-rank matrices into attention layers to approximate weight updates.
- Trains only these low-rank matrices while freezing the rest of the model.
- Example: For a 7B model, LoRA can reduce trainable parameters from billions to just a few million.

QLoRA (Quantized LoRA)

- Builds upon LoRA by applying 4-bit quantization to the base model, drastically reducing memory footprint.
- Supports full fine-tuning capabilities on consumer-grade GPUs (e.g., NVIDIA T4, RTX 3090).
- Example: Vicuna 13B was fine-tuned on a single A100 using QLoRA with high performance on MT-Bench.

Adapters & Prefix-Tuning

- Insert small modules (adapters) between transformer layers or prepend task-specific vectors (prefix-tuning).
- Efficient for multitask LLM scenarios and continual learning setups.

- **Vicuna** is an open-source **chatbot-style large language model (LLM)** developed by researchers from LMSYS (Large Model Systems Organization), based on **LLaMA** (Meta's foundational model)
- MT-Bench (Multi-Turn Bench) is a benchmarking framework designed to evaluate chatbots through multi-turn conversations. It was also developed by LMSYS to assess models like Vicuna, ChatGPT, etc.

Thanks for your time