From Language Model to Assistant

The Complete Journey

SFT, RLHF, and Modern AI Systems

Nipun Batra | IIT Gandhinagar

The Journey Complete

Week	Topic	The Question
1-2	Foundations	How do machines learn?
3-4	Supervised Learning	How do we predict?
5	Neural Networks	What makes deep learning special?
6	Computer Vision	How do machines see?
7	Language Models	How do machines understand text?
8	**LM → Assistant**	**How do we make it helpful?**

Last Week's Cliffhanger

We learned:

- LLMs predict the next token
- Transformers enable long-range attention
- Temperature controls creativity

But we left with a puzzle:

A model that predicts text well is NOT the same as a helpful assistant!

The ChatGPT Problem

What We Have	What We Want
Great text completion	Following instructions
"Paris is the capital of..." → "France"	"What's the capital of France?" → "Paris"
Random poem continuation	"Write a poem about..."
Any code that fits	Working, safe code

Today: How do we bridge this gap?

Today's Agenda

1. **The Problem** - Why base models aren't enough
2. **SFT** - Teaching models to follow instructions
3. **RLHF** - Learning from human preferences
4. **DPO** - A simpler alternative
5. **The Full Pipeline** - Pre-training → SFT → Alignment
6. **Image Generation** - Quick overview
7. **AI Ethics & Future** - Responsible AI

Part 1: The Base Model Problem

Why Prediction Isn't Enough

Base Model Behavior

A model trained on web text learns to complete web text:

Prompt: "How do I make a cake?"

Base model might respond:

```
How do I make a cake?

I'm looking for a simple chocolate cake recipe. My mom
used to make one but I lost the recipe card. Any help
would be appreciated!

Posted by CakeLover92 on Reddit, March 2019
```

It's completing forum posts, not answering questions!

More Base Model Problems

Prompt	Base Model Response	What We Want
"Write Python code to..."	`# TODO: implement this`	Working code
"Is this email spam?"	Continues the email	Yes/No
"Summarize this article"	Writes more article	Summary

Base models are trained to PREDICT, not to HELP.

The Training Data Problem

Base model training data: The entire internet

- Wikipedia, Reddit, Stack Overflow
- Books, news articles, papers
- Code repositories, forums, social media

What's missing:

- Explicit instruction-following examples
- Feedback on what makes a "good" response
- Alignment with human values

Part 2: Supervised Fine-Tuning (SFT)

Teaching Models to Follow Instructions

SFT: The Key Idea

Collect examples of good instruction-following:

```
Instruction: "What is the capital of France?"
Response: "The capital of France is Paris."

Instruction: "Write a haiku about rain"
Response: "Gentle drops descend
           Washing away dusty streets
           Earth drinks and is whole"

Instruction: "Translate 'hello' to Spanish"
Response: "Hola"
```

Then fine-tune the base model on these examples!

SFT Training Process

```
BASE MODEL                     SFT DATA
(pre-trained on               (instruction, response)
 internet text)                pairs
     |                            |
     +-------------+--------------+
                   |
                   v
        +--------------------+
        |  FINE-TUNING       |   Same architecture,
        |  (continue         |   new training objective
        |   training)        |
        +--------------------+
                   |
                   v
       INSTRUCTION-TUNED
       MODEL
```

Where Does SFT Data Come From?

Source	Method	Examples
Human annotators	Write ideal responses	Expensive, high quality
Existing datasets	Convert Q&A datasets	FLAN, Natural Instructions
Model distillation	GPT-4 generates for smaller model	Alpaca, Vicuna
User feedback	Real conversations	ChatGPT usage data

Famous SFT Datasets

Dataset	Size	Method
FLAN	1,800+ tasks	Compiled from NLP datasets
InstructGPT	~100K	Human annotators
Alpaca	52K	GPT-3 generated
ShareGPT	90K+	User-shared ChatGPT conversations
Dolly	15K	Databricks employees

SFT Training Example

```python
from transformers import AutoModelForCausalLM, Trainer

# Load base model
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b")

# Prepare SFT data
# Format: "### Instruction:\n{instruction}\n### Response:\n{response}"
train_data = load_dataset("sft_data.json")

# Fine-tune
trainer = Trainer(
    model=model,
    train_dataset=train_data,
    args=TrainingArguments(
        output_dir="./sft_model",
        num_train_epochs=3,
        per_device_train_batch_size=4,
    )
)
trainer.train()
```

SFT Results

Model	Before SFT	After SFT
Follows instructions	20%	85%
Appropriate format	15%	90%
Helpful responses	30%	75%

SFT makes a HUGE difference!

But there's still a problem...

The Limits of SFT

SFT teaches WHAT to say, but not WHICH response is BEST.

Prompt: "Write a story about a dog"

Response A: (boring but correct)
Response B: (creative and engaging)
Response C: (grammatically perfect but dull)

SFT can't distinguish between acceptable responses!

We need a way to learn from **preferences**.

Part 3: RLHF

Learning from Human Preferences

RLHF: The Key Insight

Instead of "this is the right answer"...

Let humans rank responses from best to worst!

```
Prompt: "Explain quantum physics simply"

Response A: [Technical jargon, hard to follow]
Response B: [Clear analogy with everyday objects]
Response C: [Accurate but dry explanation]

Human ranking: B > C > A
```

Train the model to generate responses like B!

RLHF: Three Steps

```
STEP 1: Collect Comparisons
──────────────────────────
Human ranks model responses: "A is better than B"

STEP 2: Train Reward Model
─────────────────────────
Learn to predict human preferences
reward(response) → score

STEP 3: Optimize with RL
───────────────────────
Fine-tune LLM to maximize reward model score
(using PPO or similar RL algorithm)
```

Step 1: Collect Human Preferences

Annotators compare pairs of responses:

Prompt	Response A	Response B	Preference
"What's 2+2?"	"The answer is 4."	"4"	A (more helpful)
"Tell a joke"	[Long, funny joke]	[Short, unfunny]	A
"Write code for..."	[Working code]	[Buggy code]	A

Need thousands of comparisons!

Step 2: Train Reward Model

Input: Prompt + Response

Output: Scalar score (how good is this?)

```python
class RewardModel(nn.Module):
    def __init__(self, base_model):
        self.base = base_model
        self.head = nn.Linear(hidden_size, 1)

    def forward(self, prompt, response):
        hidden = self.base(prompt + response)
        score = self.head(hidden[:, -1])  # Score from last token
        return score
```

Training objective: reward(preferred) > reward(rejected)

Step 3: RL Fine-tuning

Use PPO (Proximal Policy Optimization) to maximize reward:

```python
# Simplified RLHF loop
for prompt in prompts:
    # Generate response from current policy
    response = model.generate(prompt)

    # Get reward
    reward = reward_model(prompt, response)

    # Update model to increase reward
    # (with KL penalty to stay close to SFT model)
    loss = -reward + beta * KL_divergence(model, sft_model)
    loss.backward()
    optimizer.step()
```

Why the KL Penalty?

Problem: Without constraint, model might exploit reward model

```
Example reward hack:
Reward model likes "helpful" responses
→ Model learns to add "I hope this helps!" to everything
→ Gets high reward, but responses get worse
```

Solution: Stay close to the SFT model (KL divergence penalty)

RLHF Results: InstructGPT

OpenAI's InstructGPT paper (2022):

Metric	Base GPT-3	SFT	RLHF
Human preference	22%	33%	**71%**
Truthfulness	34%	47%	**68%**
Less harmful	44%	61%	**84%**

RLHF is what made GPT-3 → ChatGPT!

RLHF Challenges

Challenge	Why It's Hard
Expensive	Need many human annotations
Slow	RL training is unstable
Reward hacking	Model exploits reward model
Alignment tax	Sometimes hurts raw capability

Is there a simpler alternative?

Part 4: DPO

A Simpler Path to Alignment

DPO: Direct Preference Optimization

Skip the reward model entirely!

Key insight: We can derive a loss function directly from preferences

```
Traditional RLHF:
  Preferences → Reward Model → RL Training → Aligned Model

DPO:
  Preferences → Direct Fine-tuning → Aligned Model
```

DPO Loss Function

For each comparison (prompt, preferred response, rejected response):

$$\mathcal{L} = -\log \sigma \left(\beta \log \frac{\pi(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi(y_l|x)}{\pi_{ref}(y_l|x)} \right)$$

In simple terms:

- Increase probability of preferred responses
- Decrease probability of rejected responses
- Stay close to reference model

DPO in Practice

```python
from trl import DPOTrainer

# Prepare preference data
# Format: {"prompt": ..., "chosen": ..., "rejected": ...}
train_data = load_preferences_dataset()

# Simple training!
trainer = DPOTrainer(
    model=sft_model,
    ref_model=sft_model,  # Reference for KL penalty
    train_dataset=train_data,
    beta=0.1,  # Temperature parameter
)
trainer.train()
```

Much simpler than RLHF!

DPO vs RLHF

Aspect	RLHF	DPO
Reward model	Required	Not needed
Training stability	Unstable (RL)	Stable (supervised)
Compute	3 models needed	1 model + reference
Hyperparameters	Many (PPO)	Few (just β)
Performance	Strong	Comparable

DPO is becoming the preferred method!

Part 5: The Complete Pipeline

Pre-training → SFT → Alignment

The Full Journey

```
STAGE 1: Pre-training
────────────────────
Data: Trillions of tokens (web, books, code)
Objective: Predict next token
Compute: Thousands of GPUs, months
Result: "Base model" - knows language, facts
                   ↓
STAGE 2: Supervised Fine-Tuning (SFT)
─────────────────────────────────────
Data: ~100K instruction-response pairs
Objective: Learn to follow instructions
Compute: Hours to days
Result: "Instruction model" - follows commands
                   ↓
STAGE 3: Alignment (RLHF or DPO)
────────────────────────────────
Data: Human preference comparisons
Objective: Be helpful, harmless, honest
Compute: Days
Result: "AI Assistant" - ChatGPT, Claude, etc.
```

Real World Examples

Model	Pre-training	SFT	Alignment
GPT-4	Huge web corpus	OpenAI annotators	RLHF
Claude	Web + books	Anthropic	RLHF + Constitutional AI
Llama 2 Chat	2T tokens	Public datasets	RLHF
Mistral Instruct	Web	Public datasets	DPO
Gemma Instruct	Google data	Instruction data	SFT only

The Alignment Tax

Trade-off: Alignment can slightly reduce raw capability

Task	Base Model	Aligned Model
Trivia questions	82%	80%
Code completion	76%	74%
Math problems	68%	65%
Helpfulness	30%	**90%**
Safety	40%	**95%**

Worth it for real-world use!

Two Paradigms: Discriminative vs Generative

Discriminative Models

$$P(\text{label}|\text{input})$$

- Classification
- Regression
- Everything we learned weeks 1-6

Generative Models

$$P(\text{data}) \text{ or } P(\text{data}|\text{prompt})$$

- Text generation (LLMs)
- Image generation (Diffusion)
- This is where AI is today

The Landscape

Domain	Generative Tool	What It Creates
Text	ChatGPT, Claude	Essays, code, poems
Images	DALL-E, Midjourney	Any image from text
Audio	Suno, ElevenLabs	Music, voices
Video	Sora, Runway	Video clips
3D	DreamFusion	3D models
Code	Copilot, Cursor	Working programs

Part 2: Image Generation

From GANs to Diffusion

A Brief History

Year	Model	Key Innovation
2014	GANs	Generator vs Discriminator game
2020	VQVAE	Discrete image tokens
2021	DALL-E	Text-to-image at scale
2022	Stable Diffusion	Open-source, diffusion models
2023	DALL-E 3, Midjourney v5	Photorealistic quality
2024	Flux, SD3	Even better quality

GANs: The Generator-Discriminator Game

Generator

- Creates fake images
- Tries to fool discriminator
- Gets better at faking

Discriminator

- Tells real from fake
- Tries to catch generator
- Gets better at detecting

Both improve until generated images are indistinguishable from real!

Diffusion Models: The New King

Idea: Learn to **denoise** images

```
Training:
Real image → Add noise → Noisy image
             ↑
    Model learns to reverse this!

Generation:
Pure noise → Denoise → Denoise → ... → Final image
```

Diffusion: Step by Step

Step	Image State	What Happens
0	Pure noise	Start with random pixels
1	Mostly noise	Model removes some noise
2	Shapes emerge	Structure appears
...	...	...
50	Clear image	Final result

Each step removes a little noise!

Text-to-Image: How It Works

```
Input: "A photo of a cat wearing a hat on Mars"
       ↓
┌──────────────────────────────────────┐
│ 1. Text Encoder (CLIP)               │  → Text embedding
│ 2. Diffusion Model (guided by text)  │  → Denoising
│ 3. VAE Decoder                       │  → Final image
└──────────────────────────────────────┘
       ↓
Output: Image of a cat in a hat on Mars!
```

Using Image Generation

```python
from openai import OpenAI

client = OpenAI()

response = client.images.generate(
    model="dall-e-3",
    prompt="A serene Japanese garden with cherry blossoms",
    size="1024x1024",
    quality="standard",
    n=1,
)

image_url = response.data[0].url
```

Prompt Engineering for Images

Bad Prompt	Good Prompt
"cat"	"A fluffy orange tabby cat sleeping on a velvet cushion, soft lighting, photorealistic"
"landscape"	"Misty mountain landscape at sunrise, oil painting style, dramatic clouds, warm golden light"

Key elements: Subject, style, lighting, composition, quality modifiers

Part 3: Multimodal AI

Text + Images + More

What is Multimodal?

Modality = Type of data (text, image, audio, video)

Multimodal = Understanding/generating multiple types

Model	Modalities
GPT-4V	Text input + Image input → Text output
DALL-E	Text input → Image output
Gemini	Text + Image + Audio → Text
GPT-4o	Text + Image + Audio ↔️ Text + Audio

Vision-Language Models

Input: Image + Text question

Output: Text answer

```
response = client.chat.completions.create(
    model="gpt-4-vision-preview",
    messages=[{
        "role": "user",
        "content": [
            {"type": "text", "text": "What's in this image?"},
            {"type": "image_url", "image_url": {"url": image_url}}
        ]
    }]
)
```

Use Cases

Task	Input	Output
Image Captioning	Photo	Description
Visual QA	Photo + Question	Answer
OCR + Understanding	Document image	Extracted info
Code from Screenshot	UI mockup	Working code

Part 4: Using LLM APIs

Building with AI

The OpenAI API Pattern

```python
from openai import OpenAI

client = OpenAI()

response = client.chat.completions.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Explain quantum computing"}
    ],
    temperature=0.7
)

print(response.choices[0].message.content)
```

Message Roles

Role	Purpose	Example
system	Set behavior	"You are a Python tutor"
user	User input	"How do I read a file?"
assistant	Model response	"You can use open()..."

```
messages = [
    {"role": "system", "content": "Be concise."},
    {"role": "user", "content": "What is Python?"},
    {"role": "assistant", "content": "A programming language."},
    {"role": "user", "content": "What's it used for?"}
]
```

Key Parameters

Parameter	Controls	Range
temperature	Randomness	0.0 (deterministic) to 2.0 (random)
max_tokens	Response length	1 to context limit
top_p	Nucleus sampling	0.0 to 1.0
frequency_penalty	Repetition	-2.0 to 2.0

Prompt Engineering Basics

Technique	Example
Be specific	"Write a 3-paragraph summary" not "Summarize"
Give examples	"Format: Name: X, Age: Y"
Role-play	"You are an expert data scientist..."
Step-by-step	"Think through this step by step"

Building Applications

```python
def analyze_sentiment(text):
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": """
                Analyze sentiment of the text.
                Return JSON: {"sentiment": "positive/negative/neutral",
                              "confidence": 0.0-1.0}
            """},
            {"role": "user", "content": text}
        ],
        temperature=0
    )
    return json.loads(response.choices[0].message.content)
```

Part 5: Fine-tuning

Customizing Models

When to Fine-tune?

Scenario	Use...
General task	Prompt engineering
Specific style/format	Fine-tuning
Domain knowledge	RAG (Retrieval)
Custom behavior	Fine-tuning

Fine-tuning Overview

```
1. PREPARE DATA
   - Format: {"messages": [{"role": ..., "content": ...}]}
   - Need 50-1000+ examples

2. UPLOAD DATA
   - Upload to OpenAI/Hugging Face

3. TRAIN
   - Fine-tune on your data
   - Usually takes minutes to hours

4. USE
   - Call your custom model
```

Fine-tuning with OpenAI

```python
# 1. Upload training file
file = client.files.create(
    file=open("training_data.jsonl", "rb"),
    purpose="fine-tune"
)

# 2. Create fine-tuning job
job = client.fine_tuning.jobs.create(
    training_file=file.id,
    model="gpt-3.5-turbo"
)

# 3. Use your model
response = client.chat.completions.create(
    model="ft:gpt-3.5-turbo:org:custom-name:id",
    messages=[...]
)
```

Hugging Face: Open Models

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load model
model_name = "meta-llama/Llama-2-7b"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Generate
inputs = tokenizer("Hello, how are", return_tensors="pt")
outputs = model.generate(**inputs, max_length=50)
print(tokenizer.decode(outputs[0]))
```

RAG: Retrieval-Augmented Generation

Problem: LLMs don't know your private data

Solution: Retrieve relevant documents, add to context

```
1. User asks question
2. Search your documents for relevant chunks
3. Add chunks to prompt
4. LLM answers using retrieved context
```

Part 6: The Future

What's Next?

Current Capabilities

Task	State
Text generation	Excellent
Code generation	Very good
Image generation	Excellent
Video generation	Emerging
Audio generation	Good
Reasoning	Improving rapidly

Emerging Trends

Trend	What It Means
Agents	AI that takes actions, uses tools
Reasoning models	o1/o3 - think before answering
Multimodal	Seamless text/image/audio
Smaller models	Run on phones, edge devices
Open weights	Llama, Mistral, etc.

AI Agents

Traditional LLM: Answer questions

AI Agent: Take actions!

```
# Agent can:
# - Search the web
# - Run code
# - Send emails
# - Book appointments
# - Write and execute programs
```

Reasoning Models (o1/o3)

Standard LLM: Immediate response

Reasoning model: Think step-by-step internally

Model	Math Score	Science Score
GPT-4	52%	64%
o1	83%	78%
o3	91%	87%

Challenges Ahead

Challenge	Why It Matters
Hallucinations	Models make up facts
Bias	Reflects training data biases
Alignment	Ensuring helpful, safe behavior
Cost	Training = millions of dollars
Environment	Massive energy consumption
Jobs	Automation concerns

Responsible AI

Principle	Implementation
Transparency	Disclose AI use
Fairness	Test for bias
Privacy	Don't train on private data
Safety	Content filtering
Accountability	Human oversight

Course Summary

What We Learned

Your AI Journey

Week	You Learned
1-2	ML fundamentals, data, train/test
3	Linear/Logistic Regression, Trees, KNN
4	Cross-validation, Ensembles, Clustering
5	Neural networks, PyTorch
6	CNNs, Object detection, YOLO
7	Embeddings, Attention, Transformers
8	Generative AI, APIs, Future

The Core Ideas

1. **ML = Learning from data** (not explicit programming)
2. **Supervised learning** is most common
   - Classification (categories) vs Regression (numbers)
3. **Neural networks** can learn complex patterns
   - CNNs for images, Transformers for sequences
4. **Attention is all you need**
   - Modern AI is built on transformers
5. **Generative AI** creates new content
   - Text, images, audio, video

The Skills You Built

Skill	Tools
ML basics	sklearn, pandas, numpy
Deep learning	PyTorch
Computer vision	CNNs, YOLO
NLP	Transformers, APIs
Generative AI	OpenAI API, Hugging Face

Where to Go Next

Deepen Understanding

- Fast.ai courses
- Stanford CS229, CS231n
- Coursera/Udacity

Build Projects

- Kaggle competitions
- Personal projects
- Open source contributions

Stay Current

- arXiv papers

Key Resources

Resource	What It Offers
Hugging Face	Pre-trained models, datasets
Papers With Code	Latest research + implementations
Kaggle	Competitions, notebooks, data
Fast.ai	Practical deep learning course
3Blue1Brown	Visual math intuition
Andrej Karpathy	Deep learning from scratch

Key Takeaways

1. **AI is pattern recognition at scale**
2. **Data is everything** — garbage in, garbage out
3. **Start simple** — complex ≠ better
4. **Evaluate properly** — test set is sacred
5. **AI is a tool** — you decide how to use it

Congratulations!

You Now Understand Modern AI

"The best way to predict the future is to create it."

— Alan Kay

Go build something amazing!

Questions?