



# LLM Fine-Tuning

# Course Overview



## Day 1: Foundations

- When to fine-tune vs prompt
- Types of fine-tuning approaches
- Data preparation & formatting
- LoRA and PEFT methods
- Infrastructure & tooling



# Why Fine-Tune an LLM?



## Domain Expertise

Teach the model specialized knowledge in medicine, law, finance, or your industry



# Fine-Tuning vs Alternatives

Approach	Best For	Cost	Complexity
Prompt Engineering	Quick iterations, general tasks	Low (API costs only)	Low
RAG	Knowledge-intensive, up-to-date info	Medium	Medium
Fine-Tuning	Style, format, specialized behavior	Medium-High	High
Pre-Training	Entirely new domains/languages	Very High	Very High

# When to Fine-Tune

## □ Good Candidates

- Consistent output format required
- Specific writing style or tone
- Domain-specific terminology
- Reducing prompt length/cost
- Improving task-specific accuracy
- Teaching new behaviors

## □ Not Ideal For



# Types of Fine-Tuning

Full Fine-Tuning



LoRA



QLoRA



Adapters

# Full Fine-Tuning

## How It Works

Update **all model parameters** during training on your dataset.



# LoRA: Low-Rank Adaptation

## The Key Insight

Weight updates during fine-tuning have **low intrinsic rank**. Instead of updating the full weight matrix, we add small trainable matrices.

**~1%**

Trainable Params

**10-50x**

Memory Savings

# QLoRA: Quantized LoRA

Fine-tune 65B models on a single 48GB GPU



## 4-bit Quantization

Base model stored in 4-bit NormalFloat format, reducing memory by 4x



# Memory Requirements Comparison

Model	Full FT	LoRA (16-bit)	QLoRA (4-bit)
Llama 2 7B	~56 GB	~16 GB	~6 GB
Llama 2 13B	~104 GB	~32 GB	~10 GB
Llama 2 70B	~560 GB	~160 GB	~48 GB
Mistral 7B	~56 GB	~16 GB	~6 GB

# Data Preparation

## Data Quality Matters Most

- **Quality over quantity** - 1000 good examples beat 10000 noisy ones
- Diverse examples covering edge cases
- Consistent formatting and structure
- Representative of production use

## Data Sources



# Training Data Formats

## Instruction Format

```
{ "instruction": "Summarize the text", "input": "Long article text...", "output": "Brief summary..." }
```

## Chat Format

## Completion Format

# The Hugging Face Ecosystem



## Transformers

Model loading, tokenization, inference



# Training Infrastructure Options

Platform	GPUs	Cost	Best For
Local GPU	RTX 3090/4090	Upfront cost	Experimentation, small models
Google Colab Pro	T4, A100	\$10-50/mo	Learning, prototyping
RunPod / Vast.ai	Various	\$0.20-2/hr	Flexible, cost-effective
AWS / GCP / Azure	A100, H100	\$2-30/hr	Production, enterprise
Lambda Labs	A100, H100	\$1-2/hr	ML-focused, great value

# Day 1 Recap



## When to Fine-Tune

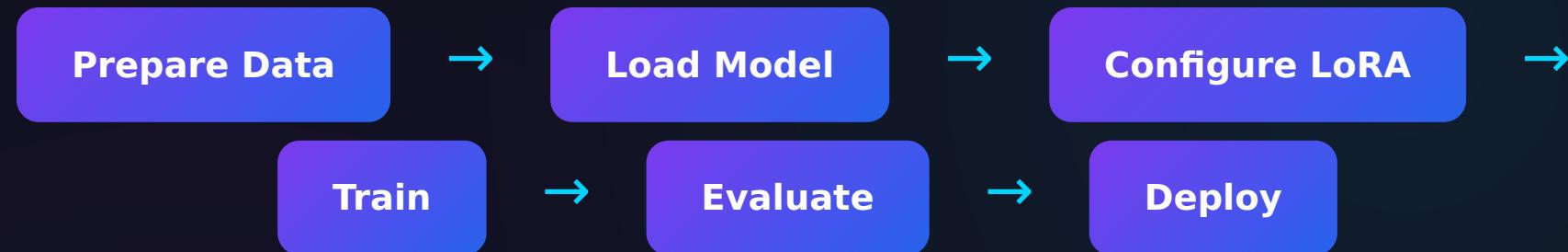
Style, format, behavior - not knowledge





## Day 2: Hands-On Fine-Tuning

# The Fine-Tuning Pipeline





# Complete Fine-Tuning Example

```
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig from peft import LoraConfig, get_peft_model from trl import SFTTrainer, SFTConfig # 1. Load quantized model bnb_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type="nf4") model = AutoModelForCausalLM.from_pretrained("mistralai/Mistral-7B-v0.1", quantization_config=bnb_config) tokenizer = AutoTokenizer.from_pretrained("mistralai/Mistral-7B-v0.1") # 2. Configure LoRA lora_config = LoraConfig(r=16, lora_alpha=32, target_modules=["q_proj", "v_proj", "k_proj", "o_proj"]) model = get_peft_model(model, lora_config) # 3. Train trainer = SFTTrainer(model=model, train_dataset=dataset, args=SFTConfig(output_dir="./output", num_train_epochs=3)) trainer.train()
```

# Key Hyperparameters

## LoRA Parameters

Parameter	Typical Values	Effect
r (rank)	8, 16, 32, 64	Capacity of adaptation
lora_alpha	16, 32, 64	Scaling factor
lora_dropout	0.0, 0.05, 0.1	Regularization
target_modules	q,k,v,o proj	Which layers to adapt

## Training Parameters



# Evaluation Metrics



## Training Metrics

- **Loss:** Should decrease
- **Perplexity:** Lower is better
- **Gradient norm:** Stability indicator



# Common Issues & Solutions

## Overfitting

**Signs:** Training loss drops, eval loss increases

**Solutions:** More data, fewer epochs, higher dropout, lower r



# Merging & Deployment

## Merge LoRA Weights

```
# Merge adapter into base model from peft import PeftModel
base_model = AutoModelForCausalLM.from_pretrained("mistralai/Mistral-7B-v0.1")
model = PeftModel.from_pretrained(base_model, "./lora-adapter") # Merge and save merged =
model.merge_and_unload()
merged.save_pretrained("./merged-model")
```

## Deployment Options

# Quantization for Inference

Format	Bits	Quality Loss	Speed	Use Case
FP16/BF16	16	None	1x	Default inference
INT8	8	Minimal	1.5-2x	Production serving
GPTQ/AWQ	4	Small	2-3x	Memory-constrained
GGUF (Q4)	4	Small	Varies	CPU / Local

# Cost Optimization Strategies



## Training Costs

- Use spot/preemptible instances
- Start with smaller models
- Use gradient accumulation
- Early stopping



# Production Best Practices

## Do

- Version control your data and configs
- Log all experiments with W&B/MLflow
- Start with proven base models
- Evaluate on multiple metrics
- Test in staging before production
- Monitor model performance over time
- Keep rollback capability

## Don't



# Beyond SFT: RLHF & DPO



## RLHF (Reinforcement Learning from Human Feedback)

Train a reward model from human preferences, then use PPO to optimize the LLM

- More complex to implement
- Requires reward model training
- Can achieve strong alignment



# Safety Considerations



## Risks

- Harmful content generation
- Bias amplification
- Privacy leaks from training data
- Jailbreak vulnerabilities



# Tools & Resources

## Libraries

- Hugging Face PEFT
- TRL
- Unsloth
- Axolotl



# Hands-On Lab

## Fine-tune Mistral-7B with QLoRA

### What You'll Build

- Load a 7B model with 4-bit quantization
- Prepare custom instruction dataset
- Configure LoRA adapters
- Train with SFTTrainer
- Evaluate results
- Export for deployment

### Requirements



# Course Summary



## When

Fine-tune for behavior, style, format - not factual knowledge





# Questions?