Fine-Tuning LLMs on Consumer Hardware: LoRA and QLoRA Guide

February 3, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Yes, you can fine-tune LLMs on consumer GPUs. QLoRA lets you train a 7B model on ~6-10 GB VRAM — an RTX 3060 12GB works. The key insight: you don't need thousands of examples. The LIMA study showed 1,000 carefully curated samples can match GPT-quality results. For most tasks, 200-500 high-quality examples are enough. Use Unsloth for 2-5x faster training with 30-70% less memory. The realistic setup: QLoRA + Unsloth + 500 good examples + a 24GB GPU = a custom model in a few hours that outperforms general models on your specific task.

📚 **More on this topic:** Qwen Models Guide · Llama 3 Guide · What Can You Run on 24GB VRAM · Used RTX 3090 Guide · Planning Tool

Fine-tuning used to require datacenter hardware. A 7B model needs ~60 GB VRAM for full fine-tuning — that's multiple A100s. Consumer GPUs couldn't touch it.

LoRA changed that in 2023. QLoRA made it accessible in 2024. Now you can fine-tune a 7B model on an RTX 3060 12GB in a few hours. The barrier isn't hardware anymore — it's knowing what actually works.

This guide covers the practical path: what fine-tuning does, when it's worth it, and how to actually do it on hardware you already own.

What Fine-Tuning Actually Does

Fine-tuning takes a pre-trained model and adjusts its weights using your data. The model learns your patterns — your writing style, your domain terminology, your specific task format.

Fine-tuning is NOT:

- Training a model from scratch (that requires millions of dollars)
- Adding new knowledge (use RAG for that)
- Making a small model as smart as a large one

Fine-tuning IS:

- Teaching a model to follow a specific format
- Adapting behavior to your domain
- Improving performance on narrow, well-defined tasks
- Making a model sound like you

When Fine-Tuning Makes Sense

Use Case	Fine-Tune?	Why
Match your writing style	Yes	Style is learnable from examples
Follow a specific output format	Yes	Consistent structure benefits from training
Domain-specific terminology	Maybe	Try RAG first, fine-tune if insufficient
Teach new factual knowledge	No	Use RAG instead
General improvement	No	Just use a better base model
Single task with clear patterns	Yes	Sweet spot for fine-tuning

The honest assessment: most people who think they need fine-tuning actually need better prompts or RAG. Fine-tuning is for when you've tried everything else and need behavior that can't be prompted.

LoRA Explained

Low-Rank Adaptation (LoRA) is why consumer fine-tuning is possible.

The Problem

Full fine-tuning updates every weight in the model. A 7B model has 7 billion parameters. At 16-bit precision, that's ~14 GB just for the weights — plus gradients, optimizer states, and activations. Total: ~60 GB VRAM.

The Solution

LoRA doesn't update the original weights. Instead, it trains small "adapter" matrices that modify the model's behavior. These adapters have millions of parameters instead of billions — 10-100x smaller.

The math: Instead of updating a weight matrix W directly, LoRA learns two small matrices A and B where the update is A × B. If W is 4096×4096 (16M params) and the "rank" is 8, then A is 4096×8 and B is 8×4096 — only 65K parameters total.

Quality Trade-off

LoRA typically recovers **90-95% of full fine-tuning quality**. For most practical applications, this is indistinguishable. The remaining 5-10% only matters for pushing state-of-the-art benchmarks.

QLoRA: The Consumer Hardware Breakthrough

QLoRA combines LoRA with quantization. The base model runs in 4-bit precision while training the LoRA adapters in 16-bit.

Memory Savings

Method	7B Model VRAM	13B Model VRAM
Full fine-tuning	~60 GB	~120 GB
LoRA (16-bit)	~16-20 GB	~32-40 GB
QLoRA (4-bit)	**~6-10 GB**	**~10-16 GB**

QLoRA reduces memory by **75-80%** compared to standard LoRA. A 7B model that needed a multi-GPU setup now fits on an RTX 3060.

Quality Trade-off

QLoRA typically achieves **80-90% of full fine-tuning quality**. The 4-bit quantization introduces some approximation error, but it's small enough that most tasks don't notice.

For practical purposes: if your task is clear and well-defined (format following, style matching, domain adaptation), QLoRA quality is more than sufficient.

Hardware Requirements

Realistic VRAM Needs

Model Size	QLoRA VRAM	Suitable GPUs
3B	~4 GB	RTX 3060 8GB, any 8GB+ card
7B	~6-10 GB	RTX 3060 12GB, RTX 3070
13B	~10-16 GB	RTX 3090, RTX 4090
32B	~24-32 GB	RTX 4090 (tight), dual GPU
70B	~46-48 GB	A100 80GB, multi-GPU

What Each GPU Tier Can Train

GPU	VRAM	Realistic Training
RTX 3060 12GB	12 GB	7B models with QLoRA
RTX 3070/3080	8-10 GB	7B with small batch size
RTX 3090	24 GB	13B with QLoRA, 7B comfortably
RTX 4090	24 GB	13B with QLoRA, ~1.5-2x faster than 3090
2x RTX 3090	48 GB	32B models, 70B with heavy quantization

Beyond VRAM

- **CPU RAM:** At least 32 GB system RAM for 7B, 64 GB for 13B+
- **Storage:** Fast SSD helps with data loading — NVMe preferred
- **Batch size:** Lower VRAM means smaller batches, longer training

---

The Fine-Tuning Stack

Unsloth (Recommended)

Unsloth is the current best option for consumer fine-tuning:

- **2-5x faster** training than standard Hugging Face

- **30-70% less VRAM** with no accuracy loss
- Free Colab notebooks for fine-tuning up to 14B models
- Supports Llama, Qwen, Mistral, Gemma, and more

Unsloth achieves this through optimized Triton kernels that fuse operations and reduce memory overhead. It's not magic — it's better engineering.

Other Options

Tool	Best For	Notes
Unsloth	Most users	Fastest, easiest
Axolotl	Complex setups	More configuration options
Hugging Face PEFT	Maximum flexibility	Standard but slower
LLaMA-Factory	GUI-based training	Good for beginners

For your first fine-tune, use Unsloth. Graduate to Axolotl if you need features Unsloth doesn't support.

Dataset Preparation

Quality Over Quantity

The single most important insight: **you need fewer examples than you think**.

Study	Dataset Size	Result
LIMA (Meta)	1,000 samples	Matched GPT-quality on evaluations
Stanford Alpaca	52,000 samples	Strong instruction-following
Practical minimum	100-200 samples	Viable for simple tasks

The LIMA paper showed that 1,000 carefully curated examples beat 50,000 mediocre ones. Quality is everything.

How Much Data You Actually Need

Task	Recommended Size	Notes
Style adaptation	100-200 samples	Examples of your writing
Format following	200-500 samples	Input/output pairs
Domain adaptation	500-1,000 samples	Domain-specific Q&A
Complex instruction-following	1,000-5,000 samples	More for edge cases

Unsloth's recommendation:

- 1,000+ rows → Train on base model
- 300-1,000 rows → Either base or instruct model
- <300 rows → Use instruct model (it already knows how to follow instructions)

Dataset Formats

Alpaca format (single-turn, most common):

```
{
  "instruction": "Summarize the following text in one sentence.",
  "input": "The quick brown fox jumps over the lazy dog...",
  "output": "A fox demonstrates agility by leaping over a resting dog."
}
```

ShareGPT format (multi-turn conversations):

```
{
  "conversations": [
    {"from": "human", "value": "What is the capital of France?"},
    {"from": "gpt", "value": "The capital of France is Paris."},
    {"from": "human", "value": "What's its population?"},
    {"from": "gpt", "value": "Paris has approximately 2.1 million residents..."}
  ]
}
```

Use Alpaca for single-turn tasks (most fine-tuning). Use ShareGPT if you're training a conversational assistant.

Creating Quality Data

1. **Start with real examples** — Use actual inputs and outputs from your use case
2. **Review manually** — Every example should be correct and representative
3. **Include edge cases** — Don't just train on easy examples
4. **Diversify inputs** — Vary phrasing, length, and complexity
5. **Keep outputs consistent** — Same task should produce similar output style

Red flag: If you're generating training data with another LLM, you're probably just teaching your model to imitate that LLM. Use real data from your actual use case.

Step-by-Step Tutorial: Unsloth + QLoRA

This tutorial fine-tunes Llama 3.1 8B on a custom dataset using QLoRA. Works on RTX 3060 12GB or better.

1. Install Dependencies

```
pip install unsloth
pip install --upgrade transformers datasets accelerate peft bitsandbytes
```

2. Load Model with 4-bit Quantization

```
from unsloth import FastLanguageModel
import torch

# Load model in 4-bit
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/llama-3.1-8b-bnb-4bit",
    max_seq_length=2048,
    dtype=None,  # Auto-detect
    load_in_4bit=True,
)

# Add LoRA adapters
model = FastLanguageModel.get_peft_model(
    model,
    r=16,  # LoRA rank
```

```
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
                    "gate_proj", "up_proj", "down_proj"],
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    use_gradient_checkpointing="unsloth",
)
```

3. Prepare Dataset

```
from datasets import load_dataset

# Load your dataset (Alpaca format)
dataset = load_dataset("json", data_files="your_data.json", split="train")

# Format for training
alpaca_prompt = """### Instruction:
{instruction}

### Input:
{input}

### Response:
{output}"""

def formatting_func(examples):
    texts = []
    for instruction, input_text, output in zip(
        examples["instruction"],
        examples["input"],
        examples["output"]
    ):
        text = alpaca_prompt.format(
            instruction=instruction,
            input=input_text if input_text else "",
            output=output
        )
        texts.append(text)
    return {"text": texts}

dataset = dataset.map(formatting_func, batched=True)
```

4. Configure Training

```
from trl import SFTTrainer
from transformers import TrainingArguments

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=2048,
    args=TrainingArguments(
        per_device_train_batch_size=2,
        gradient_accumulation_steps=4,
        warmup_steps=10,
        max_steps=100,  # Adjust based on dataset size
        learning_rate=2e-4,
        fp16=not torch.cuda.is_bf16_supported(),
        bf16=torch.cuda.is_bf16_supported(),
        logging_steps=10,
        output_dir="outputs",
        optim="adamw_8bit",
    ),
)
```

5. Train

```
trainer.train()
```

On an RTX 3090 with 500 examples, this takes ~30-60 minutes. On an RTX 3060 12GB, expect 1-2 hours.

6. Save the LoRA Adapter

```
# Save just the LoRA weights (small, ~50-200 MB)
model.save_pretrained("my-lora-adapter")
tokenizer.save_pretrained("my-lora-adapter")
```

Converting and Using Your Fine-Tune

Merge LoRA into Base Model

To run your fine-tune in Ollama or llama.cpp, you need to merge the LoRA adapter into the base model and convert to GGUF.

```
# Merge LoRA into base model
model.save_pretrained_merged(
    "merged-model",
    tokenizer,
    save_method="merged_16bit",
)
```

Convert to GGUF

```
# Clone llama.cpp if you haven't
git clone https://github.com/ggerganov/llama.cpp
cd llama.cpp

# Convert to GGUF
python convert_hf_to_gguf.py ../merged-model --outfile my-model.gguf

# Quantize (optional, for smaller size)
./llama-quantize my-model.gguf my-model-q4_k_m.gguf q4_k_m
```

Run in Ollama

Create a Modelfile:

```
FROM ./my-model-q4_k_m.gguf

PARAMETER temperature 0.7
PARAMETER num_ctx 2048

SYSTEM "You are a helpful assistant fine-tuned for [your task]."
```

```
ollama create my-fine-tune -f Modelfile
ollama run my-fine-tune
```

Common Mistakes

Overfitting

Symptom: Model memorizes training examples verbatim, fails on new inputs.

Fix:

- Use more diverse training data
- Reduce training steps
- Increase LoRA dropout (0.05-0.1)
- Lower learning rate

Catastrophic Forgetting

Symptom: Model loses general capabilities after fine-tuning.

Fix:

- Train for fewer steps
- Use lower learning rate (1e-5 instead of 2e-4)
- Include some general-purpose examples in your dataset
- Start from an instruct model, not base

Learning Rate Too High

Symptom: Training loss spikes or doesn't decrease.

Fix:

- Start with 2e-4 for QLoRA
- Reduce to 1e-4 or 5e-5 if unstable
- Use warmup steps (5-10% of total steps)

Too Few Examples

Symptom: Model doesn't learn the pattern you want.

Fix:

- Add more examples (aim for 200+ minimum)
- Ensure examples are diverse
- Check that examples are actually correct

Wrong Base Model

Symptom: Fine-tune underperforms expectations.

Fix:

- For format/style tasks: use instruct models
- For complex reasoning: use larger base models
- For coding: start from a code-specialized model

When NOT to Fine-Tune

Fine-tuning isn't always the answer. Consider alternatives first:

Prompt Engineering Might Be Enough

If you can describe what you want in a prompt, try that before fine-tuning. Modern models are remarkably good at following detailed instructions.

Example: Instead of fine-tuning for JSON output, try:

```
Respond ONLY with valid JSON in this exact format:
{"field1": "value", "field2": "value"}
No explanations, no markdown, just the JSON object.
```

RAG for New Knowledge

Fine-tuning doesn't reliably add new factual knowledge. If you need the model to know about your company's products, internal documents, or recent information, RAG is the right approach.

A Better Base Model Might Suffice

Before fine-tuning Llama 3.1 8B for coding, try Qwen 2.5 Coder 32B. The better base model might already do what you need.

The 80/20 Rule

Fine-tuning typically improves performance by 10-30% on specific tasks. If you need 2x improvement, fine-tuning alone won't get you there. Consider:

- Better base model + fine-tuning
- RAG + fine-tuning
- Multiple specialized models

LoRA Hyperparameters

Rank (r)

Rank	Use Case	Notes
r=4-8	Simple tasks, style adaptation	Maximum efficiency
r=16-32	Most tasks	**Recommended default**
r=64	Complex tasks	More capacity
r=128+	Rarely needed	Diminishing returns

Research shows **little practical difference between r=8 and r=256** for most tasks. Start with r=16.

Alpha

Common rule: **alpha = 2× rank**

- r=8 → alpha=16
- r=16 → alpha=32
- r=32 → alpha=64

This comes from Microsoft's original LoRA paper and works well in practice.

Target Modules

For most transformer models, target these layers:

- `q_proj`, `k_proj`, `v_proj`, `o_proj` (attention)
- `gate_proj`, `up_proj`, `down_proj` (MLP)

Training all of these gives best results. Training only attention layers uses less memory but may reduce quality.

Bottom Line

Fine-tuning is more accessible than ever. An RTX 3060 12GB can train a 7B model with QLoRA. An RTX 3090 handles 13B models comfortably. You don't need thousands of examples — 500 high-quality samples often suffice.

The realistic path:

1. Start with a good instruct model (Qwen 3, Llama 3)
2. Collect 200-500 real examples from your use case
3. Fine-tune with Unsloth + QLoRA
4. Convert to GGUF and run in Ollama

Before you start:

- Try prompt engineering first — it might be enough
- Consider RAG for knowledge tasks
- Make sure your examples are high quality
- Start small (fewer steps, lower rank) and iterate

Fine-tuning is a tool, not a solution. It works well for teaching models specific formats, styles, and behaviors. It doesn't work for adding knowledge or making small models smarter. Use it when it fits.

```
# Get started with Unsloth
pip install unsloth
```

```
# Or use their free Colab notebooks:
# https://github.com/unslothai/unsloth
```

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/fine-tuning-local-lora-qlora/

Free guides for running AI locally