LAB_TASK (Agentic AI)

Fine-tuning Llama-3.2-3B-Instruct with DeepSeek-R1 Thinking Capabilities

Overview

This tutorial demonstrates how to fine-tune the **Llama-3.2-3B-Instruct** model using Unsloth, a high-performance library for efficient model training. The goal is to empower the Llama model with DeepSeek-R1 like reasoning and thinking capabilities using the ServiceNow-AI/R1-Distill-SFT dataset.

Key Features

- Fast and memory-efficient training using Unsloth
- 4-bit quantization for reduced memory footprint
- LoRA (Low-Rank Adaptation) for parameter-efficient fine-tuning
- GGUF format export for Ollama deployment
- Support for multi-GPU training

Prerequisites

Hardware Requirements

- GPU with at least 16GB VRAM (T4 or better recommended)
- CUDA-enabled environment

Software Requirements

- Python 3.10 or higher
- PyTorch 2.4.0 or higher with CUDA support
- Hugging Face account for model access
- Ollama (optional, for deployment)

Installation

The notebook installs all required dependencies automatically. The main packages include:

Package	Description
unsloth	Core library for efficient model training
xformers	Memory-efficient attention mechanisms
bitsandbytes	4-bit and 8-bit quantization support
transformers	Hugging Face transformers library (≥4.46.1)
trl	Transformer Reinforcement Learning (≥0.7.9)
peft	Parameter-Efficient Fine-Tuning (≥0.7.1)

Training Workflow

The fine-tuning process follows these steps:

1. **Environment Setup:** Install all required dependencies and libraries
2. **Model Loading:** Load the Llama-3.2-3B-Instruct model with 4-bit quantization
3. **Dataset Preparation:** Load and format the ServiceNow-AI/R1-Distill-SFT dataset
4. **LoRA Configuration:** Configure Low-Rank Adaptation parameters for efficient training
5. **Training:** Fine-tune the model using SFTTrainer with configured hyperparameters
6. **Export:** Convert the fine-tuned model to GGUF format for Ollama deployment
7. **Deployment:** Create and test the model in Ollama

Model Configuration

Base Model

Model: unsloth/Llama-3.2-3B-Instruct **Quantization:** 4-bit (load_in_4bit=True) **Max Sequence Length:** Automatically detected (typically 2048 or 4096 tokens)

LoRA Parameters

The notebook uses LoRA (Low-Rank Adaptation) to efficiently fine-tune the model by training only a small subset of parameters:

Parameter	Value / Description
r (rank)	Typically 8-16 (determines adaptation capacity)
target_modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
lora_alpha	Typically 16 (scaling factor for LoRA weights)
lora_dropout	0 (no dropout, recommended for stability)

Training Configuration

The training process uses the SFTTrainer (Supervised Fine-Tuning Trainer) with the following key hyperparameters:

- **Batch Size:** Configured per GPU (typically 2-4 depending on VRAM)
- **Gradient Accumulation:** Used to simulate larger batch sizes
- **Learning Rate:** 2e-4 with warmup steps and cosine decay
- **Epochs:** Typically 1-3 epochs depending on dataset size
- **Optimizer:** AdamW 8-bit for memory efficiency
- **Mixed Precision:** FP16 or BF16 depending on hardware support

Dataset

The tutorial uses the **ServiceNow-AI/R1-Distill-SFT** dataset, which contains high-quality instruction-response pairs that include reasoning traces similar to DeepSeek-R1. The dataset teaches the model to:

- Show step-by-step reasoning before providing answers
- Break down complex problems into manageable steps
- Provide detailed explanations of thought processes
- Generate more accurate and reliable responses

Export and Deployment

GGUF Format Conversion

After training, the model is converted to GGUF (GPT-Generated Unified Format) for efficient inference with Ollama. The notebook exports the model with various quantization levels:

- **Q4_K_M:** 4-bit quantization, medium quality (recommended)
- **Q5_K_M:** 5-bit quantization, higher quality
- **Q8_0:** 8-bit quantization, highest quality but larger file size
- **F16:** 16-bit floating point, no quantization (largest file)

Ollama Integration

The final step creates a custom Ollama model using a Modelfile. This allows you to run the fine-tuned model locally with a simple command:

```
ollama create unsloth_model -f ./Modelfile
```

Once created, you can interact with your model using:

```
ollama run unsloth_model
```

Performance Optimization Tips

- **Use Gradient Checkpointing:** Reduces memory usage at the cost of slightly slower training
- **Adjust Batch Size:** Start small and increase until you hit memory limits
- **Use Flash Attention 2:** Unsloth automatically enables this for supported GPUs
- **Monitor GPU Usage:** Use nvidia-smi to track VRAM and utilization
- **Save Checkpoints:** Enable checkpoint saving to resume training if interrupted

Common Issues and Solutions

Issue	Solution
Out of Memory (OOM)	Reduce batch size, enable gradient checkpointing, or use smaller LoRA rank
Slow Training Speed	Verify Flash Attention 2 is enabled, increase batch size, or use multiple GPUs

Model Not Loading	Check Hugging Face token permissions and ensure model access is granted
GGUF Export Fails	Ensure sufficient disk space and verify gguf Python package is installed
Ollama Creation Error	Verify Modelfile syntax and ensure GGUF file path is correct

Expected Results

After fine-tuning, the model should demonstrate:

- **Enhanced Reasoning:** Explicit step-by-step thinking before providing answers
- **Improved Accuracy:** Better performance on complex reasoning tasks
- **Detailed Explanations:** Clear articulation of thought processes
- **DeepSeek-R1 Style:** Similar reasoning patterns to DeepSeek-R1 models

Additional Resources

- **Unsloth Documentation:** https://github.com/unslothai/unsloth
- **Dataset:** https://huggingface.co/datasets/ServiceNow-AI/R1-Distill-SFT
- **Llama Models:** https://huggingface.co/meta-llama
- **Ollama Documentation:** https://ollama.ai/
- **LoRA Paper:** https://arxiv.org/abs/2106.09685

Conclusion

This tutorial provides a complete workflow for fine-tuning Llama-3.2-3B-Instruct with Unsloth, enabling enhanced reasoning capabilities similar to DeepSeek-R1. The resulting model can be deployed locally using Ollama for efficient inference. The combination of 4-bit quantization, LoRA adaptation, and optimized training makes this approach both memory-efficient and effective for creating powerful reasoning models on consumer hardware.