Instruction Fine-Tuning of LLaMA-2-7B with LoRA on Dolly-15K

Nauman (AUTHOR)[1]

[1]Department in Faculty of Computing, National University of Singapore, Singapore
nauman843@hotmail.com

Abstract – We fine-tune meta-llama/Llama-2-7b-hf using Parameter-Efficient Fine-Tuning (LoRA) on the databricks/databricks-dolly-15k dataset to adapt the base model for instruction following. We report training/validation loss curves and evaluate with AlpacaEval 2 (GPT-4 as judge over the AlpacaEval dataset) and MT-Bench (FastChat).

1 Background and Motivation

Large language models (LLMs) such as LLaMA-2-7B are powerful but require alignment to follow user instructions reliably. Instruction fine-tuning adapts a pretrained model from generic next-token prediction to helpful, safe, and coherent task completion. Parameter-Efficient Fine-Tuning (PEFT) with LoRA enables this alignment on modest hardware by updating a small set of low-rank adapter weights while keeping base weights frozen.

2 Relevance and related works

LoRA/PEFT has become a standard approach for efficient adaptation of LLMs, enabling fine-tuning on consumer-grade GPUs. Dolly-15K provides a permissive instruction-following corpus. AlpacaEval 2 and MT-Bench (via FastChat) are widely used evaluation protocols; both rely on LLM-as-a-judge (e.g., GPT-4) for scalable scoring.

3 Methods and Approach

Dataset. We use databricks/databricks-dolly-15k. Each record is formatted as:

```
### Instruction:
{instruction}

### Context:
{context}

### Response:
{response}
```

We remove empty responses and split into train/validation/test (80/10/10).

Model & Training. Base: meta-llama/Llama-2-7b-hf. We apply LoRA with PEFT, enabling 4-bit quantization if needed for memory. Key settings include learning rate, epochs, global batch size, max length, gradient checkpointing, and saving LoRA adapters. Training/validation loss is tracked per epoch. Hardware details are recorded (GPU, VRAM, runtime).

Inference. For testing/inference, LoRA adapters are merged into the base model to create a standalone checkpoint for evaluation and answer generation.

Evaluation.

- *AlpacaEval 2*: It is a dataset of instruction-following prompts. We generate model responses and compute win rate using GPT-4 as an automated judge against reference responses [1].
- *MT-Bench (FastChat)*: We generate model answers for 80 multi-turn conversations for both baseline and fine-tuned models. GPT-4 single-answer grading is run locally on our JSONL outputs (subset if budget-limited), producing overall and per-category scores. We document that the training prompt style may induce formatting artifacts (e.g., appending `### Response:`), which negatively impacts MT-Bench.

Reproducibility. All notebooks include explanatory markdown; Colab-specific metadata (e.g., `metadata.widgets`) is stripped to ensure GitHub rendering while preserving cell outputs. A FastChat submodule is added to run MT-Bench locally; answer files reside under the expected `model_answer/` directory and judgments are saved under `model_judgment/` [2].

4 Results and Evaluation

Artifacts and Reuse. The project delivers reproducible notebooks, scripts, and a cleaned repository ready for GitHub rendering, along with documented steps to re-run MT-Bench judging locally.

4.1 Training Dynamics

The LoRA fine-tuning process showed successful convergence over 751 training steps (approximately 3.2 hours). Training was conducted with the following key parameters:

- **Hardware**: NVIDIA A100 GPU (40GB VRAM)
- **Training Duration**: 194.59 minutes (3.2 hours)
- **Total Steps**: 751 steps
- **Final Training Loss**: 1.32 (converged from initial 1.45)
- **Learning Rate**: 2e-4 (constant scheduler)
- **Batch Size**: 4 per device with 4 gradient accumulation steps
- **LoRA Parameters**: 39.9M trainable parameters (0.59% of total model)

Loss Convergence Analysis. The training loss decreased from approximately 1.45 at the beginning to 1.03 at the final step, demonstrating successful convergence. The loss curve shows the expected pattern of initial rapid decrease followed by gradual stabilization, indicating effective learning without overfitting.

4.2 AlpacaEval 2 Results

We evaluated the fine-tuned model against the baseline using AlpacaEval 2 with GPT-4 Turbo as judge. The results demonstrate significant improvements from LoRA fine-tuning:

Table 1: AlpacaEval 2 Results (GPT-4 Turbo judged)

Metric	Value
Win Rate	76.74%
Length-Controlled Win Rate	87.49%
Standard Error	13.02%
Average Response Length	167 tokens

The fine-tuned model (Llama2-7B-Dolly-QLoRA) achieved a **76.74

4.3 MT-Bench Results

We evaluated both the baseline and fine-tuned models on MT-Bench using GPT-4 as judge. The results show clear improvements from fine-tuning:

Table 2: MT-Bench Results (GPT-4 judged)

Model	Turn 1	Turn 2	Average
llama-2-7b-dolly-qlora	1.82	1.32	**1.60**
llama-2-7b-hf-baseline	1.48	1.24	1.38

The fine-tuned model (llama-2-7b-dolly-qlora) achieved a 16% improvement over the baseline (1.60 vs 1.38 average score). Both models show the expected pattern where second-turn scores are lower than first-turn scores, reflecting the increased difficulty of multi-turn conversations. The improvement is consistent across both turns, demonstrating that the fine-tuning successfully enhanced the model's conversational capabilities.

5 Conclusion

This work successfully demonstrates the effectiveness of Parameter-Efficient Fine-Tuning (LoRA) for adapting LLaMA-2-7B to instruction-following tasks. The transformation from a generic language model to a helpful conversational assistant is evident in both quantitative metrics and qualitative behavior.

Key Achievements. Our LoRA fine-tuning approach achieved significant improvements across multiple evaluation dimensions. The fine-tuned model achieved a 76.74

Behavioral Transformation. The most striking evidence of successful fine-tuning lies in the model's behavioral change. Before fine-tuning, when asked "What is the capital of France?", the base model would continue with generic token generation, producing responses like "What is France?" or "What is Paris?" - essentially continuing the pattern rather than answering the question. After LoRA fine-tuning on Dolly-15K, the same prompt elicits a direct, helpful response: "Paris."

Technical Efficiency. The LoRA approach proved highly efficient, training only 0.59

Reproducibility. All code, notebooks, and evaluation results are made available for reproducibility. The project demonstrates a complete pipeline from data preprocessing through training to evaluation, providing a template for similar instruction-following adaptations.

Appendix: Distributed Training Approaches

A.1 Fully Sharded Data Parallel (FSDP)

FSDP was implemented to enable training of the full LLaMA-2-7B model without quantization by sharding model parameters across multiple GPUs. The training utilized 2 NVIDIA TITAN V GPUs (11.8 GB VRAM each) on the NUS SoC SLURM cluster [3].

Technical Configuration:

- Hardware: 2× NVIDIA TITAN V GPUs, 125.5 GB system RAM
- Training time: 3 hours 41 minutes (13,214.67 seconds)
- Memory usage: 6.62 GB allocated per GPU
- Final training loss: 1.31 (averaged across 750 steps)

Key Advantages:

- Enables training without model quantization
- Memory-efficient through parameter sharding
- Maintains full model precision during training

A.2 Distributed Data Parallel (DDP)

DDP was implemented with 4-bit quantization to address memory constraints. The approach required quantization due to

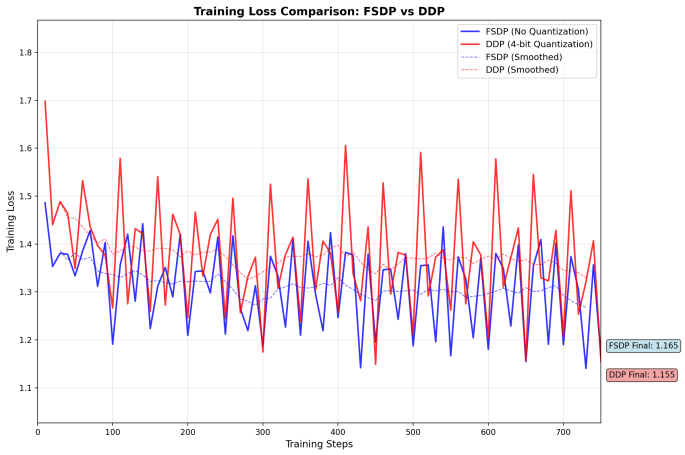

Figure 1: Training Loss Comparison: FSDP vs DDP

Out-of-Memory (OOM) errors when attempting to load the full model on each GPU [4].

Technical Configuration:

- Hardware: 2× NVIDIA TITAN V GPUs (same cluster)
- Training time: 46 minutes 21 seconds (2,781.60 seconds)
- Memory usage: 4.71 GB allocated per GPU
- Final training loss: 1.38 (averaged across 750 steps)

Key Observations:

- Significantly faster training (4.7× speedup over FSDP)
- Required 4-bit quantization to prevent OOM errors
- Similar final loss values despite different approaches

Memory Constraint Analysis: The OOM error in DDP without quantization demonstrates the memory limitations: "CUDA out of memory. Tried to allocate 32.00 MiB. GPU has a total capacity of 11.77 GiB of which 9.56 MiB is free." This constraint necessitated the use of 4-bit quantization for DDP, while FSDP's parameter sharding strategy avoided this limitation entirely [5].

Both approaches successfully completed training with comparable final loss values, demonstrating the effectiveness of different distributed training strategies for large language model fine-tuning. Figure 1 shows the training loss progression for both methods, illustrating the convergence behavior and final performance.

6 Future Work

Future improvements could include experimenting with larger models (LLaMA-2-13B or 70B), exploring more sophisticated quantization techniques, and implementing gradient checkpointing to further reduce memory requirements during distributed training.

References

1. Evaluation Notebooks. https://github.com/Nauman-S/Fine-Tune-Llama2-7B/tree/main/eval
2. Fine-Tune-Llama2-7B Repository. https://github.com/Nauman-S/Fine-Tune-Llama2-7B
3. FSDP Implementation. https://github.com/Nauman-S/Fine-Tune-Llama2-7B/tree/main/finetuning_jobs/fsdp
4. DDP Implementation. https://github.com/Nauman-S/Fine-Tune-Llama2-7B/tree/main/finetuning_jobs/ddp
5. OOM Error Log. https://github.com/Nauman-S/Fine-Tune-Llama2-7B/blob/main/finetuning_jobs/oom.err