Assignment 7 Report

Wu Rui E1373544

1. Dataset

1.1 Dataset source

The training dataset used in this assignment is Databricks Dolly 15K (an open dataset on Hugging Face, available as databricks/databricks-dolly-15k). This dataset contains 15,000 records that follow instructions, and each record typically consists of an instruction, context, and response, making it suitable for instruction fine-tuning of large language models.

1.2 Preprocessing

To ensure that the data format is consistent with the input of the model and conforms to the prompt format of instruction-tuning, the main preprocessing steps include:

1) Data Cleaning: All entries were iterated. Any row where the response field was empty was discarded to ensure all training examples were valid.
2) Whitespace Stripping: Leading and trailing whitespace was removed from the instruction, context, and response fields to maintain consistency.
Prompt Formatting: Each valid record was then formatted into a single text entry using a standardized instruction-following template.
3) Final Structure: Each formatted string was saved within a JSON object under the key "text", resulting in a final structure of {"text": "..."} for each line in the output files.

1.3 Splits

To ensure reproducible results, the entire processed dataset was first shuffled globally using a fixed random seed (seed=42).

After shuffling, the dataset was divided into three distinct splits based on the following proportions: Training Set: 80%, Validation Set: 10%, Test Set: 10%.

These splits were then saved as separate .jsonl files (train.jsonl, validation.jsonl, test.jsonl) in the data/dolly15k_prepared directory.

2. Implementation

2.1 Model Setup

The model was loaded using AutoModelForCausalLM. To optimize for memory and speed, the model was loaded with torch_dtype=torch.bfloat16 precision. Further performance enhancement was achieved by enabling TF32 for CUDA matrix multiplication operations, ensuring efficient matrix multiplication computations.

2.2 LoRA config

Parameter-Efficient Fine-Tuningis employed using the LoRA (Low-Rank Adaptation) method.

The LoraConfig from the peft library was initialized with the following key parameters:

Parameter	Value	Parameter	Value
r	8	target_modules	None
lora_alpha	16	bias	"none"
lora_dropout	0.05	task_type	CAUSAL_LM

After this setup, the final number of trainable parameters is 4,194,304, accounting for 0.0622% of the total parameters (6,742,609,920).

2.3 Tokenization & Data Collation

The corresponding AutoTokenizer was loaded to match LLaMA-2. Moreover, the pad_token of the tokenizer was set to be the same as its eos_token (End-of-Sentence token), ensuring correct padding for sequences during batching.

The input samples were uniformly truncated and encoded, with each sample having a maximum length of 1024 tokens, and the batches were organized using DataCollatorForLanguageModeling.

2.4 Hyperparams

The Trainer class from the transformers library was used to conduct the training, configured via TrainingArguments. The key hyperparameters used are summarized below:

Parameter	Value	Parameter	Value
learning_rate	2e-4	warmup_ratio	0.03
num_train_epochs	3	weight_decay	0.0
per_device_train_batch_size	2	lr_scheduler_type	cosine
gradient_accumulation_steps	8	dataloader_num_workers	8
optim	adamw_torch	max Sequence Length	1024
fp16	False	bf16	True
gradient_checkpointing	Disabled	eval_strategy	steps
eval_steps	100	save_steps	50-200

The training objective is a causal language modeling task, using the AdamW optimizer. After each epoch, the training loss and validation loss are automatically recorded, and the best weights and the LoRA adapter configuration file adapter_config.json are saved.

3. Hardware

The fine-tuning process was performed on a Vast.ai GPU instance with the following configuration:

Component	Specification	Component	Specification
GPU	NVIDIA A100	CUDA Version	12.8
VRAM	40 GB	Runtime	61 mins
CPU	AMD EPYC 7642	System RAM	128.9 GB
Disk	TEAM TM8FFD004T SSD (4 TB)	Runtime Environment	PyTorch CUDA 12.8.1

4. Evaluation

4.1 Benchmarks

This assignment evaluated the model performance based on two standard benchmarks:

1) AlpacaEval 2: It is a single-turn instruction-following benchmark. It assesses the model's ability to follow diverse instructions and generate useful, high-quality responses, that is, the overall ability of the model in the instruction-following task. Its main metric is the win rate, which is the percentage of times the model's answer is judged by the referee model to be superior to the standard text-davinci-003 baseline answer.

2) MT-Bench (FastChat): It is a multi-turn dialogue benchmark. It evaluates the model's coherence, consistency, and ability to handle context in conversations. It can use a pairwise comparison mode to allow the referee model to directly choose between the fine-tuned model and the base model, thereby calculating the score.

4.2 Metrics

4.2.1 Metrics of AlpacaEval 2

1) win_rate: The percentage of times the fine-tuned model outperforms the baseline model in the comparison.

2) standard_error: The statistical error range of the win rate, indicating the confidence level of the result. The lower the value, the better.

3) mode: The evaluation mode. `community` refers to the standard community evaluation set that uses AlpacaEval.

4) avg_length_wins: The average token length of the responses that the fine-tuned model wins.

5) discrete_length_controlled_winrate: The corrected win rate. It penalizes "length bias" through statistical methods

to evaluate the quality of responses rather than their length more fairly.

4.2.2 Metrics of MT-bench

1) win_rate: The number of wins divided by the total number of matches played. This metric does not take into account the impact of draws.
2) loss rate: The number of losses divided by the total number of matches.
3) win_rate_adjusted: It is the most crucial comparison metric. It treats "draw" as "half victory", thereby more fairly reflecting the relative performance of the model.

4.3 Implementation

In order to conduct a quantitative comparison between the fine-tuned model and the baseline model, the evaluation process is mainly divided into two steps: answer generation and judgment model assessment. The entire process utilizes the two open-source frameworks, `AlpacaEval 2` and `FastChat (MT-Bench)`, and uniformly uses `gpt-4o-mini` as the automated judgment model to ensure the consistency of the evaluation criteria.

For AlpacaEval 2, 300 standard evaluation questions (prompts) were automatically loaded from the `alpaca_eval` package, and the model's answers were generated. Then, the answers were saved as separate JSON files, and finally, they were evaluated and scored using gpt-4o-mini.

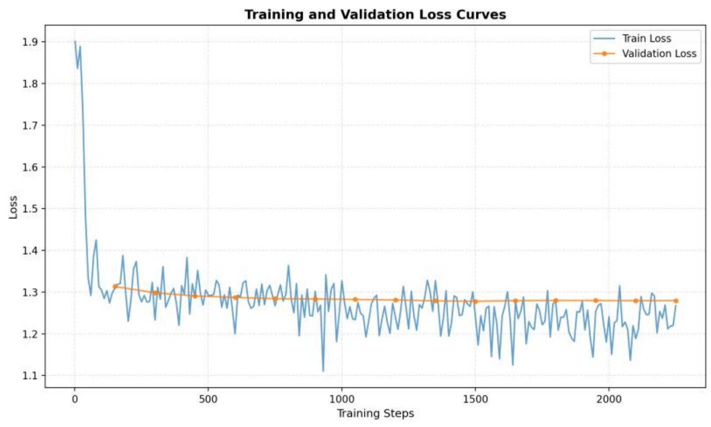
For MT-bench, 40 questions were automatically loaded from the standard evaluation problem files of MT-bench, and the answers for two models were generated. Then, the answers were saved as separate JSON files, and subsequently, they were evaluated and scored using the 'pairwise-all' mode with gpt-4o-mini.

5. Results

5.1 Loss Convergence

From the following figure, it can be seen that:

1) During the first 100 steps, there was rapid convergence, and the training loss decreased from over 1.9 to around 1.3. Then, it stabilized after 300 steps, fluctuating smoothly around an average value of approximately 1.25.
2) Both the training loss and the validation loss stabilized around 1.2-1.3, indicating strong generalization.
3) No overfitting observed, confirming the effectiveness of LoRA regularization.



5.2 Comparison based on benchmarks

5.2.1 AlpacaEval 2

Model	win_rate	standard_error	mode	avg_length	
finetuned	83	2.172340568	community	1884	
Model	**n_wins_base**	**n_draws**	**n_total**	**discrete_win_rate**	**length_controlled_winrate**
finetuned	51	0	300	83	68.83325349

From the above figure, it can be seen that:

1) Out of the total 300 evaluations, the fine-tuned model won 249 times, while the baseline model only won 51 times. The fine-tuned model has a high win rate of 83.0%, indicating that in the majority of cases, the `gpt-4o-mini` judge believes that the quality of fine-tuned model's responses is significantly superior to the base model.

2) The `avg_length_wins` is 1884, indicating that the model has learned to generate very detailed and comprehensive answers. Although the model gained an advantage in length (with a `win_rate` of 83.0% > `length_controlled_winrate` of 68.8%), the 68.8% length-controlled win rate is still an extremely excellent score. This proves that the model's responses are not just "long", but also have substantial superiority in terms of quality and helpfulness.

5.2.2 MT-bench

Model	win	loss	tie	win_rate	loss_rate	win_rate_adjusted
finetuned	28	19	34	0.350	0.225	0.5625
base	18	28	34	0.225	0.350	0.4375

From the above figure, it can be seen that:

1) Out of the 40 total evaluations, the fine-tuned model won 28 times, the baseline model won 18 times, and there were 34 ties. This indicates that in the vast majority of cases, the GPT-4o-mini judge believes that the quality of fine-tuned model's responses is superior to that of the baseline model.

2) After considering all the cases of wins, losses, and ties comprehensively, the fine-tuned model has a win_rate_adjusted of up to 0.5625, indicating that this fine-tuned model is considered superior to the baseline model in multi-round conversation tasks.

6. Discussion & Conclusion

6.1 Discussion and Insights

1) The loss curve confirms that LoRA is an efficient fine-tuning technique that can achieve significant performance improvement by reducing less than 1% of the parameters. Moreover, the model converges rapidly and does not suffer from overfitting, confirming effective learning dynamics and balanced generalization.

2) The 68.8% length_controlled_winrate of AlpacaEval 2 indicates that the fine-tuned model not only outperforms the baseline model in terms of instruction compliance, but also has learned a detailed and helpful response style.

3) The 56.25% win_rate_adjusted of MT-Bench shows that the fine-tuned model also outperforms its base model in the coherence of multi-turn conversations, the ability to follow instructions, and the ability to maintain context.

6.2 Limitations

1) The scale of Dolly-15K is relatively small. The knowledge breadth of the model is limited by these 15,000 examples, lacking diversity.

2) The evaluation only relies on gpt-4o-mini as the judge, which may introduce its own biases.

3) Without running academic benchmark tests such as MMLU or GSM8K, fine-tuning may lead to performance degradation of the model in certain factual knowledge or reasoning tasks.

6.3 Conclusion & Future work

Conclusion: This assignment successfully achieved all the predetermined goals. Efficient instruction fine-tuning on LLaMA-2-7B using LoRA was conducted, achieving stable loss convergence, and achieving significant performance improvements over the base model on the AlpacaEval 2 and MT-Bench benchmarks.

Future work:

1) Experiment with LoRA with r=16/32 and other hyperparameters to observe the changes in model performance.

2) Fine-tune on larger and more diverse instruction datasets.

3) Run benchmarks such as MMLU and GSM8K to comprehensively evaluate the impact of fine-tuning on the core reasoning and knowledge capabilities of the model.