Medical Q&A Fine-Tuning Report

Over the past month, I have been building and testing the fine-tuning pipeline. My goal was to show that we can take an existing large language model, adapt it to the medical domain, and have it answer PubMed-style or exam-style questions. Instead of immediately jumping to GPT-OSS with 20B parameters, I started with smaller models Llama-2-7B, Llama-2-13B and DialoGPT-medium to prove that our approach works.

What I Did:

1. Choosing the model and fine-tuning approach

I started with Llama-2-7B-chat-hf, a 7-billion parameter open model. I chose this because it is small enough to run on Google Colab Pro GPUs, which we have easy access to. It is also large enough to capture meaningful patterns during fine-tuning. Finally, it is a close cousin to GPT-OSS, so what we learn here applies when we scale up.

Instead of retraining all 7 billion parameters, I used LoRA adapters, only the adapters were trained. The reasons I chose LoRA was that it drastically reduces memory and compute requirements and it makes training feasible on modest GPUs. Also it lets us swap adapters in and out easily, like when training separate adapters for PubMedQA, MedMCQA or MedQA.

I set the LoRA parameters to **rank = 4** and **alpha = 8**, which means the adapters are very lightweight. This was enough for testing the pipeline without overloading the GPU.

2. Data

I created a training script to make the **Pubmed QA** data consistent with the model format:

{

"instruction": "Question: ... Options: ...",

"input": "",

"output": "Correct Answer Text"

}

I was able to convert PubMedQA into this format. This dataset is relatively simple, with short question-answer pairs, so the conversion was straightforward.

The challenges began with **MedMCQA** and **MedQA**. These datasets are multiple-choice:

- Sometimes the options are given as a dictionary (`{"A": "...", "B": "..."}`), sometimes as a list.
- The correct answer may be given as just a letter (C) or as the full text (Vitamin C).
- Some questions have more than four choices (E, F, etc.).

This makes conversion more difficult because we need to build a function that can:

- Normalize every question into a clean “Options” block (A, B, C, D.).
- Map the correct answer reliably from letter to text or match by normalized text.
- Skip malformed entries without breaking the pipeline.

I attempted this conversion previously but couldn’t finalize it in time. This is the next piece of work: to finish robust converters for MedMCQA and MedQA so they can be used in training.

3. Training pipeline

I wrote a training script (`medical_qa_robust.py`) that brings together model loading, LoRA configuration, data preparation and training.

Key things I added to make the script more robust:

- **InstructionDataset class**: formats each example into instruction -> input -> output and tokenizes it.
- **RobustDataCollator**: fixes tensor shape mismatches during batching, which is a common cause of errors in multi-choice data.
- **Tokenizer fixes**: ensures that a `[PAD]` token is available and adds one if not and resizes the embeddings if needed.
- **Stage training logic**: the script is structured in three stages:
  Stage 1 (PubMedQA), Stage 2 (MedMCQA), Stage 3 (MedQA).

Right now, I have only run Stage 1.

4. Running the training with PubMedQA

I trained the model on PubMedQA for a short test run:

- **Steps:** 50, about 0.4 of an epoch with the tiny dataset.
- **Batch size:** 1, with gradient accumulation of 4 to simulate a larger batch.
- **Max sequence length:** 512 tokens.
- **Learning rate:** 5e-5.

Results:

- Training loss dropped from around **1.79** -> **1.72**, about a 3% improvement in just 50 steps.
- The model is saved to `./checkpoints/pubmedqa`.
- I confirmed that the checkpoint can be reloaded and used for generation.

This was exactly the outcome I wanted: proof that the pipeline works end-to-end.

5. Why I didn't use GPT-OSS (20B)

The long-term goal is to move to **GPT-OSS-20B**, but there are blockers:

- 20B parameters need 16–40GB VRAM even with quantization, which Colab GPUs don't have.
- GPT-OSS requires a special tokenizer and a harmony format for responses.
- It needs specific versions of `bitsandbytes` (for quantization) and `openai-harmony`.
- To fit in memory, we'd need 4-bit quantization, batch size = 1, sequence length 256–512, and gradient accumulation 16–32.

What's missing:

- Finish the data converters for MedMCQA and MedQA so all three datasets are in the same format.
- Move to GPT-OSS once we secure a GPU with enough VRAM.
- Evaluate multiple-choice accuracy, text similarity metrics (Exact Match, ROUGE, BLEU), and later clinical safety checks.

Reward model and KL penalty:

So far, I have only done SFT, the model is trained to mimic correct answers. This is a good start, but it doesn't guarantee the model's answers are always preferable, meaning safe, clinically cautious or aligned with our preferences.

That's where reward models and KL divergence come in:

- A reward model (RM) is trained separately to score answers based on preferences (factuality, safety, bedside manner).
- With the RM in place, we can apply reinforcement learning (PPO or DPO).
- The KL penalty is a leash that prevents the fine-tuned model from drifting too far away from the base model. Without it, the model might produce unnatural or repetitive answers.

When to add this:

After we scale to GPT-OSS and once we have the complete datasets and supervised fine-tuning is stable. At that point, we can introduce DPO or PPO to make the model not only accurate but also aligned with human and clinical preferences.

Conclusion

In short, we proved that the training pipeline works with PubMedQA and LoRA adapters. The model trained, loss decreased and checkpoints saved correctly. The main blockers now are dataset conversion for MedMCQA/MedQA and infrastructure for GPT-OSS. Once we overcome those, we can scale training, add evaluation metrics and eventually move to preference optimization with a reward model and KL penalty for safety.