Department of Information Technology,
B. P. Poddar Institute of Management and Technology

Leveraging Generative AI for Medical Data Prediction

Under the Guidance of:
Dr. Gitosree Khan

Presented by:
Abhishek Saha

Sougata Seth

Sarbasish Chowdhury

Ayush Kumar Pandey

TABLE OF CONTENTS

Introduction & Motivation

Why This Project?

1. **Healthcare needs** intelligent systems for **early diagnosis**, **decision support**, and **personalized treatment**.
2. Traditional models struggle with the **complexity and variability** of medical data.

Our Solution:

LLMs can understand medical text and answer clinical questions.
We fine-tuned **Meta-Llama-3.1-8B-Instruct-bnb-4bit** for **medical QA** using:
→ Supervised Fine-Tuning (SFT)
→ Reinforcement Learning (PPO)
→ Efficient tuning via **LoRA** and **4-bit quantization**

Problem Statement:

General LLMs fall short in medicine

- Trained on **general language**, not clinical data
- Tend to **hallucinate** or give incomplete medical answers

Full fine-tuning is resource-intensive

- Needs **high-end GPUs/TPUs**
- Not feasible for academic or clinical labs with limited infrastructure

Our Need

- A **domain-specific**, **accurate**, and **resource-efficient** medical LLM
- One that works on **consumer hardware** (like RTX 3070Ti)

Our Objective

Adapt

a general LLM
(Meta-LLaMA 3.1 8B instruct) to
medical question answering

Minimize compute & memory using:

- 4-bit quantization
- LoRA (Low-Rank Adaptation)

Improve accuracy using a two-stage training pipeline:

Stage 1: Supervised Fine-Tuning (SFT)

Stage 2: Reinforcement Learning (PPO)

Deploy

locally on consumer-grade GPUs
(e.g., RTX 3070Ti)

Dataset Overview

MedQA (USMLE)

- 12,723 clinical MCQs
- US medical licensing exam
- Focus on deep clinical reasoning

MedMCQA (NEET PG/AIIMS)

- 94,000+ questions
- Indian postgraduate exam dataset
- Covers 21 subjects, 2.4k topics
- Sampled 50,000 for training

Architecture Overview

End-to-End Pipeline:

1. **Data Ingestion**
   → Load MedQA & MedMCQA
2. **Preprocessing**
   → Prompt formatting & tokenization
3. **Base Model Loading**
   → Meta-LLaMA-3.1-8B (4-bit)
4. **Training**
   → Stage 1: SFT
   → Stage 2: PPO
5. **Deployment**
   → Python CLI using **llama-cpp**

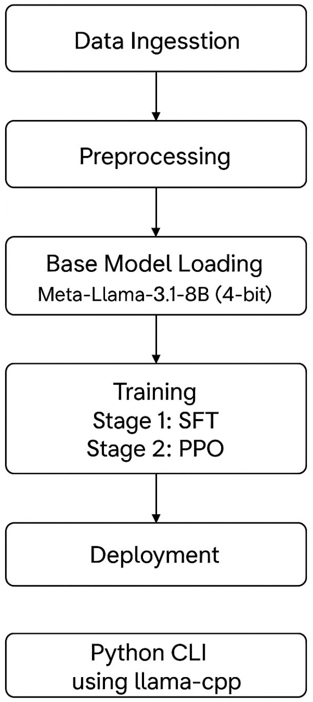

Fig 1. Architecture Overview

Base Model, LoRA & QLoRA

Meta-LLaMA-3.1-8B (4-bit)

1. Open-source LLM from Meta
2. 8B parameters
3. Loaded using **4-bit quantization** (bitsandbytes)

LoRA + QLoRA for Efficient Training

1. LoRA (Low-Rank Adaptation): fine-tune only attention sublayers (q_proj, k_proj)
2. QLoRA: a quantization-aware variant of LoRA
   a. Enables LoRA on 4-bit models
   b. Uses paged optimizers & NF4 quantization
   c. Ideal for consumer GPUs

Configuration:

1. Rank = 16
2. Alpha = 8
3. Target Modules = q_proj, k_proj

```python
from transformers import AutoModelForCausalLM, AutoTokenizer
from peft import LoraConfig, get_peft_model

# 1. Load the 4-bit quantized base model
tokenizer = AutoTokenizer.from_pretrained("unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit")
model = AutoModelForCausalLM.from_pretrained(
    "unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit",
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    device_map="auto"
)

# 2. Configure QLoRA (LoRA on 4-bit)
lora_config = LoraConfig(
    r=16,
    lora_alpha=8,
    target_modules=["q_proj", "k_proj"],
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)

# 3. Verify trainable parameters
model.print_trainable_parameters()
```

Fig 2. Example snippet

Supervised Fine-Tuning (SFT)

- **Training Config:**
  Batch size = 1 + 8-step accumulation; Epochs = 5; Learning rate = 3×10^{-6};
  FP16 + gradient checkpointing
- **Data:** MedQA + MedMCQA prompts
- **Outcome:** ~62% accuracy post-SFT

```python
class MedicalDataProcessor:
    """Data processor for medical datasets"""

    def __init__(self, tokenizer, max_length: int = 2048):
        self.tokenizer = tokenizer
        self.max_length = max_length

        if self.tokenizer.pad_token is None:
            self.tokenizer.pad_token = self.tokenizer.eos_token
            self.tokenizer.pad_token_id = self.tokenizer.eos_token_id

    def create_prompt_template(self, question: str, answer: str) -> str:
        """Create prompt template"""
        return f"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a knowledgeable medical AI assistant. Provide accurate, evidence-based medical information.
Always recommend consulting healthcare professionals for personal medical advice.<|eot_id|><|start_header_id|>user<|end_header_id|>

{question}<|eot_id|><|start_header_id|>assistant<|end_header_id|>

{answer}<|eot_id|>"""
```

Fig 3. Prompt Template

PPO – Reinforcement Learning Stage

Purpose:

- Refine the model's medical reasoning after SFT
- Reward correct answers, penalize wrong ones

PPO Config:

- Reward signal:
  - 1.0 → Correct
  - 0.1 → Incorrect
- Epochs: 2
- Batch size: 2
- Clip range (ε): 0.2
- Learning rate: 5×10^{-6}

```python
class MedicalRewardFunction:

    def calculate_medical_accuracy_reward(self, response: str, correct_answer: str, question: str) -> float:
        """Calculate reward based on medical accuracy"""
        response_clean = response.strip().upper()
        correct_clean = correct_answer.strip().upper()

        # Base accuracy reward
        if correct_clean in response_clean:
            base_reward = 1.0
        else:
            # Check for partial matches or correct choice letter
            answer_match = re.search(r'[ABCD]', response_clean)
            if answer_match and answer_match.group() == correct_clean:
                base_reward = 1.0
            else:
                base_reward = 0.1

        return base_reward

    def calculate_medical_quality_reward(self, response: str, question: str) -> float:
        """Calculate reward based on medical response quality"""
        response_lower = response.lower()
        quality_score = 0.0

        # Evidence-based language bonus
        evidence_count = sum(1 for keyword in self.medical_keywords['high_reward']
                             if keyword in response_lower)
        quality_score += min(evidence_count * 0.1, 0.3)

        # Safety awareness bonus
        safety_count = sum(1 for keyword in self.medical_keywords['safety_critical']
                           if keyword in response_lower)
        quality_score += min(safety_count * 0.15, 0.2)

        # Specialty-specific knowledge bonus
        question_lower = question.lower()
        for specialty, keywords in self.specialty_knowledge.items():
            if any(keyword in question_lower for keyword in keywords):
                specialty_mentions = sum(1 for keyword in keywords if keyword in response_lower)
                quality_score += min(specialty_mentions * 0.05, 0.15)
                break

        # Length appropriateness (not too short, not too verbose)
        response_length = len(response.split())
        if 10 <= response_length <= 100:
            quality_score += 0.1
        elif response_length < 5:
            quality_score -= 0.2
        elif response_length > 200:
            quality_score -= 0.1

        return min(quality_score, 1.0)
```

Fig 4. Accuracy and quality reward functions

Training Metrics & Graphs

Key Observations:

- **Training loss** decreases smoothly to ~1.4
- **Validation loss** remains stable → no overfitting
- **Learning rate** follows cosine decay → smooth convergence
- **Gradient norms** stable around 0.2–0.3 → numerically safe training

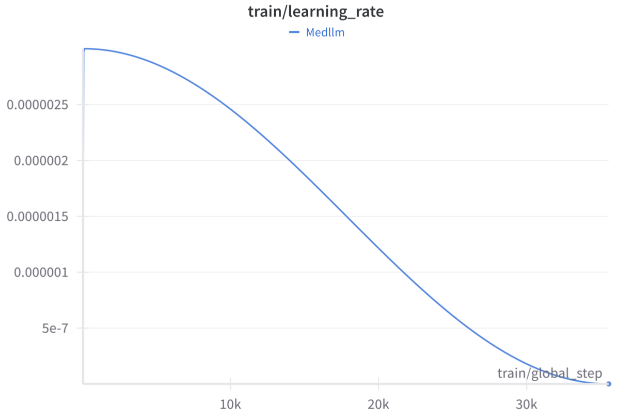

Fig 5. Learning Rate

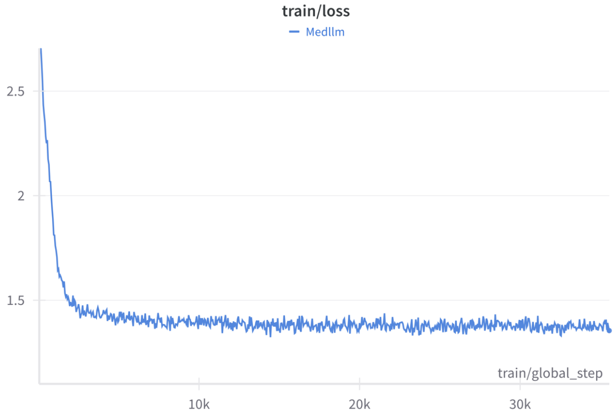

Fig 6. Training Loss

Results & Qualitative Comparison

Question:
Which vitamin deficiency causes scurvy?

SFT Answer:
“Scurvy is caused by a deficiency of Vitamin C.”

PPO Answer:
“Scurvy is caused by a deficiency of Vitamin C, a water-soluble vitamin essential for collagen synthesis. Deficiency leads to fatigue, gum bleeding, and poor wound healing.”

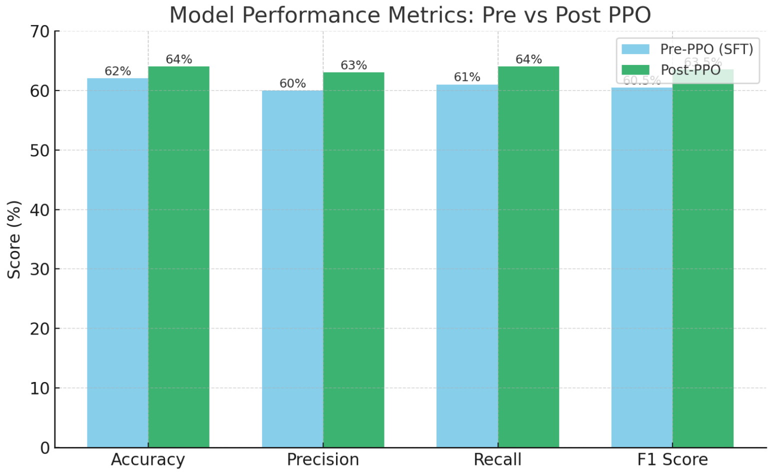

Fig 7. PPO improves all key metrics, enhancing model accuracy and consistency

Comparative Analysis

Model	Size	Accuracy	Hardware	Method	Open Source
MedLLM (Ours)	8B	~64%	RTX 3070Ti	SFT + PPO + QLoRA	✅ Yes
Med42	70B	~72%	A100	LoRA (SFT)	✅ Yes
Med-PaLM 2	540B	86.5%	TPU Pods	RLHF	❌ No
HuatuoGPT	8B	~70%	High-end GPU	SFT + PPO	✅ Partial

Table 1. Comparison

Future Scope

01
Multi-modal Expansion

02
Online PPO with Clinician Feedback

03
Ensemble QA System

04
Explanation & Trust Layer

05
Clinical Integration & Deployment

Conclusion

- **MedLLM** adapts Meta-LLaMA-3.1 to the **medical domain**
- Efficient training using **4-bit QLoRA** and **consumer GPU (RTX 3070Ti)**
- Achieved **~64% accuracy** with **2-stage fine-tuning (SFT + PPO)**
- Fully **open-source** and locally deployable
- Paves the way for **clinical AI** that is lightweight, explainable, and accessible

References

- **Hu et al., Med42: LoRA-based Fine-Tuning of Medical LLMs**
  arXiv:2404.14779, 2024
- **Touvron et al., LLaMA: Open and Efficient Foundation Language Models**
  arXiv:2302.13971, Meta AI
- **Dettmers et al., QLoRA: Efficient Finetuning of Quantized LLMs**
  arXiv:2305.14314, HuggingFace
- **Schulman et al., Proximal Policy Optimization Algorithms (PPO)**
  arXiv:1707.06347, OpenAI
- **HuatuoGPT: Tuning and Evaluating Medical LLMs in Chinese**
  arXiv:2304.06975, 2023
- **MedQA & MedMCQA Datasets**
  - USMLE-based MCQs: https://github.com/jind11/MedQA
  - Indian NEET-PG MCQs: https://github.com/medmcqa/medmcqa


THANK YOU