Introduction to Finetuning + RAG with LLMs for Healthcare Applications

The Need for Specialized AI in Healthcare

1. Complexity of Medical Knowledge
   - Vast and rapidly evolving field
   - Specialized terminology and concepts
   - Interdisciplinary nature of modern medicine
2. Critical Importance of Accuracy
   - Direct impact on patient care and safety
   - Need for up-to-date, evidence-based information
   - Reducing risks of misinformation
3. Personalization in Healthcare
   - Tailoring responses to individual patient needs
   - Incorporating latest research and treatment options
   - Supporting precision medicine initiatives
4. Enhancing Healthcare Professional Efficiency
   - Quick access to relevant medical literature
   - Assistance in diagnosis and treatment planning
   - Time-saving in research and information retrieval
5. Bridging Knowledge Gaps
   - Connecting disparate areas of medical research
   - Facilitating interdisciplinary insights
   - Supporting continuous medical education
6. Ethical and Reliable AI
   - Ensuring transparency and explainability in AI decisions
   - Maintaining privacy and security of medical data
   - Building trust in AI-assisted healthcare

Introduction

- Llama 3.1: Meta's latest Large Language Model
  - 8 billion parameters
  - Improved performance over previous versions
  - Open-weights model allowing for fine-tuning
- Fine-tuning: Process of adapting pre-trained models
  - Enhances performance on specific tasks or domains
  - Requires less data than training from scratch
- PubMedQA Dataset:
  - Collection of 1,000 medical research questions
  - Expert-written answers based on PubMed abstracts
  - Ideal for training medical AI assistants
- Goal: Create a specialized medical AI capable of answering complex healthcare queries

Setup and Requirements

- Hardware:
  • A100 GPU with 40GB VRAM (recommended)
  • Allows for efficient training of 8B parameter model
- Key libraries and their roles:
  • transformers: Provides pre-trained models and fine-tuning utilities
  • datasets: Efficient data loading and preprocessing
  • peft: Parameter-Efficient Fine-Tuning techniques (e.g., LoRA)
  • trl: Transformer Reinforcement Learning, includes SFTTrainer
  • unsloth: Optimizations for faster and more efficient fine-tuning
- GPU acceleration:
  • Crucial for handling large models and datasets
  • Enables 4-bit quantization and mixed-precision training
- Python environment:
  • Python 3.10+
  • CUDA 11.7+ for GPU support

Model Preparation

- Loading Llama 3.1 8B model:
  - Source: "unsloth/Meta-Llama-3.1-8B-bnb-4bit"
  - Pre-quantized for efficient loading
- 4-bit quantization:
  - Reduces memory footprint by 75% compared to FP16
  - Enables fine-tuning of larger models on limited hardware
- LoRA (Low-Rank Adaptation) setup:
  - r=16: Rank of LoRA update matrices
  - lora_alpha=16: Scaling factor for LoRA updates
  - Target modules: q_proj, k_proj, v_proj, up_proj, down_proj, o_proj, gate_proj
  - Covers both attention mechanisms and feed-forward networks
- RSLoRA (Rank-Stabilized LoRA):
  - Improves training stability, especially for higher ranks
- Gradient checkpointing:
  - Trades computation for memory efficiency
  - Allows for larger batch sizes or model sizes

Dataset Preparation

- PubMedQA dataset structure:
  - Context: Relevant medical literature excerpt
  - Question: Research or clinical query
  - Answer: Expert-written response
- Data formatting process:
  - Load raw dataset using Hugging Face datasets
  - Format each entry into conversation structure
  - Apply chat template for consistency
- Chat template application:
  - Ensures uniform input format for model
  - Prepares data for instruction-following tasks
  - Uses ChatML format: human ... assistant ...
- Dataset statistics:
  - Total entries: 1,000
  - Average context length: [X] tokens
  - Average question length: [Y] tokens
  - Average answer length: [Z] tokens

Training Configuration

- SFTTrainer setup:
  - Part of TRL (Transformer Reinforcement Learning) library
  - Streamlines supervised fine-tuning process
- Key training arguments:
  - Learning rate: 3e-4 (0.0003)
  - LR scheduler: Linear decay
  - Per device batch size: 1 (limited by GPU memory)
  - Gradient accumulation steps: 8 (effective batch size of 8)
  - Number of epochs: 1 (single pass through the dataset)
  - Mixed precision: FP16 or BF16 (auto-detected)
- Optimization techniques:
  - Paged AdamW 8-bit optimizer: Memory-efficient variant of AdamW
  - Weight decay: 0.01 for regularization
  - Warmup steps: 10 for gradually increasing learning rate
- Additional settings:
  - Max sequence length: 2048 tokens
  - Packing: True (combines shorter sequences for efficiency)
  - Seed: 0 (for reproducibility)

Fine-tuning Process

- Training loop explanation:
  - Load and preprocess batch of data
  - Forward pass through the model
  - Calculate loss
  - Accumulate gradients over 8 steps
  - Update model parameters
  - Repeat for entire dataset

- Resource management:
  - 4-bit quantization reduces memory usage by 75%
  - Gradient accumulation allows for larger effective batch sizes
  - Packing optimizes GPU utilization for varying sequence lengths
- Monitoring training progress:
  - Loss curves: Should show decreasing trend
  - Learning rate schedule: Linear decay from 3e-4
  - GPU utilization: Aim for >90% consistently
  - Memory usage: Monitor for any OOM (Out of Memory) issues

Model Saving and Loading

- Merging LoRA weights:
  • Combines fine-tuned adapters with base model
  • Results in a single, updated model
- Saving the fine-tuned model locally:
  • Model architecture: Saved in config.json
  • Weights: Stored in PyTorch format (.bin files)
  • Tokenizer: Vocabulary and special tokens saved separately
- Loading the model for inference:
  • Use FastLanguageModel for efficient loading
  • Set up for float16 inference to balance speed and accuracy
  • Max sequence length increased to 4096 for inference
- Applying chat template for consistent input formatting:
  • Ensures queries are formatted identically to training data
  • Helps model understand role (human/assistant) and turn structure

Inference and Usage

- Setting up the model for inference:
  • Max sequence length: 4096 tokens (allows for longer conversations)
  • Device mapping: "auto" for optimal resource usage across available GPUs
  • Data type: torch.float16 for efficient inference
- Generating responses:
  • Temperature: 0.7 (balances creativity and coherence)
  • Top-p (nucleus) sampling: 0.9
  • Max new tokens: 200 (adjustable based on desired response length)
  • Number of return sequences: 1 (can be increased for diverse outputs)
- Response generation function:
  - Format input using chat template
  - Tokenize and move to GPU
  - Generate output using model.generate()
  - Decode output and extract response
- Handling of chat history:
  • Maintain conversation context for multi-turn interactions
  • Truncate history if exceeding max sequence length

Live Demo Time!

Challenges and Considerations

- GPU memory constraints:
  - 8B parameter model requires significant VRAM
  - 4-bit quantization and gradient checkpointing as mitigation strategies
- Quantization trade-offs:
  - 4-bit: Maximum memory efficiency, potential accuracy loss
  - 8-bit: Balance between efficiency and performance
  - Full precision: Highest accuracy, highest resource requirement
- Balancing performance and resource usage:
  - Batch size adjustments impact training speed and memory usage
  - Gradient accumulation as a workaround for larger effective batch sizes
- Potential overfitting on small datasets:
  - PubMedQA's 1,000 samples may not cover all medical topics
  - Strategies: Early stopping, increased regularization, data augmentation

Future Improvements

- Extended fine-tuning strategies:
  • Increase training epochs for potentially better performance
  • Experiment with larger medical datasets or dataset combinations
  • Investigate continual learning approaches
- Potential healthcare applications:
  • Medical question answering systems for professionals
  • Assisting in literature reviews and research
  • Patient education tools with simplified explanations
  • Clinical decision support systems (with appropriate validations)
- Integration with other medical datasets:
  • Combine with Electronic Health Records (EHRs) for personalized medicine
  • Incorporate medical imaging datasets for multimodal learning
  • Integrate with latest medical guidelines and research papers
- Ethical considerations and bias mitigation:
  • Ensure diverse representation in training data
  • Implement fact-checking mechanisms for critical information
  • Clear disclosure of AI-generated content to end-users
  • Regular audits for biases and inaccuracies

Let's Rove onto RAG...

Introduction to Retrieval-Augmented Generation (RAG)

- Combines information retrieval with language generation
- Key components:
  • Retriever: Finds relevant documents from a knowledge base
  • Generator: Produces responses based on retrieved information
- Benefits:
  • Up-to-date information without full model retraining
  • Improved accuracy and factual grounding
  • Transparency through source attribution

RAG in Medical Context

- Knowledge base options:
  • PubMed abstracts
  • Medical textbooks
  • Clinical guidelines
  • EHR data (with proper anonymization)
- Advantages in healthcare:
  • Access to latest medical research
  • Ability to cite specific studies or guidelines
  • Reduced hallucination on critical medical information
- Challenges:
  • Ensuring retrieval of high-quality, peer-reviewed information
  • Handling complex medical terminology
  • Balancing between general knowledge and specific studies

Implementing RAG with Fine-tuned Llama 3.2

- Retriever options:
  • Dense retrieval (e.g., using FAISS)
  • Sparse retrieval (e.g., BM25)
  • Hybrid approaches
- Process flow:
  - Receive medical query
  - Retrieve relevant documents from medical knowledge base
  - Concatenate query with retrieved information
  - Generate response using fine-tuned Llama 3.2
- Implementation considerations:
  • Efficient indexing of large medical corpora
  • Balancing retrieval speed and accuracy
  • Handling of long contexts within model's maximum sequence length

RAG Performance and Evaluation

- Metrics to consider:
  - Accuracy of medical information
  - Relevance of retrieved documents
  - Response coherence and fluency
- Evaluation methods:
  - Expert review by healthcare professionals
  - Automated metrics (e.g., BLEU, ROUGE for text similarity)
  - Comparison with non-RAG baseline
- Expected improvements:
  - Higher factual accuracy
  - More comprehensive answers
  - Better handling of rare or recent medical topics

Future Directions with RAG

- Personalized medicine:
  - Incorporating patient-specific data in retrieval process
  - Tailoring responses based on individual medical histories
- Multimodal RAG:
  - Integrating text-based retrieval with medical imaging
  - Enhancing diagnoses with visual information
- Continuous learning:
  - Regularly updating the knowledge base with new research
  - Fine-tuning the retriever and generator on user interactions
- Ethical considerations:
  - Ensuring privacy in medical data retrieval
  - Providing clear citations for retrieved information
  - Maintaining transparency about AI-generated content