Fine-Tuning a Large Language Model for Medical Question Answering

Technical Report

Author: Atharva Kurlekar
Course: Prompt Engineering
Submission Date: April 22, 2025

Abstract

This technical report documents the implementation of a specialized medical question-answering system created by fine-tuning a pre-trained language model. Using Google's Flan-T5-base model as the foundation, the system was fine-tuned on a small but carefully curated dataset of medical questions and answers. The implementation demonstrates how even a modest dataset can effectively adapt a general-purpose language model to a specialized domain. The resulting model shows improved capability in answering medical questions compared to the base model, though limitations remain due to the dataset size constraints. The report details the methodology, results, and future improvement possibilities for this medical LLM fine-tuning project.

1. Introduction

Large language models (LLMs) have demonstrated impressive capabilities in open-domain question answering but often lack precision in specialized domains like medicine. Medical information requires accuracy, reliability, and nuance that general-purpose models may not provide without domain-specific adaptation. This project explores how fine-tuning can help bridge this gap by adapting a pre-trained model to better handle medical queries.

The approach uses instruction-tuning, where the model is trained on examples formatted as instruction-output pairs. This method has proven effective for specialized applications, allowing models to better understand domain-specific patterns and terminology.

The primary objective of this project was to develop a functional medical question-answering system through fine-tuning, with the following specific goals:

1. Prepare a small but high-quality medical QA dataset
2. Select an appropriate base model for fine-tuning
3. Implement an effective fine-tuning pipeline
4. Develop a functional inference system for testing
5. Evaluate the model's performance and identify limitations

2. Methodology

2.1 Dataset Preparation

The implementation creates a dataset consisting of 13 medical question-answer pairs. While small, the dataset was designed to be high-quality and representative of typical medical queries:

```
medical_data = [
    {
        "instruction": "What are the symptoms of type 2 diabetes?",
        "output": "The main symptoms of type 2 diabetes include increased
thirst, frequent urination, excessive hunger, fatigue, blurred vision, slow-
healing sores, and recurring infections."
    },
    # ... additional examples
]

# Add 30% more examples (3 more to make 13 total)
additional_examples = [
    {
        "instruction": "What causes migraines?",
        "output": "Migraines are caused by abnormal brain activity affecting
nerves and blood vessels, often triggered by stress, hormonal changes,
certain foods, bright lights, and changes in sleep patterns."
    },
    # ... additional examples
]
```

The dataset covers diverse medical topics including:

- Chronic conditions (diabetes, hypertension)
- Diagnostic procedures (hypertension diagnosis, strep throat diagnosis)
- Medications and treatments (asthma medications, statins)
- Disease comparisons (Alzheimer's vs. dementia, rheumatoid vs. osteoarthritis)
- Emergency conditions (stroke warning signs, heart attack symptoms)

Each example follows a consistent instruction-output format designed for the instruction-tuning approach.

2.2 Model Selection

The implementation selects Google's Flan-T5-base model as the foundation for fine-tuning:

```
model_name = "google/flan-t5-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

This model was chosen for several key reasons:

1. **Architecture**: The encoder-decoder architecture of T5 is well-suited for text generation tasks
2. **Size**: With approximately 250 million parameters, the base variant offers a good balance between capability and computational efficiency
3. **Prior Instruction Tuning**: Flan-T5 has already been exposed to instruction-following tasks, making it receptive to the fine-tuning approach

2.3 Data Preprocessing

The code implements a preprocessing function that:

1. Prefixes each question with "Medical question: " to provide consistent context
2. Tokenizes both inputs and outputs with appropriate padding and truncation
3. Handles padding tokens in labels correctly to avoid affecting the loss calculation

```
def preprocess_function(examples):
    # Simply prefix the instruction with "Medical question: "
    sources = ["Medical question: " + i for i in examples["instruction"]]
    targets = examples["output"]

    # Tokenize inputs
    inputs = tokenizer(
        sources,
        max_length=max_source_length,
        padding="max_length",
        truncation=True
    )

    # Tokenize targets
    outputs = tokenizer(
        targets,
        max_length=max_target_length,
        padding="max_length",
        truncation=True
    )

    # Replace padding token id with -100 in labels
    batch["labels"] = [
```

```
        [label if label != tokenizer.pad_token_id else -100 for label in
labels]
        for labels in batch["labels"]
    ]

    return batch
```

2.4 Fine-Tuning Configuration

The implementation configures the fine-tuning process with the following parameters:

```
training_args = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=4,
    gradient_accumulation_steps=2,
    learning_rate=1e-4,
    num_train_epochs=10,
    weight_decay=0.01,
    save_strategy="epoch",
    save_total_limit=2,
    logging_steps=10,
    fp16=False,
    report_to="none"
)
```

Key hyperparameters:

- **Learning rate**: 1e-4 (standard for fine-tuning T5 models)
- **Batch size**: 4 with gradient accumulation of 2 (effective batch size of 8)
- **Weight decay**: 0.01 (for regularization)
- **Training epochs**: 10 (extended to compensate for small dataset size)

The code uses the Hugging Face Trainer API to handle the training loop, optimization, and checkpointing:

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    data_collator=data_collator,
)

trainer.train()
```

3. Inference Pipeline

A critical component of the implementation is the inference pipeline that enables interaction with the fine-tuned model:

```
def generate_answer(question):
    input_text = f"Medical question: {question}"
    inputs = tokenizer(input_text, return_tensors="pt", padding=True,
                       truncation=True, max_length=max_source_length)

    # Move to GPU if available
    if torch.cuda.is_available():
        inputs = {k: v.to("cuda") for k, v in inputs.items()}
        model.to("cuda")

    # Generate answer
    outputs = model.generate(
        input_ids=inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_length=max_target_length,
        do_sample=True,
        temperature=0.3,
        num_beams=4,
        no_repeat_ngram_size=3
    )

    # Decode and return the answer
    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

The generation parameters are carefully tuned:

- **Temperature**: 0.3 (lower value reduces randomness for more focused outputs)
- **Beam search**: 4 beams (explores multiple generation paths)
- **No repeat n-gram size**: 3 (prevents repetition in generated text)

Additionally, the code creates a standalone inference script (`medical_qa_inference.py`) that provides an interactive command-line interface for testing:

```
# Interactive mode
if __name__ == "__main__":
    print("Medical QA System")
    print("Type 'exit' or 'quit' to end the session")

    while True:
        question = input("\nMedical question: ")
        if question.lower() in ["exit", "quit"]:
            break

        answer = generate_answer(question)
        print(f"\nAnswer: {answer}")
```

4. Results and Evaluation

4.1 Testing Approach

The implementation tests the model by passing a set of medical questions through the inference pipeline:

```
test_questions = [
    "What are the symptoms of type 2 diabetes?",
    "How is hypertension diagnosed?",
    # ... additional test questions
]

for test_question in test_questions:
    print(f"\nQuestion: {test_question}")
    print(f"Generated Answer: {generate_answer(test_question)}")
```

This approach allows for qualitative assessment of the model's responses to both seen and unseen questions.

4.2 Observed Results

The fine-tuned model demonstrates the ability to generate relevant, concise answers to medical questions within its training scope. For questions that closely match the training examples, the model produces accurate responses that capture the key medical information.

Due to the limited dataset size, the model's knowledge is naturally constrained to the topics covered in the training data. Questions outside this scope may receive less accurate or complete responses.

The model shows some ability to handle variations of trained questions, suggesting limited generalization capability despite the small dataset size.

5. Limitations and Future Improvements

5.1 Current Limitations

The implementation has several significant limitations:

1. **Limited Dataset Size**: With only 13 examples, the model's medical knowledge is extremely restricted in scope.
2. **Lack of Formal Evaluation**: The code does not implement quantitative metrics to objectively assess performance.
3. **Single Hyperparameter Configuration**: No hyperparameter optimization is performed to find the optimal training settings.
4. **No Uncertainty Expression**: The model does not indicate when it is uncertain or lacks sufficient knowledge to answer a question.

5.2 Future Improvements

Based on these limitations, several enhancements could significantly improve the system:

1. **Dataset Expansion**: Increasing the dataset to hundreds or thousands of examples would substantially improve coverage and performance.
2. **Formal Evaluation Framework**: Adding quantitative metrics such as BLEU, ROUGE, and exact match would enable objective assessment.
3. **Hyperparameter Optimization**: Implementing a systematic search across learning rates, batch sizes, and training durations could improve model performance.
4. **Medical Verification**: Establishing a process for medical experts to verify responses would be essential for any practical application.
5. **Uncertainty Quantification**: Adding mechanisms for the model to express confidence levels would improve trustworthiness.

6. Conclusion

This project demonstrates a functional implementation of fine-tuning a language model for medical question answering. Despite using a minimal dataset of just 13 examples, the fine-tuned Flan-T5-base model shows an ability to generate relevant, coherent responses to medical questions within its training scope.

The implementation includes a complete pipeline from data preparation through fine-tuning to inference, providing a foundation for more sophisticated medical AI systems. While the current system has significant limitations due to data constraints and the absence of formal evaluation, it illustrates the potential of targeted fine-tuning for domain adaptation of large language models.

With the additional improvements outlined in this report, this basic system could evolve into a more robust and comprehensive medical question-answering tool.

References

1. Chung, H. W., et al. (2022). Scaling Instruction-Finetuned Language Models. arXiv preprint arXiv:2210.11416.
2. Raffel, C., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research, 21(140), 1-67.
3. Wolf, T., et al. (2020). Transformers: State-of-the-art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 38–45.
4. Devlin, J., et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.