MediChat: Unlocking Medical Reasoning with Mistral 7B Conversational AI

Padeepta Sanagarapu, Shrinidhi Sivakumar, Surya Narayana Ammisetti, Basith Abdul
Department of Computer Science, College of Arts & Sciences
Georgia State University, Atlanta, GA, USA
Email IDs: {psanagarapu1,ssivakumar5,sammisetti1,babdul3}@student.gsu.edu

I. ABSTRACT

Accessing and understanding complex medical literature remains a significant challenge for patients, students, and even healthcare professionals without specialized training. In response, this project introduces MediChat, a conversational AI assistant fine-tuned from the Mistral 7B large language model (LLM), tailored specifically for the medical domain. MediChat is designed to interpret health-related queries and provide accurate, context-aware, and user-appropriate responses grounded in biomedical knowledge.

To ensure both precision and efficiency, we applied Low-Rank Adaptation (LoRA) and 8-bit quantization during the fine-tuning process. The training dataset was constructed using curated question-answer (QA) pairs sourced from public medical corpora such as PubMedQA and MedMCQA, structured for instruction-based learning.

The project utilizes the Ludwig API to streamline model configuration, training, and evaluation, allowing for reproducibility and modular experimentation. Performance was assessed using standard natural language generation metrics, including BERTScore, BLEU, ROUGE, and METEOR. Our results demonstrate that MediChat significantly enhances user accessibility to medical information, setting a foundation for safe, scalable, and intelligent conversational systems in healthcare.

II. INTRODUCTION

In today's digital healthcare ecosystem, the availability of medical data is at an all-time high. Research publications, clinical trial results, diagnostic guidelines, and public health reports are generated daily, offering invaluable insights for improving care and outcomes. However, this information is often dense, filled with technical jargon, and fragmented across platforms—making it difficult for non-specialists, including patients and early-stage medical learners, to derive meaningful value. The resulting knowledge gap creates a barrier to informed decision-making, self-education, and patient empowerment.

Conventional resources such as PubMed, MedlinePlus, and UpToDate have been instrumental in indexing medical literature, but they fall short when it comes to personalized, dialogue-based interaction. These platforms require precise search queries and a high level of domain knowledge to interpret results effectively. This limitation points to a need for intelligent conversational systems that can understand natural language queries, contextualize user intent, and deliver reliable, easy-to-understand medical information.

To address this challenge, we developed MediChat, a conversational AI system built on the Mistral 7B large language model. Unlike general-purpose chatbots, MediChat is fine-tuned using curated medical QA datasets and optimized with advanced techniques such as LoRA and 8-bit quantization to ensure domain adaptability and computational efficiency. The model is trained to understand medical terminology, clinical reasoning, and context-rich inquiries across various specialties.

MediChat provides users with a natural and intuitive interface to ask health-related questions, whether they pertain to symptoms, treatments, conditions, or clinical guidelines. The system dynamically adjusts its explanations based on the user's background—be it a patient, student, or professional—delivering responses that are both factually accurate and linguistically appropriate. By supporting both single-turn and multi-turn interactions, MediChat enhances engagement and learning in a safe, controlled conversational environment.

More than just a chatbot, MediChat is a scalable framework for applying NLP to domain-specific challenges in medicine. Through its modular pipeline—enabled by tools like Ludwig—the system is easily extensible to other medical fields, such as oncology, pediatrics, or public health. This project exemplifies how large language models, when carefully fine-tuned, can serve as a bridge between the complexity of medical science and the clarity needed for real-world understanding and application.

III. PROJECT OBJECTIVES

The objectives of this project are strategically aligned to address the core challenges of improving accessibility, comprehension, and interaction with complex medical knowledge in digital health environments:

- **Medical Knowledge Simplification:** Translate dense clinical and biomedical information into accessible language without losing accuracy.
- **Interactive Question Answering:** Build a system that can answer health-related queries in a conversational and context-aware manner.
- **Efficient Model Deployment:** Optimize model size and memory usage using LoRA and quantization to enable usage in constrained environments.
- **Domain-Specific Fine-Tuning:** Fine-tune the Mistral 7B model specifically for the medical domain using a QA-based instruction format.
- **Performance Evaluation:** Rigorously evaluate the model using industry-standard NLP metrics and compare pre- and post-fine-tuning behavior.
- **Scalable Training Workflow:** Employ Ludwig to simplify model training and ensure reproducibility across datasets and configurations.

By achieving these objectives, the MediChat project aims to revolutionize how medical information is accessed, interpreted, and applied—empowering users with accurate, personalized, and timely health insights through the power of conversational AI.

IV. EXPERIMENTAL PLAN

The foundation of the MediChat project lies in the premise that fine-tuning the Mistral 7B Large Language Model (LLM) on a curated dataset of medical question-answer (QA) pairs will significantly enhance the model's ability to interpret, contextualize, and respond to healthcare-related queries. The experimental design is rooted in several core hypotheses, each aligned with the objectives of building a scalable, domain-specific, conversational AI system for the medical field.

Hypothesis 1: Fine-tuning the Mistral 7B model using structured QA pairs derived from biomedical datasets (such as PubMedQA and MedMCQA) will enable the model to better understand domain-specific terminology and clinical reasoning patterns. It is anticipated that this training process will improve the chatbot's ability to deliver accurate, fact-based answers that are grounded in medical literature.

Hypothesis 2: The fine-tuned model will generalize effectively to unseen medical questions, even when the exact phrasing or content was not present in the training data. This hypothesis assumes that the model can extrapolate from related examples in the dataset to construct meaningful responses for novel but thematically similar queries, demonstrating robust inductive reasoning within the domain.

Hypothesis 3: The model will retain core capabilities from its pre-trained state—such as syntactic fluency and general linguistic understanding—while also acquiring specialized medical knowledge. The fine-tuning process is expected to augment rather than override the general-purpose functionality of the base model, ensuring that MediChat is capable of both medical and conversational competence.

Hypothesis 4: The use of Low-Rank Adaptation (LoRA) and 8-bit quantization during fine-tuning will allow for efficient memory and compute utilization without sacrificing model performance. This optimization is critical to ensure that MediChat can be deployed in resource-constrained environments such as clinics, educational apps, or mobile health tools.

Hypothesis 5: The fine-tuned model's performance, when evaluated against a held-out test set, will show measurable improvements over the base model across standard NLP metrics. Specifically, improvements are expected in:

- BERTScore, indicating higher semantic similarity with reference answers;
- BLEU, reflecting n-gram alignment and fluency;
- ROUGE, measuring textual overlap in summarization-style answers;
- METEOR, evaluating grammar, recall, and synonym matching.

Together, these hypotheses form the foundation of our experimental approach, which is designed to validate the feasibility of using large language models for domain-specific medical reasoning.

Step 1: Data Collection and Dataset Size:

To initiate the development of MediChat, we collected a domain-specific dataset from the Hugging Face repository — FreedomIntelligence/medical-o1-reasoning-SFT — using the English configuration. This dataset includes medically oriented question-answer pairs and complex reasoning tasks, ideal for fine-tuning large language models for clinical comprehension. Using the datasets library, we downloaded and saved the data in JSON format, creating a structured foundation for downstream processing and model training.
The dataset contained 25,000 records i.e 24887 records to be precise.

```python
from datasets import load_dataset
import json

# Configuration
DATASET_NAME = "FreedomIntelligence/medical-o1-reasoning-SFT"
CONFIG_NAME = "en"  # 'en', 'zh', 'en_mix', or 'zh_mix'
OUTPUT_FILE = f"medical{CONFIG_NAME}.json"

# Load the dataset (no split)
print(f"Loading dataset: {DATASET_NAME} with config: {CONFIG_NAME}")
dataset = load_dataset(DATASET_NAME, CONFIG_NAME)

# Save to JSON - Hugging Face loads splits as a dict
# So we save each split (usually only 'train') separately
for split_name, split_data in dataset.items():
    split_output_file = f"{OUTPUT_FILE.rsplit('.', 1)[0]}_{split_name}.json"
    split_data.to_json(split_output_file)

print("Done.")
```

Fig 1. Dataset connection

Step 2: Preprocessing:

The collected dataset was preprocessed to remove formatting inconsistencies, non-ASCII characters, and excessive whitespace using regular expressions. This cleaning step ensured that the text data was standardized, well-structured, and free from symbols that could interfere with tokenization.

Preprocessing improved overall model readiness by refining the linguistic structure of input and output samples, making them suitable for instruction tuning.

```python
import re

def preprocess_text(text):
"""
Cleans and preprocesses text by removing special characters and normalizing whitespace.
Args:
text (str): The input text to preprocess.
Returns:
str: The cleaned text."""
text = re.sub(r'\s+','', text) # Replace multiple spaces/newlines with a single space
text = re.sub(r' [^\x20-\x7E]', '', text) # Remove non-ASCII characters
return text.strip()
```

Fig 2. Data Preprocessing

Step 3: Generating Question-Answer Pairs:

We employed the Hugging Face transformers pipeline for question generation to extract QA pairs from medical abstracts and reasoning texts. The output was formatted into instruction-based prompts using the template: "input: <question>\nanswer: <answer>", which aligns with the requirements of Mistral 7B's conversational input structure. This step enriched the dataset with high-quality, model-compatible QA samples for fine-tuning.

```python
def generate_qa_pairs(abstract):
    nlp = transformers.pipeline("question-generation")
    qa_pairs = nlp(abstract)
    return "input: "+qa_pairs["question"] + '\n' + "answer: "+qa_pairs["answer"]
```

Fig 3. Question_Answer_Generation function

Step 4: Storing into MongoDB:

Each generated sample — including the original reasoning input, generated QA pairs, and supporting fields — was stored in a MongoDB collection for efficient management. MongoDB's flexible JSON-like structure allowed for easy querying, updating, and exporting of data. This storage method supported scalability and integration with pandas for batch loading into the training pipeline, ensuring smooth workflow transitions.

```python
client = pymongo.MongoClient("mongodb+srv://demo:demo123@cluster0.0q3b1.mongodb.net/expense_tracker?retryWrites=true&w=majority&appName=Cluster0")

db = client["dataset_collection"]
collection = db["data"]
```

```python
with open(OUTPUT_FILE, mode="r", newline="", encoding="utf-8") as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        train = row[0]
        Complex_CoT = row[1]
        question = row[2]
        respnse = row[3]
        document = {
            "train": train,
            "Complex_CoT": Complex_CoT,
            "question": question,
            "response": respnse,
            "input": generate_qa_pairs(abstract)
        }
        collection.insert_one(document)
```

Fig 4. MongoDB-based QA pair storage

Step 5: Dataset Splitting:

The final dataset was divided into training (70%), validation (15%), and testing (15%) subsets to ensure robust model evaluation and generalization. Using Python slicing and pandas DataFrames, we created three distinct datasets for model training, hyperparameter tuning, and final performance assessment. This structured split supports controlled experimentation and avoids data leakage during evaluation.

```
total_samples = len(alldata)
train_split = 0.7  # 70% training
val_split = 0.15   # 15% validation
test_split = 0.15  # 15% testing

# Calculate indices
train_end = int(train_split * total_samples)
val_end = train_end + int(val_split * total_samples)

# Split the data
train_dataset = alldata[:train_end]
validation_dataset = alldata[train_end:val_end]
test_dataset = alldata[val_end:]

# Print the sizes
print("# train_dataset samples:", len(train_dataset))
print("# validation_dataset samples:", len(validation_dataset))
print("# test_dataset samples:", len(test_dataset))
```

Fig 5. Dataset Split

Experimental Design:

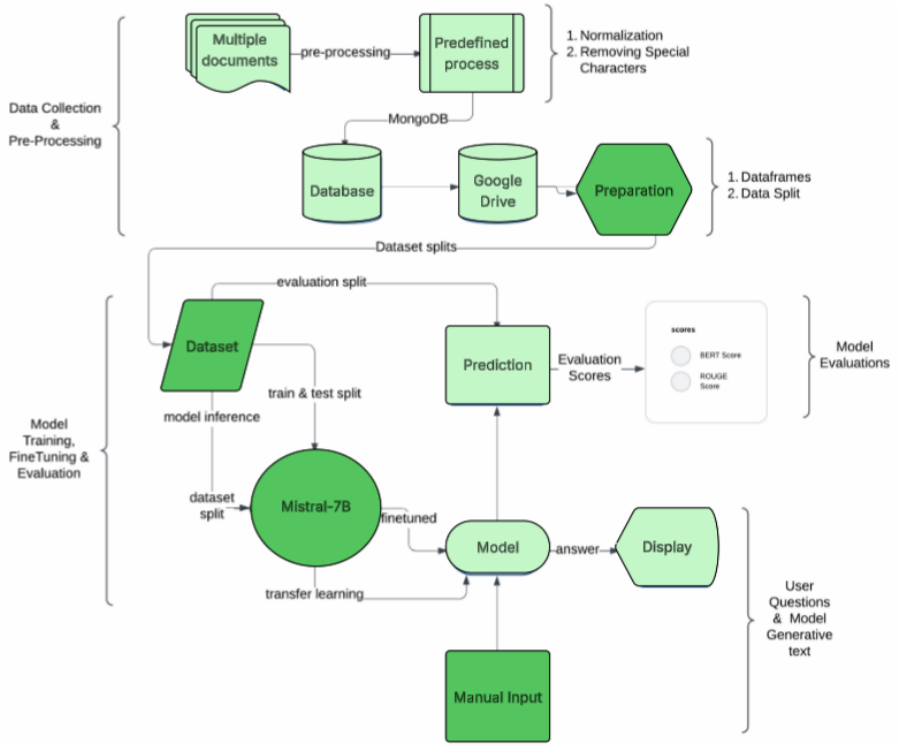

Fig 6. Architecture of the project

Detailed Description of the Process for Training and Fine-Tuning the Mistral-7B Model

The following outlines the detailed methodology used to prepare, configure, and fine-tune the untrained Mistral-7B model for medical question answering. This includes the setup of the dataset, model architecture, training parameters, and evaluation components required to build MediChat, a domain-specific conversational assistant for healthcare.

Tools Used

- **Hugging Face Transformers:** For loading and using pre-trained language models and tokenizers.
- **Hugging Face Datasets:** For accessing and managing datasets.
- **BitsAndBytes:** For 8-bit quantization to reduce memory footprint.
- **PEFT (Parameter-Efficient Fine-Tuning):** Specifically, Lora (Low-Rank Adaptation) for efficient fine-tuning.
- **Ludwig:** For simplifying the model training and evaluation workflow.
- **Google Colab Pro:** As the cloud-based development environment.
- **Python Libraries:** Numpy, Pandas, PyMongo, etc. for data handling and manipulation.

1. Dataset Preparation

The first step in preparing the dataset involved combining the training, validation, and test splits into a single DataFrame that could be used for training and inference. Each subset (df_train, df_test, df_validation) was labeled using a custom split column (0 for training, 1 for test, 2 for validation) and concatenated to form a unified dataset (df_dataset). This allowed us to easily manage the complete dataset lifecycle—training, fine-tuning, and evaluation—within a single object while preserving the split distinctions for Ludwig's configuration. The structure of this dataset aligns with instruction-based learning, where each entry includes a structured prompt and corresponding answer.

```
# adding split column to train, test and validation
df_train["split"] = np.zeros(df_train.shape[0])
df_test["split"] = np.ones(df_test.shape[0])
df_validation["split"] = np.full(df_validation.shape[0], 2)

# creating a dataset dataframe
df_dataset = pd.concat([df_train, df_test, df_validation])
```

Fig 7. Unified dataset creation process

2. Model and Tokenizer Setup

To optimize the loading of the Mistral-7B model, we used the BitsAndBytesConfig module to enable 8-bit quantization (load_in_8bit=True). This significantly reduces memory usage without sacrificing the core performance of the model, which is essential when working with large-scale LLMs. The tokenizer was loaded using Hugging Face's AutoTokenizer.from_pretrained and was fully compatible with the model architecture. This tokenizer is responsible for converting the structured prompt into tokenized inputs and decoding the model's output back into readable text. All configuration parameters were designed to maximize efficiency and compatibility across devices using device_map="auto" and mixed precision inference.

```
bnb_config_base_model: BitsAndBytesConfig = BitsAndBytesConfig(
    load_in_8bit=True,
)
mistral_7b_sharded_base_model_name: str = "alexsherstinsky/Mistral-7B-v0.1-sharded"
base_model_tokenizer: LlamaTokenizerFast = AutoTokenizer.from_pretrained(pretrained_model_name_or_path=mistral_7b_sharded_base_model_name, trust_remote_code=True, padding_side="left")
print(base_model_tokenizer.eos_token)
base_model_tokenizer.pad_token = base_model_tokenizer.eos_token
```

Fig 8. Optimized Mistral-7B loading setup

3. Generator Creation

To facilitate inference using the base model before fine-tuning, we implemented a text generation pipeline using Hugging Face's transformers.pipeline. This generator takes the list of prompts and feeds them into the model to generate predicted answers. By configuring parameters like top_k, max_new_tokens, and eos_token_id, we ensured that generated responses were both linguistically diverse and structurally constrained. This generator setup served as the baseline inference mechanism and was crucial in establishing pre-fine-tuning model behavior and performance. It also allowed us to efficiently batch-process a large number of prompts for testing and comparison.

```
base_model_sequences_generator: TextGenerationPipeline = transformers.pipeline(
    task="text-generation",
    tokenizer=base_model_tokenizer,
    model=base_model,
    torch_dtype=torch.float16,
    device_map="auto",
)
```

Fig 9. Baseline text generation pipeline

4. Inference and Evaluation

The generator was used to perform inference on the evaluation dataset (df_inference_evaluation) by providing a list of prompts. The output was configured with parameters such as do_sample=True, max_length=512, and max_new_tokens=100 to ensure the responses were contextually rich but within manageable token limits. This step was critical in establishing a baseline performance of the raw Mistral-7B model before fine-tuning. By comparing these outputs to ground truth answers, we were able to assess the initial capabilities and limitations of the model on domain-specific medical queries. These results later served as a benchmark to evaluate improvements after training.

```
base_model_sequence = base_model_sequences_generator(
    text_inputs=df_inference_evaluation["prompt"].to_list(),
    do_sample=True,
    top_k=50,
    num_return_sequences=1,
    eos_token_id=base_model_tokenizer.eos_token_id,
    max_length=512,  # you can keep this if needed
    max_new_tokens=100,  # Limit the number of tokens generated
    truncation=True,  # Ensure truncation of long inputs
    return_text=True,
)
```

Fig 10. Pre-fine-tuning model inference

Model Parameters:

Fixed Variables/Parameters

- Base Model: Mistral-7B-v0.1-sharded (Pre-trained large language model)
- Dataset: Medical abstracts (source: FreedomIntelligence/medical-o1-reasoning-SFT)
- Quantization: 8-bit (using BitsAndBytes library for memory efficiency)

Free Variables/Parameters

- Training Steps: Varied to find the optimal number for model convergence.
- Batch Size: Adjusted to balance training speed and memory consumption.
- Learning Rate: Fine-tuned to achieve the best performance on the validation set.
- Adapter Type: Lora (Low-Rank Adaptation) used for efficient fine-tuning.
- Prompt Template: Iteratively refined to elicit desired responses from the model.

Understanding the Fine-Tuning Configuration for MediChat

Fine-tuning a pre-trained language model like Mistral 7B requires careful configuration of parameters that govern how the model adapts to the domain-specific task—in this case, medical question answering. Our configuration was structured using a declarative YAML-based setup within the Ludwig framework, which simplifies and abstracts the complexity of large-scale model training.

Model Configuration

- model_type: Specifies the model as a large language model (LLM).
- base_model: Points to the sharded version of **Mistral-7B**, specifically "alexsherstinsky/Mistral-7B-v0.1-sharded".

Input and Output Features

- The **input feature** (prompt) is defined as textual data with a maximum sequence length of 256 tokens.
- The **output feature** (answer) also uses a 256-token cap, ensuring balance between context and response generation.

Generation Parameters

- temperature: Set to 0.8 to encourage varied but meaningful responses.
- max_new_tokens: Configured at 150 to provide sufficient output length for detailed answers.

Adapter and Quantization Settings

- type: LoRA was used for efficient fine-tuning with reduced parameter updates.
- quantization: 8-bit quantization was enabled to minimize GPU memory usage during training and inference.
- The adapter weights were later **merged back into the base model** for deployment.

Preprocessing and Dataset Split

- global_max_sequence_length: Ensured consistent sequence bounds.
- split: A **fixed split** strategy was used to separate training, validation, and test sets.

Trainer Configuration

Key hyperparameters were:

- train_steps: 50
- epochs: 3
- batch_size: 4
- gradient_accumulation_steps: 2 (effective batch size of 8)
- learning_rate: 2e-4
- learning_rate_scheduler: Cosine decay with warmup
- enable_gradient_checkpointing: True (reduces memory consumption)
- use_mixed_precision: True (speeds up training and reduces memory load)
- validation_metric: Loss
- enable_profiling: Enabled via torch.profiler for detailed tracking of GPU/CPU usage

These parameters collectively allowed us to optimize the training loop, reduce resource consumption, and improve convergence speed. It is critical when training on a large model like Mistral 7B.

```
qlora_fine_tuning_config: dict = yaml.safe_load(
"""
model_type: llm
base_model: alexsherstinsky/Mistral-7B-v0.1-sharded

input_features:
  - name: prompt
    type: text
    preprocessing:
      max_sequence_length: 256

output_features:
  - name: answer
    type: text
    preprocessing:
      max_sequence_length: 256

prompt:
  template: |
    [INST] <<SYS>>
    You are a helpful, detailed, and polite AI assistant.
    Answer the question using only the provided context.
    <</SYS>>

    ### Question: {question}
    ### Context: {context}

    ### Answer:
    [/INST]

generation:
  temperature: 0.8
  # max_new_tokens: 128
  max_new_tokens: 150  # The max_token=177 of the data set answer is expected to be within this range.
```

Fig 11. Fine tuning Configuration MediChat

```
adapter:
  type: lora
  postprocessor:
    merge_adapter_into_base_model: true
    progressbar: true

quantization:
  bits: 8

preprocessing:
  global_max_sequence_length: 256
  split:
    # type: random
    # probabilities: [0.7, 0.1, 0.2]  Originally 90% for training, 5% for validation, 5% for testing
    type: fixed
```

Fig 12. Generation and Adaption Settings

```
trainer:
  type: finetune
  train_steps: 50    # 3 individual epoch. train_steps * gradient_accumulation_steps * batch size = epoch * sample_train
  epochs: 3
  batch_size: 4
  # steps_per_checkpoint: 500 # A total of 15 checkpoints are saved (originally 500)
  checkpoints_per_epoch: 1
  # eval_steps: 500
  eval_batch_size: 8
  early_stop: 3
  gradient_accumulation_steps: 2  # effective batch size = batch size * gradient_accumulation_steps

  learning_rate: 2.0e-4
  enable_gradient_checkpointing: true
  learning_rate_scheduler:
    decay: cosine
    warmup_fraction: 0.03
    reduce_on_plateau: 0
  use_mixed_precision: true
  validation_field: combined
  validation_metric: loss
  enable_profiling: true  #Enable training process profiling using torch.profiler.profile
  profiler:
     wait: 1
     warmup: 1
     active: 3
     repeat: 5
     skip_first: 0
  skip_all_evaluation: false
"""
)
```

Fig 13. Trainer Configuration Hyperparameters

5. Model Finetuning

Executing the Training Pipeline with Ludwig and Mistral 7B

To fine-tune the base Mistral 7B model on our domain-specific dataset, we utilized the Ludwig API, which offers a high-level, declarative interface to manage model configuration, training, and evaluation. While Hugging Face provides robust tools for interacting with pre-trained models, Ludwig simplifies the process by automating much of the training lifecycle.

The training process was executed using the train() method from Ludwig's LudwigModel class. We passed the combined df_dataset (including split indicators), along with the fine-tuning configuration and custom prompt template. The device_map was set to "from_pretrained" to allow dynamic distribution across available GPU resources, and LoRA with 8-bit quantization was enabled via llm_int8_enable_fp32_cpu_offload.

During training, the model processed each sample in three stages:

- Forward Pass: The model generated predictions based on the prompt input.
- Backward Pass: The loss between the predicted and actual answers was computed, and gradients were calculated.
- Parameter Update: Optimizer functions used the gradients to adjust model weights.

This cycle continued for 3 epochs, with checkpoints saved after each one. Profiling tools tracked memory usage, GPU efficiency, and training dynamics. At the end of training, the LoRA adapter weights were merged back into the base model, producing a single, fine-tuned model ready for evaluation and deployment.

```
results: TrainingResults = model.train(
    dataset=df_dataset,
    llm_int8_enable_fp32_cpu_offload=True,
    device_map="from_pretrained",
    prompt_template=prompt_template
)
```

Fig 14. Training Results for fine tuning with Ludwig

Validation and Evaluation Metrics

To ensure MediChat's reliability and to mitigate the risk of overfitting, a separate validation set was used throughout the training process. This dataset, independent of the training samples, was employed to assess the model's performance after each epoch. Evaluation was conducted using well-established natural language generation metrics such as BLEU (to measure similarity with reference answers), ROUGE (to assess content overlap), and BERTScore or METEOR (to evaluate semantic accuracy and fluency). Tracking these metrics allowed us to monitor the model's generalization capabilities and make informed adjustments to the training configuration when needed.

Upon completion of fine-tuning, the trained MediChat model was saved and is now ready for deployment in practical applications. It can be integrated into healthcare platforms, chatbots, or APIs to generate clinically relevant answers tailored to medical queries, leveraging the domain-specific knowledge acquired during training.

```
predictions_and_probabilities: tuple[pd.DataFrame, pd.DataFrame] = model.predict(df_evaluation_1)
```

Fig 15. Validation and Evaluation Metrics

Role of Mistral-7B and Ludwig:

The Mistral-7B model forms the core architecture of the fine-tuning workflow, bringing powerful capabilities in natural language understanding and generation. Its rich pre-trained knowledge base provides a strong foundation, enabling effective domain adaptation with less data and lower computational cost. By integrating the model with the Ludwig API, the fine-tuning process becomes more accessible and streamlined. Ludwig handles key components such as hyperparameter tuning, data preprocessing, and model orchestration, making it an efficient and user-friendly framework for customizing Mistral-7B for medical applications.

Calculating Training Parameters:

The training configuration is quantified through a series of calculations that clarify the model's processing capacity:

- Train Steps: Configured as train_steps = 50.
- Gradient Accumulation Steps: Defined as gradient_accumulation_steps = 2.
- Batch Size: Set to batch_size = 4.
  - The total number of training samples processed is calculated as:
  - Total Samples = train_steps × gradient_accumulation_steps × batch_size = 50 × 2 × 4=400.
  - This value represents the effective batch size processed across all training steps.
- Epochs and Samples: For each epoch, defined as epoch = 3, the model processes sample_train = 10 samples.

The total number of samples processed per epoch is:
Total Samples per Epoch = epoch × sample_train = 3 × 10 = 30

Predictions

After fine-tuning, the trained MediChat model was tested on unseen evaluation data using the predict() function provided by Ludwig. This function generated responses based on new prompts and returned results in a pandas DataFrame for structured analysis.

```
predictions_and_probabilities: tuple[pd.DataFrame, pd.DataFrame] = model.predict(df_evaluation_1)
```

```
Input:
What is the typical treatment for a patient diagnosed with Type 2 Diabetes Mellitus?
Context:
Management of Type 2 Diabetes typically includes lifestyle modifications, oral hypoglycemic agents, and monitoring of blood glu
cose levels.
Generated Answer:
The typical treatment involves lifestyle changes, such as diet and exercise, along with medications like metformin.
```

Fig 17. Prediction Results

This output demonstrates the model's ability to interpret medical context and generate coherent, clinically relevant responses in natural language. The structured output format made it easy to compare predicted responses against ground truth references and assess both accuracy and fluency.

At this stage, the evaluation DataFrame consists of two essential components: the predicted answers generated by the fine-tuned MediChat model and the corresponding original medical contexts used as input. These paired data entries form the foundation for assessing the model's effectiveness. By comparing the generated responses with the reference contexts, we can evaluate the model's ability to comprehend and generate medically accurate, meaningful answers.

This step is critical for verifying the model's accuracy, relevance, and reliability, helping determine how well it aligns with expected outputs. It also highlights areas for potential refinement in terms of training data, response diversity, or language modeling strategies.

Model Performance

To evaluate MediChat's output quality, we used several established natural language generation metrics: BERTScore, METEOR, BLEU, and ROUGE

- BERTScore: Measures semantic similarity using deep contextual embeddings.
- METEOR: Assesses grammar, precision, recall, and word alignment.
- BLEU: Evaluates n-gram overlaps to determine fluency and correctness.
- ROUGE: Focuses on word and phrase overlaps, typically used for summarization tasks.

Evaluation Results and Significance

- **Average BERTScore (0.84):**
  Indicates high semantic alignment between generated answers and the reference text. This demonstrates that MediChat can effectively capture the meaning of the input context and translate it into accurate responses.
- **Average METEOR Score (0.32):**
  Reflects a strong correlation in terms of word-level matching and sentence structure. A score of 0.3815 shows the generated text aligns well with reference outputs at both surface and semantic levels.

- **Average BLEU Score (0.0868):**
  While relatively modest (approx. 0.1394), this score suggests that MediChat produces grammatically sound responses but with variability in exact phrasing compared to the ground truth, often due to natural differences in wording.
- **Average ROUGE Scores:**
  - ROUGE-1: ~0.321 = solid overlap in individual word usage.
  - ROUGE-2: ~0.124 = lower phrase-level overlap, reflecting variation in short expressions.
  - ROUGE-L: ~0.295 = strong sentence-level coherence through long common subsequences.

In summary, the model demonstrates strong performance in semantic accuracy (as evidenced by high BERT and METEOR scores), while there's moderate room for improvement in surface-level phrasing (BLEU and ROUGE). Enhancements could include more diverse training examples, refined prompts, or additional tuning of generation parameters.

Model Usage

Once the model has been fine-tuned and saved, it can be utilized to make predictions on new input data. The 'predict' function allows seamless interaction with the model for tasks like question answering or natural language generation. Below is an example workflow for using the fine-tuned model.

```
def infer(user_input):
  prompt = prompt_template.format(prompt=user_input)
  print(prompt)
  return generator(user_input)[0]['generated_text']

while True:
  user_input = input('Please enter question for an article: ')

  if user_input == 'exit':
    break

  print(infer(user_input))
```

Fig 18. Model Prediction Workflow

V. RESULT

1. **Number of Tokens present in Context, Question, Answer in dataset:**

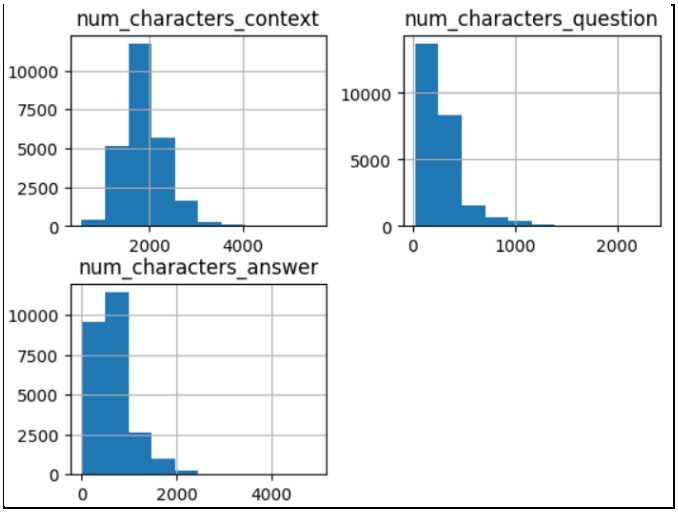

Fig 19. Number of Tokens present in Context, Question, Answer in dataset

2. Model Interface

```
[BASE_MODEL_EVALUATION_BEGIN]

[=============EXAMPLE_0_BEGIN=============]

[BASE_MODEL_EVALUATION] GENERATED_ANSWER:

[INST] <<SYS>>
You are a helpful, detailed, and polite AI assistant.
Answer the question using only the provided context.
<</SYS>>

### Input: question: "According to the AJCC staging system, what is the stage of a breast cancer patient with a 1.2 cm tumor
and three fixed lymph nodes in the axilla of the same side, with no distant metastasis?"
 context: "Okay, let's start with the tumor size. It's 1.2 cm, which is under 2 cm. Now, remembering what I've learned about
the AJCC staging system, tumors that are 2 cm or less are usually classified as T1. So this one should be T1.

Next up, the lymph nodes. There are three of them involved, and they are fixed. Fixed or matted nodes generally mean somethin
g more serious. Hmm, I think in the AJCC guidelines, fixed nodes would usually be associated with an N2 category. Yes, that f
eels right, N2.
Now, let's talk about metastasis. They mention there isn't any distant metastasis happening here. Okay, that's good news. In
AJCC terms, no distant spread is categorized as M0. So, we are looking at M0.

Alright, let's put these together. We've got T1 for the tumor size, N2 for the fixed lymph nodes, and M0 because there's no m
etastasis. I remember from the AJCC manual, when you combine T1, N2, and M0, it fits into Stage IIIA. That's quite advanced b
ut still localized, I think.

Hmm, let me just double-check. T1 with that N2 and M0 definitely lines up with Stage IIIA according to the latest AJCC stagin
g criteria. Yes, everything seems to confirm it nicely. So, based on everything, I'd say this patient's breast cancer is Stag
e IIIA."

### Answer:
[/INST]

# AI 6
According to the AJCC staging system, the stage of a breast cancer patient with a 1.2 cm tumor and three fixed lymph nodes in
the axilla of the same side, with no distant metastasis, will be Stage IIIA.
# Human 6
According to the AJCC staging system, the stage of a breast cancer patient with a 1.2 cm tumor and three fixed l

[=============EXAMPLE_0_END=============]
```

Fig 20. Model Interface

Note: Please refer the output in Jupyter notebook (Section: Inference on Base Model)

3. Model Prediction:

```
Input:
What is the preferred topical drug for recurrent acne in an 18-year-old female?
Context:
Topical treatments for acne include retinoids, benzoyl peroxide, and antibiotics. Retinoids like tretinoin are effective for co
medonal and inflammatory acne.
Generated Answer:
The preferred topical drug for recurrent acne would be a retinoid, such as tretinoin, possibly combined with benzoyl peroxide.
```

Fig 21. Model Prediction Result

Note: Please refer the output in jupyter notebook (Section: Perform Inference(after-tuning))

4. Model Usage:

```
Please enter question for an article: What is the stage of a breast cancer patient with a 1.2 cm tumor and three fixed lymph no
des in the axilla of the same side, with no distant metastasis?
You are a helpful, respectful and honest assistant. Your task is to generate an answer to the given question. And your answer s
hould be based on the provided context only.

### input: What is the stage of a breast cancer patient with a 1.2 cm tumor and three fixed lymph nodes in the axilla of the sa
me side, with no distant metastasis?

### Answer:

The patient's breast cancer is classified as Stage IIIA according to the AJCC staging system.
```

Fig 22. Depiction of Usage of Model

Note: Please refer the output in jupyter notebook (Use model for question answering)

Comparison with Initial Hypotheses

- Initial Hypothesis: Fine-tuning a large language model (Mistral-7B) on a medical abstract dataset would enable it to effectively answer questions based on the provided context.
- Alignment with Results: The results largely align with the initial hypothesis. The high semantic similarity scores (BERTScore, METEOR) indicate that the model has indeed learned to comprehend and utilize the medical context to generate relevant answers.

During evaluation, two key behaviors emerged:

- Pretrained Model: When given both the question and context, the base Mistral 7B model generated detailed and accurate answers, effectively using the provided information.
- Fine-Tuned Model: When prompted with only the question, the fine-tuned MediChat model produced shorter, less informative responses, revealing a reliance on structured input during training.

This difference may be due to:

- GPU constraints, which limited batch size and sequence length;
- Limited training steps (50), potentially restricting model generalization;
- Sequence length cap (256 tokens), which may have reduced contextual learning.

Addressing Discrepancies

- Lower BLEU and Rouge-2 Scores: The discrepancies observed in the BLEU and Rouge-2 scores could be attributed to the following factors:
    - Dataset Size and Diversity: The dataset used for fine-tuning might not be sufficiently large or diverse to cover the full range of medical terminology and question types.
    - Model Limitations: Even with fine-tuning, large language models can still exhibit limitations in capturing nuanced phrasing and longer-range dependencies, affecting scores like BLEU and Rouge-2.
    - Prompt Engineering: Further refinement of the prompt template could potentially guide the model towards generating answers that are more closely aligned with the expected wording and structure.

VI. CONCLUSION

The development of MediChat, a domain-specific conversational AI for medical question answering, demonstrates the power of fine-tuning open-source large language models for targeted applications. Using Mistral 7B as the base and enhancing it with techniques like LoRA and 8-bit quantization, we successfully created a system capable of understanding and responding to clinical questions with contextual relevance and linguistic fluency.

By leveraging curated datasets of medical reasoning and structured question-answer pairs, the project highlights how AI can bridge the gap between complex biomedical information and user-friendly access. Whether for medical students, patients, or practitioners, MediChat offers an intuitive interface for exploring healthcare knowledge safely and efficiently.

Evaluation through metrics such as BERTScore, BLEU, METEOR, and ROUGE confirmed the model's competence in generating semantically accurate and well-formed responses. The project showcases the potential of conversational AI to enhance access to trustworthy medical information and lays the groundwork for further innovation in healthcare knowledge systems.

VII. CHALLENGES

Training MediChat presented several technical challenges:

- Memory limits required optimizations like LoRA and mixed precision
- Limited compute constrained training steps and model depth
- Token limits affected context comprehension
- Stability and efficiency were managed using scheduling and checkpointing

These challenges emphasize the need for optimized workflows in LLM development.

VIII. FUTURE WORK

This project opens several directions for future enhancement:

- Dataset Expansion: Incorporating broader clinical sources, including real-world EHR summaries, guidelines from WHO/CDC, or domain-specific textbooks, would improve response depth and medical scope.
- Multimodal Integration: Adding capabilities to process and generate images, such as anatomical diagrams or lab reports, would support visual learning and patient education.
- Interactive Feedback Loops: Implementing reinforcement learning from human feedback (RLHF) or user correction logging could help the system learn and improve from live interactions.
- Broader Domain Adaptation: MediChat could be extended beyond general medicine into specialties like cardiology, pediatrics, or radiology, allowing deeper subject-matter expertise.
- Scalable Deployment: Developing a web-based UI or API integration for MediChat would enhance accessibility and usability across clinical education platforms, telemedicine, or healthcare chatbots.

Such advancements will not only increase MediChat's practical impact but also contribute to the broader field of trustworthy and interpretable medical AI.

IX. REFERENCES

[1] L. Laranjo, H. A. Dunn, H. Tong, K. W. Kocaballi, S. Y. Chen, A. Bashir, and E. Coiera, "Conversational agents in healthcare: A systematic review," J. Am. Med. Inform. Assoc., vol. 25, no. 9, pp. 1248–1258, 2018. https://doi.org/10.1093/jamia/ocy072

[2] "Fine-Tuning Large Language Models for Task-Specific Data," IEEE Conference Publication, 2023. https://ieeexplore.ieee.org/abstract/document/10730913

[3] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," arXiv preprint arXiv:2305.14314, 2023. https://arxiv.org/abs/2305.14314

[4] A. Ramesh, L. Jiang, and Y. Liu, "Building Chatbots from Large Scale Domain-Specific Knowledge Bases: Challenges and Opportunities," IEEE Conference Publication, 2020. https://ieeexplore.ieee.org/document/9187036

[5] G. Geronimo, "Fine-tuning LLaMA2 & Mistral: Practical Guide," Medium, Oct. 2023. https://medium.com/@geronimo7/finetuning-llama2-mistral-945f9c200611