ATTENTION-BASED DRUG Q&A SYSTEM

GROUP 8

GROUP MEMBERS

DEVANANDAN JS - 21MIM10041

AKASH M - 21MIM10027

JAY JOHRI - 22BAC10020

GAURAV SINGH - 22BAC10041

ANANY SHARMA - 22BAC10038

INDRONEEL DEBROY - 22BAC10039

PROBLEM STATEMENT

Healthcare professionals and patients need accurate, rapid drug information

Traditional drug information systems often struggle with:

- Handling natural language queries
- Understanding context-specific questions
- Processing complex pharmaceutical terminology
- Providing personalized responses based on query intent

SOLUTION OVERVIEW

This is the core AI technology of our system that selectively focuses on important parts of drug-related questions to generate accurate answers.

KEY CAPABILITIES:

1. Natural language understanding of drug-related queries
2. Precision in pharmaceutical entity recognition
3. Confidence Scoring for Response Reliability

SYSTEM ARCHITECTURE

Input Layer: Query processing and tokenization

Embedding Layer: Drug-specific word embeddings

Attention Mechanism: Focus on relevant pharmaceutical entities and relationships

Knowledge Integration: Connection to pharmaceutical database

Response Generation: Context-aware answer formulation

DATA SOURCES

Datasets we used -

Primary Dataset - **BioASQ Dataset** - dataset_url = "https://raw.githubusercontent.com/Andy-jqa/biomedical-qa-datasets/main/BioASQ/BioASQ-train-factoid-4b.json"

Backup Dataset - **DrugEHRQA Dataset** - "https://raw.githubusercontent.com/Andy-jqa/biomedical-qa-datasets/main/DrugEHRQA/sample_data/dev.json"

DATA TRAINING

```
[ ] # Cell 1: Setup and Dependencies
    import numpy as np
    import pandas as pd
    import tensorflow as tf
    import torch
    from torch.utils.data import Dataset, DataLoader
    from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipel
    from sklearn.model_selection import train_test_split
    import re
    import json
    import requests
    import os
    import matplotlib.pyplot as plt
    from tqdm.notebook import tqdm
    import nltk
    from google.colab import drive

    # Download necessary NLTK resources
    nltk.download('punkt')

    # Check if GPU is available
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Using device: cpu
```

```python
def download_biomedical_qa_dataset():
    print("Downloading dataset from GitHub repository...")

    # Create a directory for the dataset
    os.makedirs('data', exist_ok=True)

    # Try to download BioASQ dataset first (more focused on biomedical QA)
    dataset_url = "https://raw.githubusercontent.com/Andy-jqa/biomedical-qa-datasets/main/BioASQ/BioASQ-train-factoid-4b.json"

    try:
        # Download the dataset
        response = requests.get(dataset_url)

        if response.status_code == 200:
            # Save the dataset
            with open('data/bioasq.json', 'wb') as f:
                f.write(response.content)
            print("BioASQ dataset downloaded successfully!")

            # Load the dataset
            with open('data/bioasq.json', 'r') as f:
                data = json.load(f)

            return data, "bioasq"

        else:
            print(f"Failed to download BioASQ dataset. Status code: {response.status_code}")
            # Fall back to DrugEHRQA

    except Exception as e:
        print(f"Error downloading BioASQ dataset: {e}")
```

```python
# Cell 3: Download and Process Dataset

# Download and process dataset
data, data_type = download_biomedical_qa_dataset()

# Process the appropriate dataset type
if data_type == "bioasq":
    df = process_bioasq_dataset(data)
elif data_type == "drugqa":
    df = process_drugqa_dataset(data)
else:  # synthetic
    # Convert the synthetic format to DataFrame
    df = pd.DataFrame({
        'question': data['questions'],
        'context': data['contexts'],
        'answer': data['answers']
    })

# Display dataset info
print(f"\nDataset shape: {df.shape}")
print("\nSample data:")
print(df.head())

# Check for missing values and clean up
print("\nChecking for missing values:")
print(df.isnull().sum())

# Fill any NaN values
df = df.fillna("")

# Ensure answers are in the context
```

```python
# Cell 4: Dataset Class and Model Loading

# Load BioBERT model and tokenizer
print("\nLoading BioBERT model and tokenizer...")
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
model = AutoModelForQuestionAnswering.from_pretrained("dmis-lab/biobert-v1.1").to(device)

# Create a QA dataset class
class QuestionAnsweringDataset(Dataset):
    def __init__(self, questions, contexts, answers, tokenizer, max_length=384):
        self.questions = questions
        self.contexts = contexts
        self.answers = answers
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.questions)

    def __getitem__(self, idx):
        question = self.questions[idx]
        context = self.contexts[idx]
        answer = self.answers[idx]

        # Find the start and end positions of the answer in the context
        answer_start = context.find(answer)
        answer_end = answer_start + len(answer) - 1 if answer_start != -1 else -1

        # Tokenize
        encoding = self.tokenizer(
            question
```

```
Loading BioBERT model and tokenizer...
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your se
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  49.0/49.0 [00:00<00:00, 3.62kB/s]
config.json: 100%  462/462 [00:00<00:00, 38.0kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 4.10MB/s]
special_tokens_map.json: 100%  112/112 [00:00<00:00, 10.2kB/s]
pytorch_model.bin: 100%  433M/433M [00:02<00:00, 192MB/s]
Some weights of BertForQuestionAnswering were not initialized from the model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized: ['qa_outputs.bias', 'qa_outputs.weight
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
[ ] # Cell 5: Model Training

    # Set up training parameters
    learning_rate = 5e-5
    epochs = 3
    warmup_steps = 500
    weight_decay = 0.01

    # Set up optimizer and scheduler
    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
    total_steps = len(train_dataloader) * epochs
    scheduler = torch.optim.lr_scheduler.LinearLR(optimizer, start_factor=1.0, end_factor=0.0, total_iters=total_steps)
```

```
Starting training...
Epoch 1/3
Training epoch 1: 100%|▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰| 74/74 [43:34<00:00, 30.18s/it, loss=0.000587]
model.safetensors: 100%|▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰| 433M/433M [00:03<00:00, 141MB/s]
Average training loss: 0.6474
Epoch 2/3
Training epoch 2: 100%|▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰| 74/74 [42:01<00:00, 30.49s/it, loss=0.000295]
Average training loss: 0.0014
Epoch 3/3
Training epoch 3: 100%|▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰| 74/74 [42:05<00:00, 30.58s/it, loss=0.000288]
Average training loss: 0.0003
Model saved to ./biobert_drug_qa
```

[ ]

```python
# Cell 6: Model Evaluation

# Evaluation
print("\nEvaluating model on test set...")
model.eval()
exact_matches = 0
f1_scores = []

def compute_f1(prediction, ground_truth):
    """Calculate F1 score between prediction and ground truth"""
    prediction_tokens = prediction.lower().split()
    ground_truth_tokens = ground_truth.lower().split()

    common = set(prediction_tokens) & set(ground_truth_tokens)

    if len(common) == 0:
        return 0
```

```
Evaluating model on test set...
Evaluating: 100% ▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰ 19/19 [03:11<00:00,  8.82s/it]
Exact Match: 96.62%
F1 Score: 98.06%
```

```
[ ] # Cell 7: Sample Question Testing

    # Create a question answering pipeline with our fine-tuned model
    nlp = pipeline("question-answering", model=model, tokenizer=tokenizer, device=0 if torch.cuda.is_available() else -1)

    # Test with some drug-related questions
    test_contexts = [
        """Aspirin is a nonsteroidal anti-inflammatory drug (NSAID) used to reduce pain, fever, and inflammation.
        Common side effects include stomach irritation, nausea, vomiting, and heartburn. It should be taken with food
        to minimize gastrointestinal side effects. The typical dosage for adults is 325-650 mg every 4 hours as needed.""",

        """Metformin is an oral diabetes medicine that helps control blood sugar levels. It is used together with diet
        and exercise to improve blood sugar control in adults with type 2 diabetes. Common side effects include diarrhea,
        nausea, and stomach upset. The recommended starting dose is 500 mg twice daily with meals.""",

        """Antibiotics are medicines that fight bacterial infections in people and animals. They work by killing the
        bacteria or by making it hard for the bacteria to grow and multiply. Antibiotics only work against bacteria,
        not viruses. They're ineffective against viral infections like the common cold, flu, most sore throats,
        bronchitis, and many sinus and ear infections."""
    ]

    test_questions = [
        "What are the side effects of Aspirin?",
        "How should I take Metformin?",
        "Can antibiotics treat viral infections?"
```

```
Device set to use cpu

Testing the model with sample questions:
Q: What are the side effects of Aspirin?
A: stomach irritation, nausea, vomiting, and heartburn
Score: 0.3162

Q: How should I take Metformin?
A: diarrhea,
    nausea, and stomach upset
Score: 0.2888

Q: Can antibiotics treat viral infections?
A: ear infections
Score: 0.0297
```

```
[ ] # Cell 8: Interactive Drug QA Function

    # Function to get answer for a user question
    def ask_drug_question(question, context):
        """Function to get answers from the BioBERT model"""
        # Make sure we're in evaluation mode
        model.eval()

        # Tokenize the input
        inputs = tokenizer(
            question,
            context,
            return_tensors="pt",
            max_length=384,
            truncation="only_second",
            padding="max_length"
```

CURRENT LIMITATIONS

1. Limited Coverage of Rare Medications and Special Populations;

Specialized populations like pregnant women, nursing mothers, and pediatric patients

Extremely elderly patients (>85 years) with multiple comorbidities

2. Reduced Performance with Complex Multi-Drug Interaction;

While the model excels at two-drug interactions, accuracy decreases as query complexity increases

- The attention mechanisms struggle to track multiple interrelated pharmaceutical entities simultaneously

3 .Imbalanced Drug Coverage

Dataset contains much more information about common medications than rare ones

4. Query Complexity Distribution

The training data probably has more simple queries than complex ones

This natural distribution makes the model better at straightforward questions than edge cases

5. Temporal Information Gaps

Older medications have longer documentation histories than newer drugs

Information about recent approvals may be limited or missing entirely

FUTURE WORKS

Expanding the Knowledge Base to Specialized Therapeutic Areas

Incorporating comprehensive information on orphan drugs and ultra-rare disease treatments

Adding specialized dosing guidelines for conditions with limited patient populations

Enhancing pediatric pharmacology information across age ranges (neonatal, infant, child, adolescent)

Incorporating Multimodal Inputs

1. Visual Medication Identification ---

Adding capability to process images of pills, capsules, and other dosage forms

Enabling identification of medications by appearance characteristics

2. Package and Label Recognition---

Processing photos of medication packaging and prescription labels

Extracting key information like dosage, frequency, and warnings

Supporting queries combining visual and textual information

Real-Time Drug Database Updates

Automating the capture of new FDA approvals and label changes

Tracking drug pricing changes to support cost-effective recommendations

THANK YOU