INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

AIML B23 - PGCP

Explainability and Interpretability of Large Language Models(RAG)

Group -27

Deepak Tiwari
Bhoopendra Dwivedi
Gaurav Katyal
Ashish Verma

Project Mentor: Hiranmai Sri

Project Supervisor: Manish Shrivastava

Problem Statement

RAG models have demonstrated high performance, they often suffer from a lack of **transparency** and **explainability**. Users and stakeholders may struggle to understand why a model generates a certain response or what specific pieces of information influenced the generated output. This lack of explainability can lead to issues like reduced trust, difficulty in debugging, and challenges in verifying the accuracy of the model's answers.

Data Wrangling

Dataset Description

To prepare the data for the QA Information Retrieval System, we used **SQUAD** dataset.

The squad dataset file contained the following information about each ultrasound probe:

- Title
- Context
- Question
- Answer

Processing Tabular Data

	id	title	context	question	answers
0	5733be284776f41900661182	University_of_Notre_Dame	Architecturally, the school has a Catholic cha...	To whom did the Virgin Mary allegedly appear i...	{'text': ['Saint Bernadette Soubirous'], 'answ...
1	5733be284776f4190066117f	University_of_Notre_Dame	Architecturally, the school has a Catholic cha...	What is in front of the Notre Dame Main Building?	{'text': ['a copper statue of Christ'], 'answe...
2	5733be284776f41900661180	University_of_Notre_Dame	Architecturally, the school has a Catholic cha...	The Basilica of the Sacred heart at Notre Dame...	{'text': ['the Main Building'], 'answer_start'...
3	5733be284776f41900661181	University_of_Notre_Dame	Architecturally, the school has a Catholic cha...	What is the Grotto at Notre Dame?	{'text': ['a Marian place of prayer and reflec...
4	5733be284776f4190066117e	University_of_Notre_Dame	Architecturally, the school has a Catholic cha...	What sits on top of the Main Building at Notre...	{'text': ['a golden statue of the Virgin Mary'...

To process the tabular data into a format suitable for the QA system, each row, column, and cell was transcribed into human-readable sentences containing the key information. Custom Python functions were applied to each row of the probes and systems DataFrames, resulting in multiple documents for each probe/system. Each document provided a sentence of information that

could be extrapolated from the original data, enhancing compatibility with natural language processing techniques and improving readability for the QA system.

Data Preprocessing

Created custom function to preprocess data so that we can get relevant output. The goal was to create questions that could be answered based on the information provided in each document, improving the system's robustness and ability to handle a wide range of user queries.

```
train_df.isnull().sum()
train_df = train_df.dropna()
print("After dropping null values:", train_df.shape)
train_df = train_df.drop_duplicates()
print("After dropping duplicates:", train_df.shape)
train_df = train_df[["question", "context", "answer_text"]]
train_df.head(5)
```

```
After dropping null values: (87599, 5)
After dropping duplicates: (87599, 5)
```

	question	context	answer_text
0	To whom did the Virgin Mary allegedly appear i...	Architecturally, the school has a Catholic cha...	Saint Bernadette Soubirous
1	What is in front of the Notre Dame Main Building?	Architecturally, the school has a Catholic cha...	a copper statue of Christ
2	The Basilica of the Sacred heart at Notre Dame...	Architecturally, the school has a Catholic cha...	the Main Building
3	What is the Grotto at Notre Dame?	Architecturally, the school has a Catholic cha...	a Marian place of prayer and reflection
4	What sits on top of the Main Building at Notre...	Architecturally, the school has a Catholic cha...	a golden statue of the Virgin Mary

```
def processField(data):
    data_str = data.replace("array", "np.array").replace("dtype=object", "").replace("dtype=int32", "")
    data = eval(data_str)
    return data['text'][0]
```

+ Code + Markdown

```
train_df["answer_text"] = train_df["answers"].apply(processField)
train_df.drop(columns=["answers"], inplace=True)
```

To streamline the process of extracting the questions from the model's output, a template was incorporated into the prompt, instructing the model to respond solely with the questions structured.

Prompt snippet used for question generation:

```
def generate_rag_answer(question, context, max_length=200):
    # Format input as per T5's requirement
    input_text = f"""question: {question} Answer the question as precise as possible using the provided context. If the answer is
                not contained in the context, say 'answer not available in context'  context: {context}"""
```

A regular expression function was created to look for a sequence of a number followed by a period, whitespace, and then capture the text until the first question mark.

This allowed us to parse the outputs, map each question onto its corresponding document and tags, and create a training dataset containing question-document pairs.

Split into Train, Test

```
train_data, test_data = train_test_split(train_df, test_size = 0.2, random_state=42)
```

```
# Load the T5 model & tokenizer
model_name = "t5-small"  # You can try "t5-base" for better results
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Move model to GPU if available
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
```

Tokenize Inputs

```
# Function to preprocess data to create tokens
def preprocess_function(examples):
    inputs = [f"question: {q}  context: {c}" for q, c in zip(examples["question"], examples["context"])]
    targets = examples["answer_text"]

    # Tokenize inputs
    model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding="max_length")
    labels = tokenizer(targets, max_length=128, truncation=True, padding="max_length")

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

+ Code + Markdown

```
train_dataset = Dataset.from_pandas(train_data)
test_dataset = Dataset.from_pandas(test_data)
train_dataset = train_dataset.map(preprocess_function, batched=True)
test_dataset = test_dataset.map(preprocess_function, batched=True)
```

```
Map: 100% 700/700 [00:00<00:00, 772.43 examples/s]
Map: 100% 175/175 [00:00<00:00, 733.87 examples/s]
```

Training and Validation Loss for Each Epoch

```
warnings.warn(
[17520/17520 2:26:10, Epoch 4/4]
```

Epoch	Training Loss	Validation Loss
1	0.019400	0.015012
2	0.016600	0.014593
3	0.012900	0.014520
4	0.010000	0.014860

Use FAISS For Storing

- **Index Creation**: Initialize the FAISS index with the appropriate configuration.
- **Data Preprocessing**: Convert text or other data into vector embeddings using a model like **Sentence Transformers** or **OpenAI embeddings**.
- **Storing Data**: Insert the vector embeddings into the FAISS index.
- **Retrieving Data**: Search for the most similar vectors based on query embeddings.

Filtering Generated Questions for Relevance

To ensure the quality of the generated questions, we calculated scores for 1000 Validation Records.

- F1 Score, Recall, Precision
- Rouge, Blue Score
- Cosine Similarity

Below are the results:

```
# Evaluation
scorer = rouge_scorer.RougeScorer(["rouge1", "rougeL"], use_stemmer=True)

bleu_scores, rouge_scores = [], []

for _, row in test_df.head(1000).iterrows():
    pred_answer, relevant_docs = generate_answer_with_explanation(row["question"])

    # Ensure correct variable names
    #pred_answer = row["answer_text"]  # Rename to match later usage

    # BLEU Score
    bleu = sentence_bleu([row["answer_text"] .split()], pred_answer.split())
    bleu_scores.append(bleu)

    # ROUGE Score
    rouge = scorer.score(row["answer_text"], pred_answer)
    rouge_scores.append(rouge)
```

```
# Compute average scores
print(f"Average BLEU Score: {sum(bleu_scores) / len(bleu_scores):.4f}")
print(f"Average ROUGE Score: {sum([r['rouge1'].fmeasure for r in rouge_scores]) / len(rouge_scores):.4f}")
```

```
Average BLEU Score: 0.4515
Average ROUGE Score: 0.4704
```

```
cosine_scores = []
for _, row in test_df.head(1000).iterrows():
    pred_answer, relevant_docs = generate_answer_with_explanation(row["question"])

    # Compute Cosine Similarity
    cosine_sim = compute_cosine_similarity(row["answer_text"], pred_answer)
    cosine_scores.append(cosine_sim)

# Compute average cosine similarity score
print(f"Average Cosine Similarity: {sum(cosine_scores) / len(cosine_scores):.4f}")
```

```
Average Cosine Similarity: 0.6669
```

The final fine-tuning dataset comprised **queries** and **documents** in the corpus, serving as the foundation for training and evaluating the QA Information Retrieval System.

Exploratory Data Analysis

Evaluation of Pre-trained Embedding Models

Building the question-answering system, the focus was on selecting an appropriate embedding model. Embedding models convert text data into numerical representations, enabling the system to understand and compare the semantic meaning of words and phrases. The goal was to identify the best-performing out-of-the-box model, which would then be fine-tuned with the query-document dataset.

Few pre-trained models from Hugging Face were evaluated: all-mpnet-base-v2, all-MiniLM-L6-v2', all-multi-qa-mpnet-base-dot-v1, and multi-qa-distilbert-cos-v1. all-mpnet-base-v2 is a
general-purpose model, while the other two were specifically trained on a question-answer dataset of 215 million pairs. *multi-qa-distilbert-cos-v1* is based on the DistilBERT architecture, which uses knowledge distillation to create a smaller, faster model.

Fine-Tuning Process

Creating Datasets

Before splitting the data, a custom QA dataset class was defined, which inherits from the PyTorch dataset class. This class takes a list of input example objects, where each example consists of the query text and the content of the relevant document.

Three datasets were created for training, validation, and testing (70/30) by iterating over the respective query IDs and constructing input example objects with the query text and relevant document content.

Optimization for Hyperparameter Tuning

Optimization was employed to find the optimal hyperparameters for fine-tuning the model. This approach aims to maximize the Mean Reciprocal Rank (MRR) metric on the validation set by intelligently searching the hyperparameter space.

The search space for the Bayesian optimization was defined as follows:

- output_dir="./t5_squad",
- evaluation_strategy="epoch",
- save_strategy="epoch",
- learning_rate=3e-4, # Adjust based on performance
- per_device_train_batch_size=8,
- per_device_eval_batch_size=8,
- num_train_epochs=4,
- weight_decay=0.1,
- logging_dir="./logs",
- logging_steps=50,
- fp16=True, # Mixed precision for speedup
- load_best_model_at_end=True

The Optimization library was used to perform the optimization, running for 4 iterations. The best hyperparameters found were:

- per_device _batch_size: 8
- weight_decay: 0.1
- learning_rate: 3.6e-5
- warmup_steps: 106
- num_train_epochs: 4

Training the Final Model

With the best hyperparameters identified through Bayesian optimization, the next step was to train the final model using these optimal settings. DataLoader objects were created for the training, validation, and test datasets with the best batch size hyperparameter of 56. Several performance metrics were monitored throughout the 4 training epochs.

Loss Function

We utilized the loss function, which is well-suited for scenarios where we have only positive pairs of data, such as (query, relevant document) pairs. The loss function operates as follows:

1. Compute the cosine similarity scores between the query and document embeddings for each (query, document) pair in the batch.
2. Compute Rouge, BLUE Score.
3. Compute F1 Score, Precision, Recall.

By minimizing this loss function during training, the model learns to rank the relevant documents higher than the irrelevant ones for a given query.

Evaluation Metrics

During the training process, we monitored several performance metrics to evaluate the effectiveness and progress of our model. These metrics were extracted from the training logs and visualized over the epochs. We tracked these metrics at different epoch values to understand how well the model performs at various levels:

- **Training Loss @epoch:** The percentage of queries for which the relevant document is among the top-k retrieved documents.
- **Validation Loss @epoch:** The percentage of relevant documents that were successfully retrieved among results.

```
warnings.warn(
```

[17520/17520 2:26:10, Epoch 4/4]

Epoch	Training Loss	Validation Loss
1	0.019400	0.015012
2	0.016600	0.014593
3	0.012900	0.014520
4	0.010000	0.014860

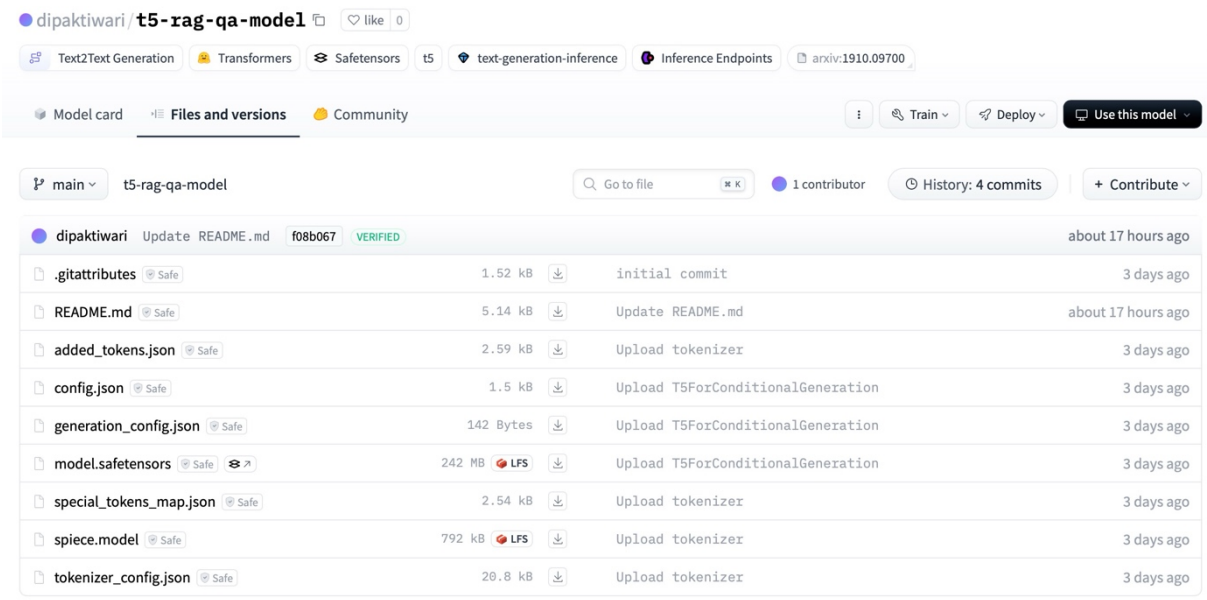
Model Evaluation and Comparison

After training the final model with the best hyperparameters, its performance was evaluated using a holdout test set and compared against three other models: the pre-trained *multi-qa-distilbert-cos-v1*, *multi-qa-mpnet-base-dot-v1*, and *all-mpnet-base-v2*. The performance of the four models along various performance metrics are displayed below:

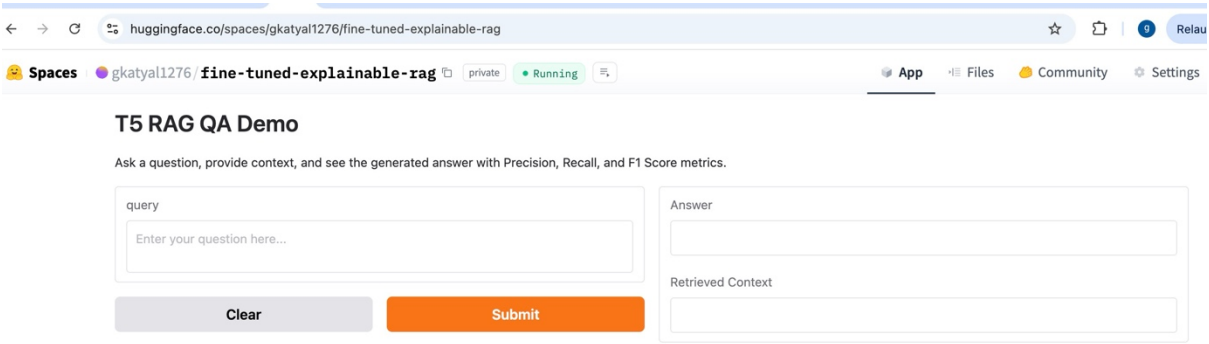
In all the performance metrics, the fine-tuned T5 (Text-to-Text Transfer Transformer) model consistently scored the highest, followed by the pre-trained base model (multi-qa-distilbert-cos-v1), then the multi-qa-mpnet-base-dot-v1 model, and finally the all-mpnet-base-v2 model.

Deployment

Model Deployment



Gradio Deployment



Conclusion

In this project, we successfully developed a Retrieval-Augmented Generation (RAG) Question-Answering (QA) model using the **T5 transformer** trained on the **SQUAD dataset**. Our approach involved data preprocessing, model fine-tuning, and evaluation using **F1-score** and **cosine similarity** to measure answer accuracy.

Key takeaways from the implementation:

- **Effective Training & Fine-tuning:** The model was trained using an optimized learning rate and hyperparameters to improve performance on factual QA tasks.

- **Evaluation Metrics:** The use of F1-score and cosine similarity helped quantify the accuracy and relevance of generated answers.

- **Model Deployment:** The trained model was successfully uploaded to **Hugging Face**, making it accessible for real-world applications.

Future improvements could involve integrating larger transformer models (such as **T5-Base** or **T5-Large**) for improved accuracy, leveraging better retrieval mechanisms, and applying the model to broader datasets beyond SQUAD to enhance its generalizability.

This RAG-based QA system showcases the potential of transformer models in open-domain question answering, paving the way for more robust and intelligent retrieval-augmented NLP solutions.

Limitations and Future Directions

Limitations

Dependence on Contextual Input:

- The model heavily relies on the provided context. If the relevant information is missing or ambiguous, it struggles to generate accurate answers.

Lack of Real-Time Knowledge:

- The model is static after training, meaning it does not update itself with new information unless retrained or fine-tuned again.

Computational Cost:

- Fine-tuning and inference on large datasets require substantial GPU resources, making it less accessible for real-time deployment on edge devices.

Future Direction

Fine-Tuning with Diverse Datasets

- Extend training beyond SQuAD by incorporating datasets like Natural Questions, TriviaQA, or HotpotQA to improve generalization.

Few-shot & Zero-shot Learning

- Leverage prompting techniques (e.g., **GPT-style few-shot learning**) to improve adaptability across domains **without retraining**.

Longer-Context Handling (Scaling with T5-XL or LLaMA)

- Utilize models like **LongT5 or GPT-4 Turbo** to handle **multi-paragraph contexts** for complex reasoning tasks.