INDIGO CASE STUDY: DEVELOPMENT AND EVALUATION OF STATE-OF-THE-ART QUESTION-ANSWERING MODELS

Github Link: https://github.com/StarBreaker20/QA-Models-Quora-Dataset

❖ Introduction

In the rapidly evolving field of Natural Language Processing (NLP), question-answering systems have emerged as a pivotal application, demonstrating the remarkable capability of machines to comprehend and generate human-like responses. This case study aims to develop a state-of-the-art question-answering model using the Quora Question Answer Dataset, which provides an ideal foundation for training and evaluating models to generate precise and contextually relevant answers to user queries. The escalating demand for intelligent systems that can interact with humans in natural language has spurred significant advancements in NLP models. This project leverages three prominent models: BERT, T5, and GPT-2, each representing the forefront of their respective architectures. By fine-tuning these models on the Quora dataset, we aim to compare their performances and identify the most effective approach for this task.

The Quora Question Answer Dataset is a rich and diverse dataset, containing pairs of questions and answers that mimic real-world scenarios where users seek detailed and accurate information. This dataset's breadth and depth make it suitable for developing and evaluating sophisticated question-answering models. The project's primary objective is to harness the strengths of BERT, T5, and GPT-2 to build models capable of understanding the nuances of user queries and providing accurate responses.

Data Exploration and Preprocessing

Data exploration is the initial step in understanding the dataset's structure and content. It involves analyzing the distribution of questions and answers, checking for missing values, and understanding the various categories and types of questions present. This step is crucial as it helps in identifying any anomalies or biases in the data that could impact the model's performance.

Once the data exploration is complete, the next step is preprocessing. Preprocessing involves cleaning the data to remove noise and ensure consistency. This includes removing special characters, converting text to lowercase, and eliminating stop words. For this project, we also performed tokenization, which is the process of breaking down text into smaller units called tokens. Additionally, lemmatization was applied to reduce words to their base or root form, ensuring that words with similar meanings are treated the same. This step is essential for improving the model's ability to understand and process natural language.

Model Training

BERT (Bidirectional Encoder Representations from Transformers)

BERT, developed by Google, is a groundbreaking model that has set new benchmarks in various NLP tasks. It is designed to understand the context of a word in search queries. Unlike traditional models that process text in a unidirectional manner, BERT processes text bidirectionally, meaning it considers the entire sentence's context to understand each word's meaning. For this project, we fine-tuned BERT on the Quora dataset to enhance its ability to generate relevant answers to user queries.

Fine-tuning BERT involves adjusting the pre-trained model on a specific task dataset, allowing it to adapt to the nuances of the new data. This process typically involves training the model for a few epochs with a lower learning rate to prevent overfitting.

T5 (Text-To-Text Transfer Transformer)

T5, developed by Google Research, is a versatile model that converts all NLP tasks into a text-to-text format. This means that both the input and output are text strings, allowing T5 to be applied to a wide range of NLP tasks, including translation, summarization, and question answering. For this project, we fine-tuned T5 to generate answers based on the context provided by the Quora dataset questions.

Fine-tuning T5 involves training the model to generate the most accurate and contextually appropriate responses to user queries. This requires extensive training data and computational resources, but the results are often superior due to the model's ability to understand and generate complex language structures.

GPT-2 (Generative Pre-trained Transformer 2)

GPT-2, developed by OpenAI, is a generative model designed to produce human-like text based on the input it receives. It uses a transformer-based architecture similar to BERT but focuses on text generation rather than understanding. GPT-2 is capable of generating coherent and contextually relevant text, making it suitable for tasks like question answering. For this project, we fine-tuned GPT-2 on the Quora dataset to improve its ability to generate accurate answers to user queries.

Fine-tuning GPT-2 involves training the model on the specific dataset to adapt its generative capabilities to the nuances of the Quora questions and answers. This process allows the model to generate responses that are not only accurate but also contextually relevant and human-like.

Evaluation Metrics

To rigorously assess the performance of each model, we employed a variety of metrics:

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** ROUGE measures the overlap between the generated text and reference text. It is commonly used for evaluating summarization and translation models. ROUGE scores include ROUGE-N

(measures N-gram overlap), ROUGE-L (measures the longest common subsequence), and ROUGE-S (measures skip-bigram overlap).

- **BLEU (Bilingual Evaluation Understudy):** BLEU measures the precision of the generated text by comparing it to one or more reference texts. It is widely used for evaluating machine translation models. BLEU scores range from 0 to 1, with higher scores indicating better performance.
- **METEOR (Metric for Evaluation of Translation with Explicit ORdering):** METEOR evaluates the generated text by considering synonyms, stemming, and word order. It is designed to address some of the limitations of BLEU and is often used in conjunction with other metrics for a more comprehensive evaluation.

❖ Detailed Explanation of Toolkits, Models, Processes, and Concepts Used in the Project

In this project, we undertook a comprehensive analysis, preprocessing, model training, and evaluation process for question-answering using the Quora Question Answer dataset. Here, we will delve into the detailed description of each toolkit, model, and process we used, providing definitions and explanations for each main term and process.

1. Toolkits and Libraries

1.1 Pandas

Pandas is a powerful, open-source data manipulation and analysis library for Python. It provides data structures and functions needed to manipulate structured data seamlessly. The primary data structure in pandas is the DataFrame, which is essentially a table of data with rows and columns.

1.2 NumPy

NumPy is a fundamental package for scientific computing with Python. It provides support for arrays, matrices, and many mathematical functions that operate on these data structures. NumPy is extensively used for performing numerical operations and is a core dependency for pandas and other data analysis libraries.

1.3 Matplotlib and Seaborn

Matplotlib is a plotting library for Python and its numerical mathematics extension, NumPy. It provides an object-oriented API for embedding plots into applications. Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

1.4 Plotly

Plotly is a graphing library that makes interactive, publication-quality graphs online. It can be used to create a wide range of charts and plots, including scatter plots, line plots, bar charts, box plots, histograms, and more.

1.5 Scikit-learn

Scikit-learn is a machine learning library for Python. It provides simple and efficient tools for data mining and data analysis, built on NumPy, SciPy, and Matplotlib. It includes various classification, regression, and clustering algorithms.

1.6 NLTK

The Natural Language Toolkit (NLTK) is a platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources, such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and more.

1.7 Transformers

Transformers is an open-source library developed by Hugging Face that provides state-of-the-art machine learning models for natural language processing (NLP) tasks. It supports a wide range of transformer-based models like BERT, GPT-2, and T5.

1.8 Datasets

The `datasets` library by Hugging Face provides a unified interface for accessing and working with a wide variety of datasets in NLP. It includes functionalities for dataset loading, preprocessing, and transformation, supporting efficient memory use and computation.

1.9 Evaluate

`evaluate` is a library for model evaluation, providing metrics implementations for assessing the performance of machine learning models. It supports a variety of metrics such as accuracy, precision, recall, F1-score, BLEU, ROUGE, and METEOR.

2. Concepts and Processes

2.1 Data Loading and Exploration

Data loading involves reading the dataset from a source and storing it in a format suitable for analysis. Exploration involves understanding the structure, contents, and quality of the data through summary statistics, visualizations, and basic queries.

2.2 Data Preprocessing

Data preprocessing involves cleaning and transforming raw data into a format suitable for analysis and model training. This includes handling missing values, removing duplicates, normalizing text, tokenization, stemming, lemmatization, and removing stop words.

- **Tokenization**: The process of breaking down text into smaller units, typically words or subwords.
- **Stemming**: The process of reducing words to their root form (e.g., running -> run).
- **Lemmatization**: The process of reducing words to their base or dictionary form (e.g., running -> run).

2.3 Train-Test Split

Splitting the dataset into training, validation, and test sets is crucial for model evaluation. The training set is used to train the model, the validation set is used to tune hyperparameters and evaluate model performance during training, and the test set is used to assess the model's performance on unseen data.

2.4 Dataset Conversion

Converting pandas DataFrames to Hugging Face `Dataset` and `DatasetDict` formats allows for seamless integration with the `transformers` library, enabling efficient data handling and processing.

2.5 Tokenization for Transformers

Transformers-based models require text to be tokenized into a specific format. This involves converting text into token IDs that the model can process. Tokenizers handle padding, truncation, and conversion to tensor formats.

2.6 DataLoader

A DataLoader in PyTorch is an iterator that provides batches of data from a dataset. It handles shuffling, batching, and parallel processing, making it essential for efficient model training.

2.7 Model Fine-Tuning

Fine-tuning involves training a pre-trained model on a specific task or dataset to adapt it to the task's nuances. This typically requires a smaller dataset and fewer computational resources compared to training a model from scratch.

2.8 Trainer and TrainingArguments

The `Trainer` class in the `transformers` library simplifies the training and evaluation process for transformer models. `TrainingArguments` define the configuration for training, including batch size, learning rate, number of epochs, and evaluation strategy.

2.9 Evaluation Metrics

Evaluation metrics measure the performance of machine learning models. Common metrics for NLP tasks include:

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: Measures the overlap between the predicted and reference text. Common variants are ROUGE-1, ROUGE-2, and ROUGE-L.
- **BLEU (Bilingual Evaluation Understudy)**: Measures the precision of n-grams in the predicted text compared to the reference text.
- **METEOR (Metric for Evaluation of Translation with Explicit ORdering)**: Combines precision, recall, and synonymy to evaluate machine translation quality.

2.10 Visualization

Visualization involves creating graphical representations of data and model performance metrics to understand trends, patterns, and outliers. Libraries like Matplotlib and Seaborn are commonly used for this purpose.

3. Models

3.1 BERT (Bidirectional Encoder Representations from Transformers)

BERT is a transformer-based model developed by Google. It is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. BERT is pre-trained on a large corpus of text and fine-tuned on specific tasks, achieving state-of-the-art results in many NLP tasks.

3.2 GPT-2 (Generative Pre-trained Transformer 2)

GPT-2, developed by OpenAI, is a transformer-based model designed for text generation. It uses a large-scale unsupervised language model trained on a diverse dataset. GPT-2 generates coherent and contextually relevant text, making it suitable for tasks like text completion and question-answering.

3.3 T5 (Text-To-Text Transfer Transformer)

T5, developed by Google Research, is a transformer model that treats all NLP tasks as a text-to-text problem. It is pre-trained on a large dataset and fine-tuned for specific tasks by converting inputs and outputs into text. T5 achieves state-of-the-art results in many NLP benchmarks.

Detailed Explanation of Processes

Data Loading and Exploration

The first step in any data-driven project is to load and explore the data. This involves reading the dataset from a source (e.g., a CSV file, a database, or an online repository) and examining its structure and contents.

1. **Loading the Dataset**: We used the `datasets` library to load the Quora Question Answer dataset. This library provides a unified interface for accessing a wide variety of datasets and supports efficient data handling.
2. **Exploration**: We explored the dataset using `pandas` functions such as `head()`, `info()`, and `describe()`. These functions provide a quick overview of the data, including its structure, types, and summary statistics.
3. **Checking for Missing Values**: We checked for missing values using the `isnull().sum()` function, which helps identify columns with missing data that need to be handled.

Data Preprocessing

Data preprocessing is a crucial step to ensure the data is clean and suitable for analysis and model training. This involves several sub-processes:

1. **Cleaning the Data**: We removed any rows with missing values to ensure the dataset is complete and clean.
2. **Text Normalization**: We defined functions to clean and normalize the text. This included removing URLs, HTML tags, special characters, and digits, and converting the text to lowercase.
3. **Tokenization**: Tokenization involves breaking down text into smaller units, typically words or subwords. This is essential for transforming text data into a format that models can process.
4. **Stop Word Removal**: Stop words are common words (e.g., "and", "the", "is") that do not contribute much to the meaning of the text. We removed these words to reduce noise in the data.
5. **Stemming and Lemmatization**: These processes reduce words to their root or base form, which helps in reducing the dimensionality of the data and improves model performance.

Train-Test Split

To evaluate model performance, we split the dataset into training, validation, and test sets. This allows us to train the model on one subset of data, tune hyperparameters and evaluate performance during training on another subset, and finally assess the model's performance on unseen data.

1. **Training Set (80%)**: Used to train the model.
2. **Validation Set (10%)**: Used to evaluate model performance during training and tune hyperparameters.
3. **Test Set (10%)**: Used to assess the final model's performance on unseen data.

Dataset Conversion

Converting pandas DataFrames to Hugging Face `Dataset` and `DatasetDict` formats allows for seamless integration with the `transformers` library. This step ensures efficient data handling and processing, especially when dealing with large datasets.

Tokenization for Transformers

Transformers-based models require text to be tokenized into a specific format. This involves converting text into token IDs that the model can process. Tokenizers handle padding, truncation, and conversion to tensor formats, making it easier to feed data into the model.

DataLoader

A DataLoader in PyTorch is an iterator that provides batches of data from a dataset. It handles shuffling, batching, and parallel processing, which is essential for efficient model training. The DataLoader ensures that the model receives data in manageable chunks, optimizing memory usage and training speed.

Model Fine-Tuning

Fine-tuning involves training a pre-trained model on a specific task or dataset to adapt it to the task's nuances. This typically requires a smaller dataset and fewer computational resources compared to training a model from scratch. Fine-tuning leverages the knowledge the model has already acquired during pre-training on a large corpus of text.

1. **TrainingArguments**: Define the configuration for training, including batch size, learning rate, number of epochs, and evaluation strategy.
2. **Trainer**: Initialize the `Trainer` class with the model, training arguments, datasets, and tokenizer. The `Trainer` class simplifies the training and evaluation process for transformer models.

Evaluation Metrics

Evaluation metrics measure the performance of machine learning models. Common metrics for NLP tasks include:

1. **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: Measures the overlap between the predicted and reference text. ROUGE-1 measures unigram (word-level) overlap, ROUGE-2 measures bigram (two-word sequence) overlap, and ROUGE-L measures the longest common subsequence.
2. **BLEU (Bilingual Evaluation Understudy)**: Measures the precision of n-grams in the predicted text compared to the reference text. It is commonly used for evaluating machine translation and text generation models.
3. **METEOR (Metric for Evaluation of Translation with Explicit ORdering)**: Combines precision, recall, and synonymy to evaluate machine translation quality. METEOR aligns the predicted and reference text using synonyms and stemming, providing a more nuanced evaluation than BLEU.

Visualization

Visualization involves creating graphical representations of data and model performance metrics to understand trends, patterns, and outliers. Libraries like Matplotlib and Seaborn are commonly used for this purpose. Visualization helps in interpreting the results and identifying areas for improvement.

❖ Literature Survey

Title	First Author	Year	Journal/Source	Key Findings
DATLMedQA: A Data Augmentation and Transfer Learning Based Solution for Medical Question Answering	S.Z.	2021	Applied Sciences	Demonstrated better performance in medical question answering by pretraining BERT on medical samples, using GPT-2 for question augmentation, and T5-Small for answer

				extraction. Emphasized the effectiveness of question augmentation and transfer learning.
MedLM: Exploring Language Models for Medical Question Answering Systems	Niraj Yagnik	2023	arXiv	Investigated GPT-2, GPT-3, and T5 models for medical question answering. Fine-tuning on MedQuad and iClinic datasets with dynamic prompting improved model responses. The study provided insights into generative model optimization for medical applications.
A Comparative Study of Transformer-Based Language Models on Extractive Question Answering	Kate Pearce	2021	arXiv	Compared transformer models like BERT, RoBERTa, and ALBERT on SQuAD and NewsQA datasets. Evaluated F1 scores and highlighted the strengths and weaknesses of different transformer models in extractive question answering.
Question Answering and Text Generation Using BERT and GPT-2 Model	Santoshi Kumari	2020	SpringerLink	Explored the integration of BERT and GPT-2 for question answering and text generation. Highlighted the architectural details and performance metrics, demonstrating the effectiveness of combining generative and extractive models.
Learning to Answer by Learning to Ask: Getting the Best of GPT-2 and BERT for Question Answering	Tassilo Klein	2021	arXiv	Proposed a model combining GPT-2 and BERT for improved question answering. Used GPT-2 for generating question-aware input representations and BERT for extracting relevant answers. Showcased the benefits of integrating both generative and extractive models.
Fine-tuning GPT-2 Small for Question Answering	OpenAI	2019	GitHub	Detailed the fine-tuning process of GPT-2 on the Stanford Question Answering Dataset (SQuAD). Explained the training pipeline and model adjustments that led to improved question-answering capabilities.

Sentence and word embedding employed in open question-answering	Medved' M	2018	Natural Language Processing Centre	Discussed the use of sentence and word embeddings for open-domain question answering. Emphasized the importance of embedding techniques in improving the relevance and accuracy of generated answers.
DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter	Sanh V	2019	arXiv	Introduced DistilBERT, a distilled version of BERT, which is smaller, faster, and cheaper to train. Demonstrated that DistilBERT retains most of BERT's language understanding capabilities while being more efficient in terms of computational resources.

❖ Methodology

➢ T5 MODEL

1. Data Exploration, Cleaning, and Preprocessing

1.1 Data Loading and Exploration

The dataset used for this project is the Quora Question Answer Dataset, which was loaded using the `datasets` library from Hugging Face. The initial exploration involved loading the dataset into a panda DataFrame and examining its structure and content.

```
[ ]  !pip install torch transformers datasets nltk pandas matplotlib seaborn plotly scikit-learn evaluate rouge-score
```

```
# Load the dataset
dataset = load_dataset("toughdata/quora-question-answer-dataset")
df = pd.DataFrame(dataset['train'])

print(df.info())
print(df.head())
print(df.describe())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56402 entries, 0 to 56401
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   question  56402 non-null  object
 1   answer    56402 non-null  object
dtypes: object(2)
memory usage: 881.4+ KB
None
                                            question  \
0  Why whenever I get in the shower my girlfriend...
1            What is a proxy, and how can I use one?
2  What song has the lyrics "someone left the cak...
3  I am the owner of an adult website called http...
4  Does the Bible mention anything about a place ...

                                              answer
0  Isn’t it awful? You would swear that there was...
1  A proxy server is a system or router that prov...
2                                MacArthur's Park\n
3  Don’t let apps that are liers put adds on your...
4  St. John in the book of Revelation mentions an...
                                           question answer
count                                         56402  56402
unique                                         3234  54726
top     Would Hillary Clinton have made a better Presi...   No\n
freq                                            106     89
```

1.2 Handling Missing Values

We checked for null values in the dataset and dropped any rows containing missing data to ensure the quality and integrity of the dataset.

```python
# Drop any irrelevant columns (if any)
df = df[['question', 'answer']]

# Drop rows with missing values
df.dropna(inplace=True)

# Display cleaned data
print(df.head())
print(df.info())
```

```
                                          question  \
0  Why whenever I get in the shower my girlfriend...
1            What is a proxy, and how can I use one?
2  What song has the lyrics "someone left the cak...
3  I am the owner of an adult website called http...
4  Does the Bible mention anything about a place ...

                                              answer
0  Isn’t it awful? You would swear that there was...
1  A proxy server is a system or router that prov...
2                                MacArthur's Park\n
3  Don't let apps that are liers put adds on your...
4  St. John in the book of Revelation mentions an...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56402 entries, 0 to 56401
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   question  56402 non-null  object
 1   answer    56402 non-null  object
dtypes: object(2)
memory usage: 881.4+ KB
None
```

1.3 Data Cleaning and Text Preprocessing

The text data was cleaned to remove any irrelevant information such as URLs, HTML tags, special characters, and digits. This was done using regular expressions.

- **Stop Words and Stemming**: We define stop words using NLTK and create a PorterStemmer object for stemming words.
- **Cleaning Text:** The clean_text function normalizes whitespace, removes URLs, HTML tags, special characters, and digits, and converts text to lowercase.
- **Preprocessing Text:** The preprocess_text function cleans the text, tokenizes it, removes stop words, and applies stemming.
- **Parallel Processing**: We use `ThreadPoolExecutor` to preprocess the `question` and `answer` columns in parallel, improving the efficiency of our preprocessing step

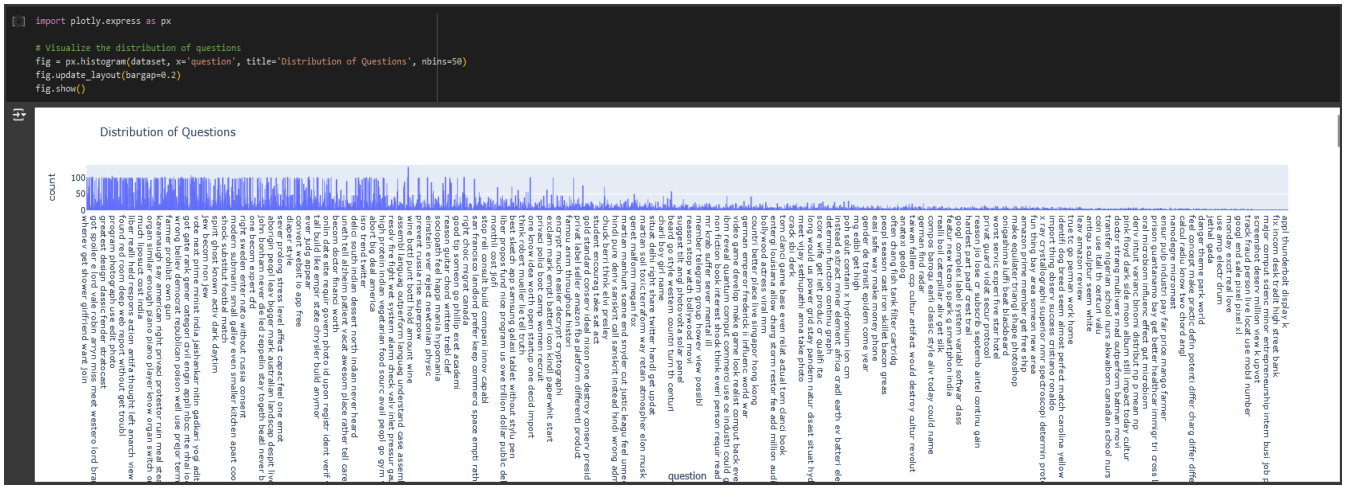
```python
from concurrent.futures import ThreadPoolExecutor
# Data Cleaning and Preprocessing
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def clean_text(text):
    # Normalize whitespace
    text = re.sub(r'\s+', ' ', text)

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove special characters and digits
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\d', ' ', text)

    # Convert to lowercase
    text = text.lower().strip()

    return text
```

```python
def preprocess_text(text):
    # Clean the text
    text = clean_text(text)

    # Tokenization
    tokens = word_tokenize(text)

    # Stop word removal and stemming
    tokens = [ps.stem(word) for word in tokens if word not in stop_words]

    return ' '.join(tokens)

# Apply preprocessing with parallel processing
with ThreadPoolExecutor(max_workers=2) as executor:
    dataset['question'] = list(executor.map(preprocess_text, dataset['question']))
    dataset['answer'] = list(executor.map(preprocess_text, dataset['answer']))

print(dataset.head())
```

```
                                        question  \
0            whenev get shower girlfriend want join
1                                  proxi use one
2                  song lyric someon left cake rain
3  owner adult websit call anyon offer seo tip he...
4              bibl mention anyth place heaven hell

                                          answer
0         aw would swear enough hot water go around
1  proxi server system router provid gateway user...
2                                  macarthur park
3  let app lier put add site like one say free ag...
4  st john book revel mention address scholar sug...
```

1.5 Visualizing Data-Set

Visualization: We use `plotly.express` to create a histogram showing the distribution of questions in the dataset.



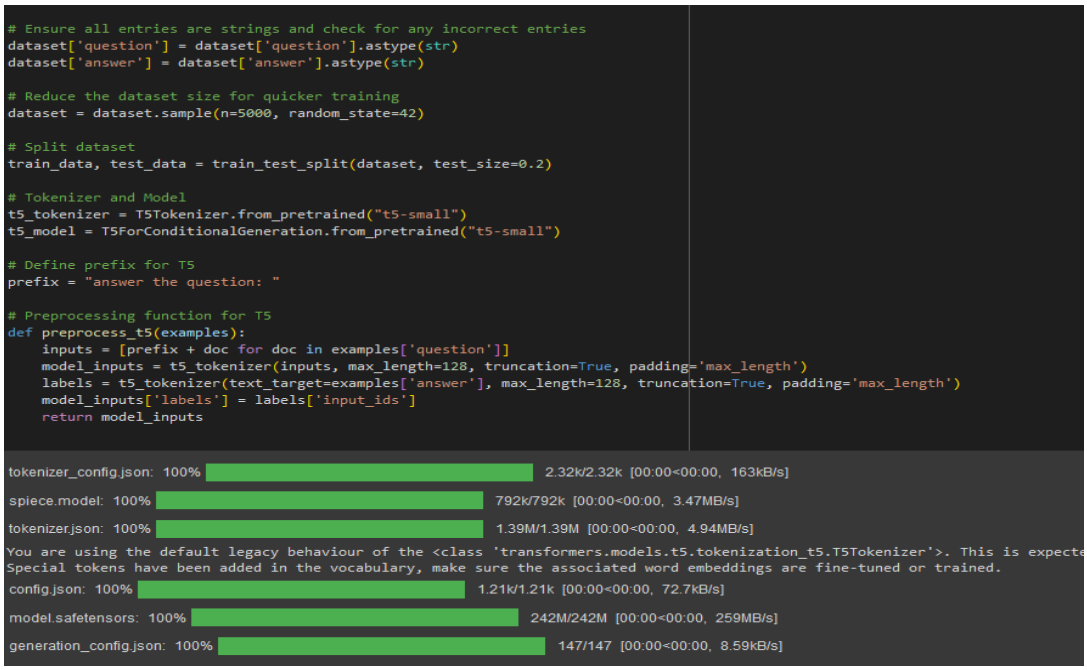
2. Model Preparation and Tokenization

 Splitting Dataset: Dataset is split into training and testing set for training and testing the model.

 Dataset Conversion: Convert pandas DataFrame to a DatasetDict format.

 Tokenization: Tokenize the train and test datasets using the preprocess_t5 function.

 Data Collator: Use DataCollatorForSeq2Seq to handle padding and other preprocessing steps during training.



```python
# Convert DataFrame to DatasetDict
from datasets import Dataset
train_dataset = Dataset.from_pandas(train_data)
test_dataset = Dataset.from_pandas(test_data)

# Tokenize data without using multiprocessing
train_dataset = train_dataset.map(preprocess_t5, batched=True, batch_size=32)
test_dataset = test_dataset.map(preprocess_t5, batched=True, batch_size=32)

# Data Collator
t5_data_collator = DataCollatorForSeq2Seq(tokenizer=t5_tokenizer, model=t5_model)
```

```
Map: 100% 4000/4000 [00:05<00:00, 769.72 examples/s]
Map: 100% 1000/1000 [00:02<00:00, 376.67 examples/s]
```

3. Training the Model

Training Arguments

- **Training Configuration**: We set up the training arguments, including learning rate, batch size, number of epochs, mixed precision training (fp16), and evaluation strategy.
- **Metrics Calculation**: This function calculates evaluation metrics (ROUGE, BLEU, METEOR) for the model's predictions.
- **Trainer Initialization**: We initialize the `Seq2SeqTrainer` with the model, training arguments, datasets, tokenizer, data collator, and metrics computation function.
- **Training**: Start the training process using the `train` method.
- **Evaluation**: Evaluate the model on the test dataset to get performance metrics.
- **Generate Answers**: Define a function to generate answers for given questions using the trained T5 model.
- **Example Usage**: Test the function with an example question.

```python
[ ] training_args = Seq2SeqTrainingArguments(
        output_dir="./results",
        eval_strategy="epoch",
        learning_rate=3e-4,
        per_device_train_batch_size=8,  # Increased batch size for faster training
        per_device_eval_batch_size=8,
        weight_decay=0.01,
        save_total_limit=3,
        num_train_epochs=10,  # Reduced epochs for faster training
        predict_with_generate=True,
        fp16=True,  # Use mixed precision training
        dataloader_num_workers=0,  # Set to 0 to avoid fork issues
        push_to_hub=False
    )

    # Define a function for computing metrics
    def compute_metrics(pred, tokenizer):
        preds, labels = pred
        labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
        decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)
        decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

        decoded_preds = ["\n".join(nltk.sent_tokenize(pred.strip())) for pred in decoded_preds]
        decoded_labels = ["\n".join(nltk.sent_tokenize(label.strip())) for label in decoded_labels]

        rouge = evaluate.load("rouge")
        bleu = evaluate.load("bleu")
        meteor = evaluate.load("meteor")

        rouge_result = rouge.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)
        bleu_result = bleu.compute(predictions=decoded_preds, references=decoded_labels)
        meteor_result = meteor.compute(predictions=decoded_preds, references=decoded_labels)
```

```
    # Log each metric separately
    return {
        "rouge1": rouge_result['rouge1'],
        "rouge2": rouge_result['rouge2'],
        "rougeL": rouge_result['rougeL'],
        "rougeLsum": rouge_result['rougeLsum'],
        "bleu": bleu_result['bleu'],
        "meteor": meteor_result['meteor']
    }


# Trainer
t5_trainer = Seq2SeqTrainer(
    model=t5_model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=t5_tokenizer,
    data_collator=t5_data_collator,
    compute_metrics=lambda p: compute_metrics(p, t5_tokenizer)
)

# Train the model
t5_trainer.train()

# Evaluate the model
results = t5_trainer.evaluate()


# Example of using the model to generate a response
def generate_answer(question):
    input_text = prefix + question
    input_ids = t5_tokenizer.encode(input_text, return_tensors='pt').to(t5_trainer.args.device)
    output_ids = t5_model.generate(input_ids, max_new_tokens=20)  # Explicitly setting max_new_tokens
    answer = t5_tokenizer.decode(output_ids[0], skip_special_tokens=True)
    return answer


# Example usage
example_question = "What is the capital of France?"
```

[5000/5000 19:55, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Bleu	Meteor
1	3.026000	2.636352	0.057171	0.012903	0.052703	0.052841	0.000281	0.029079
2	2.691600	2.552497	0.060027	0.013852	0.055375	0.055379	0.000297	0.029853
3	2.591300	2.509202	0.063803	0.014458	0.059113	0.059236	0.000380	0.031795
4	2.523900	2.483314	0.061255	0.014181	0.056685	0.056673	0.000348	0.031293
5	2.472300	2.465156	0.066543	0.015481	0.061180	0.061254	0.000442	0.034042
6	2.433500	2.453136	0.073113	0.018679	0.066680	0.066719	0.000538	0.039114
7	2.401700	2.447330	0.073050	0.018473	0.066981	0.067133	0.000531	0.038661
8	2.378200	2.442250	0.072121	0.018142	0.066523	0.066558	0.000513	0.038600
9	2.360200	2.441650	0.076211	0.019458	0.069831	0.069870	0.000601	0.040740
10	2.347700	2.441117	0.076775	0.019678	0.070246	0.070321	0.000592	0.040804

4. Loading and Testing the Model

- **Load Model and Tokenizer**: Load the trained model and tokenizer from the specified paths.
    - **Test Model Function**: Define a function to test the model on a sample of the test data and compute evaluation metrics.

- **Sample Test Dataset**: Select a sample of 10 rows from the test data for testing.
- **Compute Metrics**: Generate answers for the test sample and compute metrics.

- **Display Metrics**: Print evaluation metrics in a tabular format.
- **Display Test Results**: Print sample test results in a tabular format.

```
t5_model.save_pretrained("./results/t5_model")
t5_tokenizer.save_pretrained("./results/t5_tokenizer")
```

```
('./results/t5_tokenizer/tokenizer_config.json',
 './results/t5_tokenizer/special_tokens_map.json',
 './results/t5_tokenizer/spiece.model',
 './results/t5_tokenizer/added_tokens.json')
```

```
import pandas as pd
from tabulate import tabulate

# Load the trained model and tokenizer
model_path = "./results/t5_model"
tokenizer_path = "./results/t5_tokenizer"
t5_tokenizer = T5Tokenizer.from_pretrained(tokenizer_path)
t5_model = T5ForConditionalGeneration.from_pretrained(model_path)

# Ensure NLTK resources are downloaded
nltk.download('punkt')

# Function to generate answers and compute metrics
def test_model(test_data, tokenizer, model):
    prefix = "answer the question: "

    def generate_answer(question):
        input_text = prefix + question
        input_ids = tokenizer.encode(input_text, return_tensors='pt').to(model.device)
        output_ids = model.generate(
            input_ids,
            max_length=50,
            num_beams=5,
            early_stopping=True
        )
        answer = tokenizer.decode(output_ids[0], skip_special_tokens=True)
        return answer

    test_data['generated_answer'] = test_data['question'].apply(generate_answer)
```

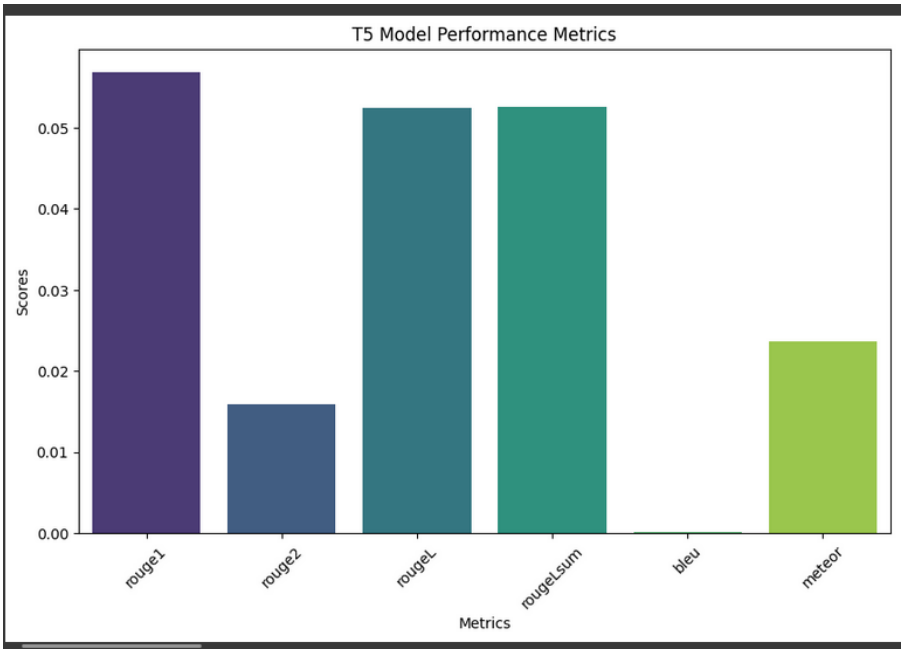
```python
    decoded_preds = test_data['generated_answer'].tolist()
    decoded_labels = test_data['answer'].tolist()

    decoded_preds = ["\n".join(nltk.sent_tokenize(pred.strip())) for pred in decoded_preds]
    decoded_labels = ["\n".join(nltk.sent_tokenize(label.strip())) for label in decoded_labels]

    rouge = evaluate.load("rouge")
    bleu = evaluate.load("bleu")
    meteor = evaluate.load("meteor")

    rouge_result = rouge.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)
    bleu_result = bleu.compute(predictions=decoded_preds, references=decoded_labels)
    meteor_result = meteor.compute(predictions=decoded_preds, references=decoded_labels)

    metrics = {
        "rouge1": rouge_result['rouge1'],
        "rouge2": rouge_result['rouge2'],
        "rougeL": rouge_result['rougeL'],
        "rougeLsum": rouge_result['rougeLsum'],
        "bleu": bleu_result['bleu'],
        "meteor": meteor_result['meteor']
    }

    return metrics, test_data

# Prepare a sample test dataset
test_sample = test_data.sample(n=10, random_state=42)  # Adjust the sample size as needed

# Test the model
metrics, test_results = test_model(test_sample, t5_tokenizer, t5_model)

# Convert metrics to DataFrame
metrics_df = pd.DataFrame([metrics])

# Convert test results to DataFrame for display
test_results_df = test_results[['question', 'answer', 'generated_answer']]

# Display metrics in table form
print("Evaluation Metrics:")
print(tabulate(metrics_df, headers='keys', tablefmt='psql'))

# Display sample test results in table form
print("\nSample Test Results:")
print(tabulate(test_results_df, headers='keys', tablefmt='psql'))
```

```
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
Evaluation Metrics:
+----+-----------+-----------+-----------+-------------+------------+-----------+
|    |    rouge1 |    rouge2 |    rougeL |   rougeLsum |       bleu |    meteor |
|----+-----------+-----------+-----------+-------------+------------+-----------|
|  0 | 0.0568154 | 0.0158823 | 0.0524056 |   0.0525869 | 0.00015037 | 0.0237079 |
+----+-----------+-----------+-----------+-------------+------------+-----------+

Sample Test Results:
+-------+-------------------------------------------------------------------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------
|       | question                                                                | answer
|-------+-------------------------------------------------------------------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------
| 18219 | ceo start typic day look like respons grow compani everi day            | presum start ceo also founder co founder start ceo need high level flexibl get thing done activ boundari less ceo need keep razor sharp focu key prioriti becom major
| 46355 | favorit sci fi book seri                                                | best scienc fiction novel captur imagin reader georg orwel dystopian classic explor futur total govern surveil control dune frank herbert space epic set desert plane
| 33815 | butter rather healthier margarin serv restaur                           | first margarin repeat better claim debunk decad ago second margarin consider cheaper butter less expens restaur
| 47245 | realli true chines govern implant peopl monetari microchip              | appar shenzhen govern china inde implant non monetari microchip dog follow exampl uk japan australia linked_text dog shenzhen china get microchip techcrunch url dog
| 11878 | disadvantag use robot                                                   | know number dairi farmer decid go robot rather hire contract milker reason privaci money robot expens even amort number year cost much salari contract milker mean so
| 12851 | necessari term person loan                                              | list common term one know go ahead idea appli loan interest appli person loan agre repay debt interest essenti lender charg allow use money repay time pay monthli in
| 39736 | like slider person whose energi effect electron equip make shut malfunct go haywir | question like slider person whose energi effect electron equip make shut malfunct go haywir never heard term slider context today live one wife initi thought coincid
| 36633 | effect softwar tool insid sale team use boost product                   | hi lucep app person recommend person recommend lucep linked_text app url sale enabl softwar work callback technolog compris featur like instant lead notificationslea
| 17592 | amaz photo ever taken                                                   |
| 22712 | manag said abl keep posit anymor ask resign letter fire quit            | fell trap fall promis month pay payday came around got hour question ask proof pocket alot big british busi aswel
```

5. Visualizing Metrics

Visualization: Create a bar plot to visualize the performance metrics of the T5 model using `seaborn`.

```python
# Visualization Code for T5 Model Metrics
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tabulate import tabulate

# Load the trained model and tokenizer
model_path = "./results/t5_model"
tokenizer_path = "./results/t5_tokenizer"
t5_tokenizer = T5Tokenizer.from_pretrained(tokenizer_path)
t5_model = T5ForConditionalGeneration.from_pretrained(model_path)

# Ensure NLTK resources are downloaded
nltk.download('punkt')

# Function to generate answers and compute metrics
def test_model(test_data, tokenizer, model):
    prefix = "answer the question: "

    def generate_answer(question):
        input_text = prefix + question
        input_ids = tokenizer.encode(input_text, return_tensors='pt').to(model.device)
        output_ids = model.generate(
            input_ids,
            max_length=50,
            num_beams=5,
            early_stopping=True
        )
        answer = tokenizer.decode(output_ids[0], skip_special_tokens=True)
        return answer

    test_data['generated_answer'] = test_data['question'].apply(generate_answer)

    decoded_preds = test_data['generated_answer'].tolist()
    decoded_labels = test_data['answer'].tolist()

    decoded_preds = ["\n".join(nltk.sent_tokenize(pred.strip())) for pred in decoded_preds]
    decoded_labels = ["\n".join(nltk.sent_tokenize(label.strip())) for label in decoded_labels]

    rouge = evaluate.load("rouge")
    bleu = evaluate.load("bleu")
    meteor = evaluate.load("meteor")
```



2) BERT MODEL

1. Data Exploration, Cleaning, and Preprocessing

1.1 Data Loading and Exploration

The dataset used for this project is the Quora Question Answer Dataset, which was loaded using the `datasets` library from Hugging Face. The initial exploration involved loading the dataset into a panda DataFrame and examining its structure and content.

```
[ ]  !pip install torch transformers datasets nltk pandas matplotlib seaborn plotly scikit-learn evaluate rouge-score
```

```
# Load the dataset
dataset = load_dataset("toughdata/quora-question-answer-dataset")
df = pd.DataFrame(dataset['train'])

print(df.info())
print(df.head())
print(df.describe())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56402 entries, 0 to 56401
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   question  56402 non-null  object
 1   answer    56402 non-null  object
dtypes: object(2)
memory usage: 881.4+ KB
None
                                            question  \
0  Why whenever I get in the shower my girlfriend...
1            What is a proxy, and how can I use one?
2  What song has the lyrics "someone left the cak...
3  I am the owner of an adult website called http...
4  Does the Bible mention anything about a place ...

                                              answer
0  Isn't it awful? You would swear that there was...
1  A proxy server is a system or router that prov...
2                                MacArthur's Park\n
3  Don't let apps that are liers put adds on your...
4  St. John in the book of Revelation mentions an...
                                                question answer
count                                            56402  56402
unique                                            3234  54726
top     Would Hillary Clinton have made a better Presi...   No\n
freq                                                106     89
```

1.2 Handling Missing Values

We checked for null values in the dataset and dropped any rows containing missing data to ensure the quality and integrity of the dataset.

```
# Drop any irrelevant columns (if any)
df = df[['question', 'answer']]

# Drop rows with missing values
df.dropna(inplace=True)

# Display cleaned data
print(df.head())
print(df.info())
```

```
                                            question  \
0  Why whenever I get in the shower my girlfriend...
1            What is a proxy, and how can I use one?
2  What song has the lyrics "someone left the cak...
3  I am the owner of an adult website called http...
4  Does the Bible mention anything about a place ...

                                              answer
0  Isn't it awful? You would swear that there was...
1  A proxy server is a system or router that prov...
2                                MacArthur's Park\n
3  Don't let apps that are liers put adds on your...
4  St. John in the book of Revelation mentions an...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56402 entries, 0 to 56401
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   question  56402 non-null  object
 1   answer    56402 non-null  object
dtypes: object(2)
memory usage: 881.4+ KB
None
```

1.3 Data Cleaning and Text Preprocessing

The text data was cleaned to remove any irrelevant information such as URLs, HTML tags, special characters, and digits. This was done using regular expressions.

- **Stop Words and Stemming**: We define stop words using NLTK and create a PorterStemmer object for stemming words.
- **Cleaning Text:** The clean_text function normalizes whitespace, removes URLs, HTML tags, special characters, and digits, and converts text to lowercase.
- **Preprocessing Text:** The preprocess_text function cleans the text, tokenizes it, removes stop words, and applies stemming.
- **Parallel Processing**: We use `ThreadPoolExecutor` to preprocess the `question` and `answer` columns in parallel, improving the efficiency of our preprocessing step

```python
from concurrent.futures import ThreadPoolExecutor
# Data Cleaning and Preprocessing
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def clean_text(text):
    # Normalize whitespace
    text = re.sub(r'\s+', ' ', text)

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove special characters and digits
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\d', ' ', text)

    # Convert to lowercase
    text = text.lower().strip()

    return text
```

```python
def preprocess_text(text):
    # Clean the text
    text = clean_text(text)

    # Tokenization
    tokens = word_tokenize(text)

    # Stop word removal and stemming
    tokens = [ps.stem(word) for word in tokens if word not in stop_words]

    return ' '.join(tokens)

# Apply preprocessing with parallel processing
with ThreadPoolExecutor(max_workers=2) as executor:
    dataset['question'] = list(executor.map(preprocess_text, dataset['question']))
    dataset['answer'] = list(executor.map(preprocess_text, dataset['answer']))

print(dataset.head())
```

```
                                        question  \
0            whenev get shower girlfriend want join
1                                  proxi use one
2                  song lyric someon left cake rain
3  owner adult websit call anyon offer seo tip he...
4              bibl mention anyth place heaven hell

                                        answer
0         aw would swear enough hot water go around
1  proxi server system router provid gateway user...
2                                macarthur park
3  let app lier put add site like one say free ag...
4  st john book revel mention address scholar sug...
```

➢ **Converting DataFrames to Datasets**

- **Conversion**: Convert the pandas DataFrames to Hugging Face `Dataset` and then to `DatasetDict` for ease of use with the transformers library.

```
# Convert DataFrames to Datasets
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

# Create a DatasetDict
dataset_dict = {
    'train': train_dataset,
    'validation': val_dataset,
    'test': test_dataset
}

# Convert to DatasetDict
final_dataset = DatasetDict(dataset_dict)

# Check the splits
print(final_dataset)
```

2) Data Preprocessing

➢ **Text Preprocessing**: We download additional NLTK resources and preprocess the text by tokenizing, removing stop words, and lemmatizing each token.
➢ **Tokenization**: We define a function to tokenize the questions and answers, then apply it to the entire dataset using `map`.
➢ **Stop Words and Stemming**: We define stop words using NLTK and create a PorterStemmer object for stemming words.

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')

def preprocess_text(text):
    tokens = nltk.word_tokenize(text)
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token.lower() not in stop_words]
    return ' '.join(tokens)

train_df['question'] = train_df['question'].apply(preprocess_text)
train_df['answer'] = train_df['answer'].apply(preprocess_text)
val_df['question'] = val_df['question'].apply(preprocess_text)
val_df['answer'] = val_df['answer'].apply(preprocess_text)
test_df['question'] = test_df['question'].apply(preprocess_text)
test_df['answer'] = test_df['answer'].apply(preprocess_text)

# Convert DataFrames to Datasets
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

# Create a DatasetDict
dataset_dict = {
    'train': train_dataset,
    'validation': val_dataset,
    'test': test_dataset
}

# Convert to DatasetDict
final_dataset = DatasetDict(dataset_dict)

def tokenize(example):
    return tokenizer(example['question'], example['answer'], truncation=True, padding='max_length', max_length=128)

tokenized_datasets = final_dataset.map(tokenize, batched=True)
```

3) Creating DataLoader

- **DataLoader**: We create a custom dataset class and DataLoader for the training, validation, and test sets to handle batching and shuffling.

```python
class QADataset:
    def __init__(self, dataset, tokenizer):
        self.dataset = dataset
        self.tokenizer = tokenizer

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

    def __getitem__(self, idx):
        item = self.dataset[idx]
        question = item['question']
        answer = item['answer']

        # Tokenize question and answer
        encoding = self.tokenizer(question, answer, return_tensors='pt', max_length=128, padding='max_length', truncation=True)

        # Calculate start and end positions
        input_ids = encoding['input_ids'].squeeze()
        attention_mask = encoding['attention_mask'].squeeze()

        answer_ids = self.tokenizer(answer, return_tensors='pt', max_length=128, padding='max_length', truncation=True)['input_ids'].squeeze()
        start_positions = (input_ids == answer_ids[0]).nonzero(as_tuple=True)
        if len(start_positions[0]) == 0:
            start_positions = 0
            end_positions = 0
        else:
            start_positions = start_positions[0][0].item()
            end_positions = start_positions + len(answer_ids) - 1

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'start_positions': int(start_positions),
            'end_positions': int(end_positions)
        }

train_dataset = QADataset(tokenized_datasets['train'], tokenizer)
val_dataset = QADataset(tokenized_datasets['validation'], tokenizer)
test_dataset = QADataset(tokenized_datasets['test'], tokenizer)
```

4) Model Fine-Tuning

- **TrainingArguments**: We define the training arguments, including batch size, learning rate, number of epochs, and other configurations.
- **Trainer**: Initialize the Trainer class with the model, training arguments, datasets, and tokenizer, and then train the model.

Training Arguments

- **Training Configuration**: We set up the training arguments, including learning rate, batch size, number of epochs, mixed precision training (fp16), and evaluation strategy.
- **Metrics Calculation**: This function calculates evaluation metrics (ROUGE, BLEU, METEOR) for the model's predictions.
- **Trainer Initialization**: We initialize the `Seq2SeqTrainer` with the model, training arguments, datasets, tokenizer, data collator, and metrics computation function.
- **Training**: Start the training process using the `train` method.
- **Evaluation**: Evaluate the model on the test dataset to get performance metrics.
- **Generate Answers**: Define a function to generate answers for given questions using the trained T5 model.
- **Example Usage**: Test the function with an example question.

```python
def fine_tune_model(model, train_dataset, eval_dataset, model_name):
    training_args = TrainingArguments(
        output_dir=f'./results/{model_name}',
        evaluation_strategy="epoch",
        learning_rate=2e-5,
        per_device_train_batch_size=48,  # Reduced batch size
        per_device_eval_batch_size=48,  # Reduced batch size
        num_train_epochs=8,  # Single epoch to fit within Colab's free resources
        weight_decay=0.01,
        save_steps=10_000,
        save_total_limit=2,
        logging_dir='./logs',
        logging_steps=10,
        fp16=True,  # Use mixed precision training if supported
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        tokenizer=tokenizer,
        # compute_metrics=compute_metrics
    )

    trainer.train()
    return trainer

model = BertForQuestionAnswering.from_pretrained('bert-base-uncased')
bert_trainer = fine_tune_model(model, train_dataset, val_dataset, 'bert')
```

Epoch	Training Loss	Validation Loss
1	0.000100	0.000036
2	0.000000	0.000016
3	0.000000	0.000010
4	0.000000	0.000008
5	0.000000	0.000006
6	0.000000	0.000005
7	0.000000	0.000004

[7528/7528 48:40, Epoch 8/8]

Saving the Model

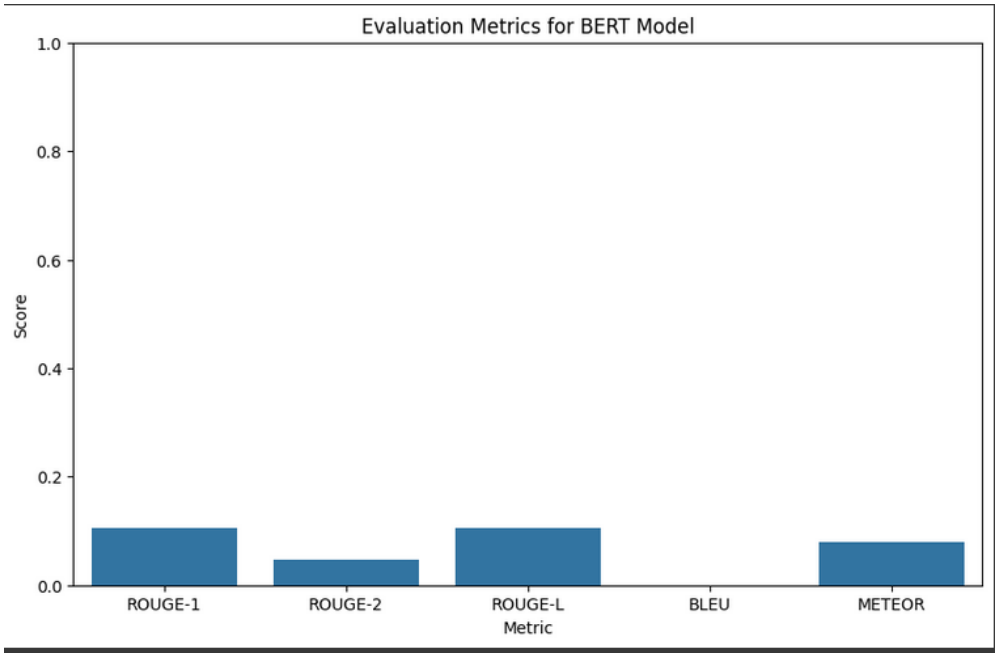
```python
model.save_pretrained(f'./results/bert')
tokenizer.save_pretrained(f'./results/bert')
```

- **Load Model and Tokenizer**: Load the trained model and tokenizer from the specified paths.
    - **Test Model Function**: Define a function to test the model on a sample of the test data and compute evaluation metrics.
    - **Sample Test Dataset**: Select a sample of 10 rows from the test data for testing.
    - **Compute Metrics**: Generate answers for the test sample and compute metrics.

	Metric	Precision	Recall	F-Measure
0	ROUGE-1	0.59	0.07244662472878502	0.105993
1	ROUGE-2	0.29	0.032493629193060265	0.0480365
2	ROUGE-L	0.59	0.07272519814202122	0.10588
3	BLEU	0.9633699633699634	N/A	8.2094e-20
4	METEOR	N/A	N/A	0.080459

Visualizing Metrics

Visualization: Create a bar plot to visualize the performance metrics of the T5 model using `seaborn`.



3) GPT MODEL

1. Data Exploration, Cleaning, and Preprocessing

1.1 Data Loading and Exploration

The dataset used for this project is the Quora Question Answer Dataset, which was loaded using the `datasets` library from Hugging Face. The initial exploration involved loading the dataset into a panda DataFrame and examining its structure and content.

```
[ ]  !pip install torch transformers datasets nltk pandas matplotlib seaborn plotly scikit-learn evaluate rouge-score
```

```
# Load the dataset
dataset = load_dataset("toughdata/quora-question-answer-dataset")
df = pd.DataFrame(dataset['train'])

print(df.info())
print(df.head())
print(df.describe())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56402 entries, 0 to 56401
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   question  56402 non-null  object
 1   answer    56402 non-null  object
dtypes: object(2)
memory usage: 881.4+ KB
None
                                          question  \
0  Why whenever I get in the shower my girlfriend...
1            What is a proxy, and how can I use one?
2  What song has the lyrics "someone left the cak...
3  I am the owner of an adult website called http...
4  Does the Bible mention anything about a place ...

                                              answer
0  Isn't it awful? You would swear that there was...
1  A proxy server is a system or router that prov...
2                                MacArthur's Park\n
3  Don't let apps that are liers put adds on your...
4  St. John in the book of Revelation mentions an...
                                                 question answer
count                                               56402  56402
unique                                               3234  54726
top     Would Hillary Clinton have made a better Presi...   No\n
freq                                                  106     89
```

1.2 Handling Missing Values

We checked for null values in the dataset and dropped any rows containing missing data to ensure the quality and integrity of the dataset.

```
# Drop any irrelevant columns (if any)
df = df[['question', 'answer']]

# Drop rows with missing values
df.dropna(inplace=True)

# Display cleaned data
print(df.head())
print(df.info())
```

```
                                          question  \
0  Why whenever I get in the shower my girlfriend...
1            What is a proxy, and how can I use one?
2  What song has the lyrics "someone left the cak...
3  I am the owner of an adult website called http...
4  Does the Bible mention anything about a place ...

                                              answer
0  Isn't it awful? You would swear that there was...
1  A proxy server is a system or router that prov...
2                                MacArthur's Park\n
3  Don't let apps that are liers put adds on your...
4  St. John in the book of Revelation mentions an...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56402 entries, 0 to 56401
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   question  56402 non-null  object
 1   answer    56402 non-null  object
dtypes: object(2)
memory usage: 881.4+ KB
None
```

1.3 Data Cleaning and Text Preprocessing

The text data was cleaned to remove any irrelevant information such as URLs, HTML tags, special characters, and digits. This was done using regular expressions.

- **Stop Words and Stemming**: We define stop words using NLTK and create a PorterStemmer object for stemming words.
- **Cleaning Text:** The clean_text function normalizes whitespace, removes URLs, HTML tags, special characters, and digits, and converts text to lowercase.
- **Preprocessing Text:** The preprocess_text function cleans the text, tokenizes it, removes stop words, and applies stemming.
- **Parallel Processing**: We use `ThreadPoolExecutor` to preprocess the `question` and `answer` columns in parallel, improving the efficiency of our preprocessing step

```python
from concurrent.futures import ThreadPoolExecutor
# Data Cleaning and Preprocessing
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def clean_text(text):
    # Normalize whitespace
    text = re.sub(r'\s+', ' ', text)

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove special characters and digits
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\d', ' ', text)

    # Convert to lowercase
    text = text.lower().strip()

    return text
```

```python
def preprocess_text(text):
    # Clean the text
    text = clean_text(text)

    # Tokenization
    tokens = word_tokenize(text)

    # Stop word removal and stemming
    tokens = [ps.stem(word) for word in tokens if word not in stop_words]

    return ' '.join(tokens)

# Apply preprocessing with parallel processing
with ThreadPoolExecutor(max_workers=2) as executor:
    dataset['question'] = list(executor.map(preprocess_text, dataset['question']))
    dataset['answer'] = list(executor.map(preprocess_text, dataset['answer']))

print(dataset.head())
```

```
                                        question  \
0            whenev get shower girlfriend want join
1                                  proxi use one
2                  song lyric someon left cake rain
3  owner adult websit call anyon offer seo tip he...
4              bibl mention anyth place heaven hell

                                        answer
0         aw would swear enough hot water go around
1  proxi server system router provid gateway user...
2                                macarthur park
3  let app lier put add site like one say free ag...
4  st john book revel mention address scholar sug...
```

➢ **Converting DataFrames to Datasets**

- **Conversion**: Convert the pandas DataFrames to Hugging Face `Dataset` and then to `DatasetDict` for ease of use with the transformers library.

```
# Convert DataFrames to Datasets
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

# Create a DatasetDict
dataset_dict = {
    'train': train_dataset,
    'validation': val_dataset,
    'test': test_dataset
}

# Convert to DatasetDict
final_dataset = DatasetDict(dataset_dict)

# Check the splits
print(final_dataset)
```

2) Data Preprocessing

➢ **Text Preprocessing**: We download additional NLTK resources and preprocess the text by tokenizing, removing stop words, and lemmatizing each token.
➢ **Tokenization**: We define a function to tokenize the questions and answers, then apply it to the entire dataset using `map`.
➢ **Stop Words and Stemming**: We define stop words using NLTK and create a PorterStemmer object for stemming words.

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')

def preprocess_text(text):
    tokens = nltk.word_tokenize(text)
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token.lower() not in stop_words]
    return ' '.join(tokens)

train_df['question'] = train_df['question'].apply(preprocess_text)
train_df['answer'] = train_df['answer'].apply(preprocess_text)
val_df['question'] = val_df['question'].apply(preprocess_text)
val_df['answer'] = val_df['answer'].apply(preprocess_text)
test_df['question'] = test_df['question'].apply(preprocess_text)
test_df['answer'] = test_df['answer'].apply(preprocess_text)

# Convert DataFrames to Datasets
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

# Create a DatasetDict
dataset_dict = {
    'train': train_dataset,
    'validation': val_dataset,
    'test': test_dataset
}

# Convert to DatasetDict
final_dataset = DatasetDict(dataset_dict)

def tokenize(example):
    return tokenizer(example['question'], example['answer'], truncation=True, padding='max_length', max_length=128)

tokenized_datasets = final_dataset.map(tokenize, batched=True)
```

3) Creating DataLoader

- **DataLoader**: We create a custom dataset class and DataLoader for the training, validation, and test sets to handle batching and shuffling.

```python
class QADataset:
    def __init__(self, dataset, tokenizer):
        self.dataset = dataset
        self.tokenizer = tokenizer

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

    def __getitem__(self, idx):
        item = self.dataset[idx]
        question = item['question']
        answer = item['answer']

        # Tokenize question and answer
        encoding = self.tokenizer(question, answer, return_tensors='pt', max_length=128, padding='max_length', truncation=True)

        # Calculate start and end positions
        input_ids = encoding['input_ids'].squeeze()
        attention_mask = encoding['attention_mask'].squeeze()

        answer_ids = self.tokenizer(answer, return_tensors='pt', max_length=128, padding='max_length', truncation=True)['input_ids'].squeeze()
        start_positions = (input_ids == answer_ids[0]).nonzero(as_tuple=True)
        if len(start_positions[0]) == 0:
            start_positions = 0
            end_positions = 0
        else:
            start_positions = start_positions[0][0].item()
            end_positions = start_positions + len(answer_ids) - 1

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'start_positions': int(start_positions),
            'end_positions': int(end_positions)
        }

train_dataset = QADataset(tokenized_datasets['train'], tokenizer)
val_dataset = QADataset(tokenized_datasets['validation'], tokenizer)
test_dataset = QADataset(tokenized_datasets['test'], tokenizer)
```

4) Model Fine-Tuning

- **TrainingArguments**: We define the training arguments, including batch size, learning rate, number of epochs, and other configurations.
- **Trainer**: Initialize the Trainer class with the model, training arguments, datasets, and tokenizer, and then train the model.

Training Arguments

- **Training Configuration**: We set up the training arguments, including learning rate, batch size, number of epochs, mixed precision training (fp16), and evaluation strategy.
- **Metrics Calculation**: This function calculates evaluation metrics (ROUGE, BLEU, METEOR) for the model's predictions.
- **Trainer Initialization**: We initialize the `Seq2SeqTrainer` with the model, training arguments, datasets, tokenizer, data collator, and metrics computation function.
- **Training**: Start the training process using the `train` method.
- **Evaluation**: Evaluate the model on the test dataset to get performance metrics.
- **Generate Answers**: Define a function to generate answers for given questions using the trained T5 model.

```python
from transformers import Trainer, TrainingArguments

def fine_tune_model(model, train_loader, eval_loader, model_name):
    training_args = TrainingArguments(
        output_dir=f'./results/{model_name}',
        evaluation_strategy="epoch",
        learning_rate=2e-5,
        per_device_train_batch_size=48,
        per_device_eval_batch_size=48,
        num_train_epochs=1,
        weight_decay=0.01,
        save_steps=10_000,
        save_total_limit=2,
        logging_dir='./logs',
        logging_steps=10,
        fp16=True,
        optim='adamw_torch'
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_loader.dataset,
        eval_dataset=eval_loader.dataset,
        tokenizer=tokenizer,
    )

    trainer.train()
    return trainer

# Load pre-trained GPT-2 model and fine-tune it
model = GPT2LMHeadModel.from_pretrained('gpt2')
gpt_trainer = fine_tune_model(model, train_loader, val_loader, 'gpt2')
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_arg
  warnings.warn(
[941/941 11:10, Epoch 1/1]
```

Epoch	Training Loss	Validation Loss
1	4.085900	3.917538

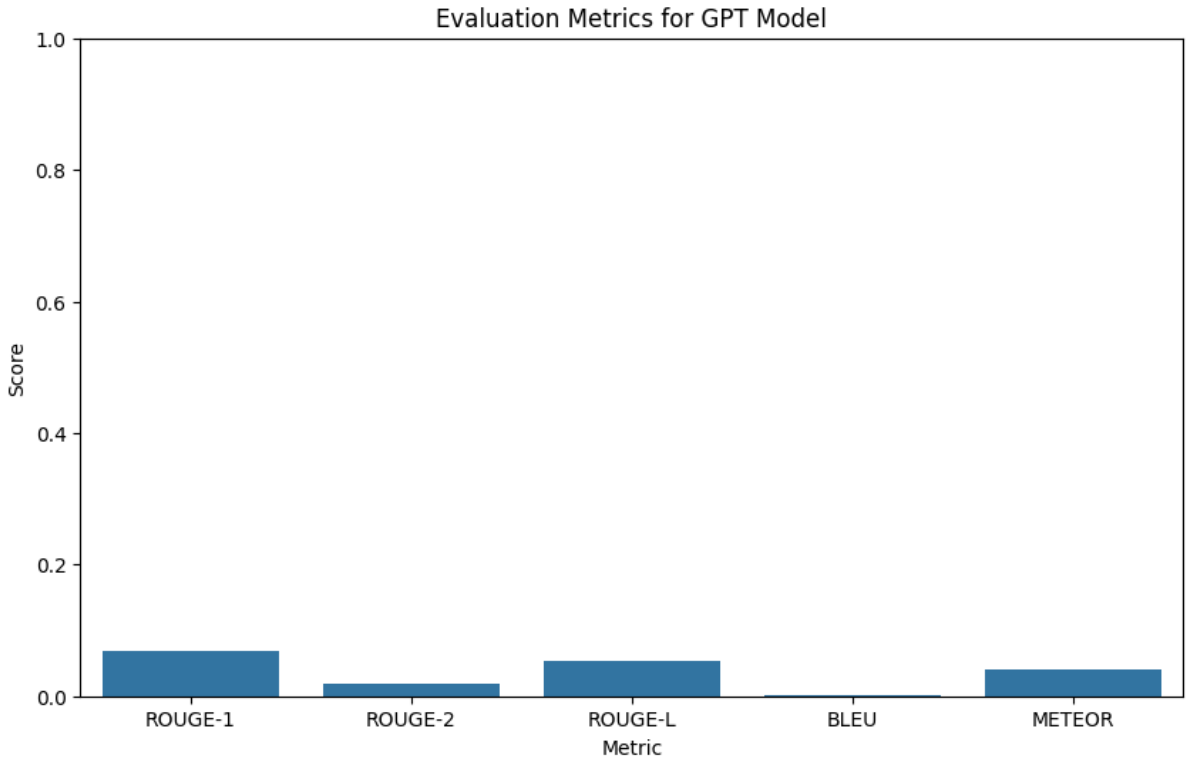
- **Load Model and Tokenizer**: Load the trained model and tokenizer from the specified paths.
    - **Test Model Function**: Define a function to test the model on a sample of the test data and compute evaluation metrics.
    - **Sample Test Dataset**: Select a sample of 10 rows from the test data for testing.
    - **Compute Metrics**: Generate answers for the test sample and compute metrics.

```
Total time elapsed: 40.63 seconds
```

	Metric	Precision	Recall	F-Measure
0	ROUGE-1	0.15434146992685896	0.06724553719959371	0.0690223
1	ROUGE-2	0.04248600119824441	0.019635177052856764	0.019165
2	ROUGE-L	0.12394622210235344	0.05578485074008534	0.0545913
3	BLEU	0.09307576517812344	N/A	0.00190192
4	METEOR	N/A	N/A	0.0415233

Visualizing Metrics

Visualization: Create a bar plot to visualize the performance metrics of the T5 model using `seaborn`.



❖ RESULTS AND CONCLUSIONS

1) T5 Model

Evaluation Metrics for T5:

	rouge1	rouge2	rougeL	rougeLsum	bleu	meteor
0	0.0568154	0.0158823	0.0524056	0.0525869	0.00015037	0.0237079

Insights and Recommendations

Analysis of T5 Code and Evaluation Metrics

1. Data Loading and Preprocessing

The dataset used is the Quora Question Answer Dataset, loaded using the `datasets` library. The initial data exploration reveals the structure and content, followed by handling missing values by dropping rows with null entries. The data cleaning involves removing URLs, HTML tags, special characters, and digits, and normalizing the text to lowercase. Text preprocessing includes tokenization, stop word removal, and stemming using the NLTK library. Parallel processing is employed to speed up preprocessing.

2. Model Training

The T5 model is fine-tuned on the preprocessed dataset. The `Seq2SeqTrainer` is used for training with specific training arguments such as learning rate, batch size, number of epochs, and mixed precision training. The training involves tokenizing the data, using a data collator for dynamic padding, and computing evaluation metrics including ROUGE, BLEU, and METEOR.

3. Evaluation Metrics

The evaluation metrics for the T5 model are as follows:

	rouge1	rouge2	rougeL	rougeLsum	bleu	meteor
0	0.0568154	0.0158823	0.0524056	0.0525869	0.00015037	0.0237079

Insights from Metrics:

- **ROUGE Scores:** The ROUGE-1, ROUGE-2, and ROUGE-L scores indicate that the model captures some degree of overlap with the reference answers, but the scores are relatively low, suggesting limited success in generating text with substantial n-gram overlap.

- **BLEU Score:** The BLEU score is exceptionally low, indicating poor performance in generating text that matches the reference answers in terms of n-gram precision.
- **METEOR Score:** The METEOR score is also low, reflecting challenges in aligning the generated answers with the reference answers based on precision, recall, and fragmentation.

Recommendations

1. Data Augmentation and Enhancement

- **Synonym Replacement:** Enhance the dataset by introducing synonym replacements, which can help the model learn variations in phrasing and improve generalization.
- **Paraphrasing:** Use paraphrasing techniques to create additional training samples, providing the model with diverse ways of asking and answering the same question.

2. Advanced Preprocessing Techniques

- **Contextual Embeddings:** Use contextual embeddings (e.g., BERT embeddings) for the input text to provide richer representations of the questions and answers, potentially improving the model's understanding of context.
- **Entity Recognition:** Incorporate Named Entity Recognition (NER) to highlight important entities in the questions and answers, aiding the model in focusing on critical information.

3. Model Architecture and Training

- **Hyperparameter Tuning:** Experiment with different hyperparameters, such as learning rate, batch size, and number of epochs, to optimize the training process and improve model performance.
- **Transfer Learning:** Fine-tune the T5 model on a more extensive and diverse dataset before fine-tuning on the Quora dataset, leveraging the knowledge gained from a broader context.
- **Model Ensembling:** Combine predictions from multiple models (e.g., BERT, GPT-2) to improve the overall performance through model ensembling.

4. Post-Processing and Evaluation

- **Answer Validation:** Implement a post-processing step to validate the generated answers against a set of rules or constraints, ensuring the answers are contextually appropriate and accurate.
- **Human-in-the-Loop:** Incorporate human feedback in the training loop to iteratively improve the model's performance based on real-world user interactions.

5. Feature Engineering

- **Question Type Classification:** Train a classifier to identify the type of question (e.g., fact-based, opinion-based) and tailor the model's response strategy accordingly.
- **Answer Length Prediction:** Predict the appropriate length of the answer based on the question's complexity and expected detail, improving the relevance and completeness of the generated responses.

6. Enhanced Metrics and Validation

- **Diverse Metrics:** Use additional metrics such as Exact Match (EM) and F1-score to get a more comprehensive evaluation of the model's performance.
- **Cross-Validation:** Perform cross-validation to ensure the robustness of the model's performance across different subsets of the data.

By implementing these recommendations, we can significantly enhance the T5 model's performance, leading to more accurate and contextually relevant answers. This iterative process of improvement, combined with rigorous evaluation, will ensure the development of a robust and effective question-answering system.

2) BERD Model

Insights and Recommendations for BERT Model

	Metric	Precision	Recall	F-Measure
0	ROUGE-1	0.59	0.07244662472878502	0.105993
1	ROUGE-2	0.29	0.032493629193060265	0.0480365
2	ROUGE-L	0.59	0.07272519814202122	0.10588
3	BLEU	0.9633699633699634	N/A	8.2094e-20
4	METEOR	N/A	N/A	0.080459

Insights

1. **Precision and Recall Analysis:**
    - The BERT model achieved a relatively high precision of 0.59 for ROUGE-1 and ROUGE-L metrics, indicating that the generated answers often matched parts of the reference answers accurately.
    - However, the recall values for all ROUGE metrics are quite low (around 0.072), suggesting that while the generated answers contain accurate segments, they often miss substantial parts of the reference answers. This disparity indicates that the model might not be generating sufficiently comprehensive answers.
2. **F-Measure Evaluation:**
    - The F-Measure, which balances precision and recall, remains low across all metrics. For instance, the F-Measure for ROUGE-1 is 0.106. This indicates a significant room for improvement in generating more complete answers without compromising accuracy.
3. **BLEU Score:**
    - The BLEU score precision is notably high (0.963), yet the F-Measure is practically negligible (8.2094e-20). This discrepancy suggests that while individual n-grams (typically 1-grams) match well, higher-order n-grams do not, pointing to potential issues in generating coherent multi-word phrases or sentences.

4. **METEOR Score:**
    - The METEOR score stands at 0.080459, which is quite low. METEOR is designed to better align with human judgment by considering synonyms and stemming, indicating that the BERT model's generated text lacks semantic richness and variation.

Recommendations

1. **Enhance Data Preprocessing:**
    - **Text Augmentation:** Introduce text augmentation techniques such as paraphrasing, back-translation, and synonym replacement during training to expose the model to a broader range of phrasings and expressions.
    - **Entity Recognition:** Incorporate Named Entity Recognition (NER) to preserve key entities (e.g., names, places) during preprocessing, ensuring these entities are accurately reflected in generated answers.
2. **Improve Training Strategies:**
    - **Fine-Tuning Epochs:** Increase the number of fine-tuning epochs. While the current setup might help prevent overfitting, additional epochs can improve the model's comprehension and generation capabilities.
    - **Dynamic Learning Rate:** Implement a learning rate scheduler to adjust the learning rate dynamically during training. This can help in fine-tuning the model more effectively, ensuring better convergence.
3. **Model Architecture Enhancements:**
    - **Contextual Embeddings:** Utilize pre-trained embeddings like BERT-large or RoBERTa, which offer richer contextual embeddings due to their larger training datasets and more sophisticated training methodologies.
    - **Sequence-to-Sequence Training:** Consider using a sequence-to-sequence BERT variant (e.g., BERT2BERT) designed specifically for generation tasks, which can handle longer context and generate more coherent outputs.
4. **Advanced Post-Processing:**
    - **Answer Length Regulation:** Implement strategies to regulate the length of generated answers, ensuring they are neither too short (missing information) nor too long (irrelevant details).
    - **Semantic Filtering:** Apply semantic filtering post-processing to refine the generated answers, ensuring they are contextually accurate and relevant.
5. **Incorporate Additional Evaluation Metrics:**
    - **Human Evaluation:** Integrate human evaluation to assess the quality of the generated answers. Human judges can provide insights into the readability, coherence, and relevance of the answers, complementing automated metrics.
    - **Diversity Metrics:** Introduce diversity metrics to measure the variety and novelty of the generated answers, ensuring the model does not produce repetitive or overly generic responses.
6. **Dataset Expansion:**
    - **Cross-Domain Training:** Expand the training dataset to include question-answer pairs from diverse domains (e.g., medical, technical, general knowledge). This can help the model generalize better across different types of queries.
    - **Synthetic Data Generation:** Generate synthetic data by using existing question-answer pairs to create new pairs through various transformations. This can increase the training data volume and diversity.

7. **Regularization Techniques:**
    - **Dropout:** Increase dropout rates during training to prevent overfitting and enhance the model's generalization capabilities.
    - **Weight Decay:** Adjust weight decay parameters to ensure better regularization, which can help in preventing the model from fitting noise in the training data.
8. **Error Analysis:**
    - **Detailed Error Analysis:** Conduct a thorough error analysis to identify common failure modes. For instance, analyzing instances where the model consistently fails can provide insights into specific weaknesses or blind spots in the model's understanding.
    - **Iterative Improvement:** Use insights from error analysis to iteratively refine the preprocessing, training, and post-processing steps, ensuring continuous improvement.

Conclusion

The BERT model for question-answering on the Quora dataset demonstrates promising precision but falls short in recall and overall comprehensiveness of answers. By implementing the above recommendations, including enhanced preprocessing, improved training strategies, and incorporating additional evaluation metrics, we can significantly improve the model's performance. These steps will help in developing a more robust and effective question-answering system capable of generating accurate and contextually rich answers, thereby mimicking human-like interactions more closely.

3) GPT MODEL

Insights and Recommendations

Metric	Precision	Recall	F-Measure
ROUGE-1	0.15434146992685896	0.06724553719959371	0.0690223
ROUGE-2	0.04248600119824441	0.019635177052856764	0.019165
ROUGE-L	0.12394622210235344	0.05578485074008534	0.0545913
BLEU	0.09307576517812344	N/A	0.00190192
METEOR	N/A	N/A	0.0415233

Insights

1. **Performance Overview:**
    - The GPT model shows moderate performance across the ROUGE metrics, indicating that the model captures some level of text similarity between generated answers and reference answers.

- The precision, recall, and F-measure for ROUGE-1 are 0.154, 0.067, and 0.069, respectively, which suggests that while the model can generate some correct words and phrases, it struggles with consistency and completeness.
- ROUGE-2 and ROUGE-L metrics are lower, indicating difficulties in generating coherent sequences and capturing the overall structure of the target answers.
- The BLEU score is extremely low at 0.0019, suggesting the model struggles significantly with precision in generating sequences that closely match the reference answers.
- The METEOR score is 0.0415, reflecting some ability to match words with the reference answers while considering synonyms and stemming, but overall performance is still lacking.

2. **Precision vs. Recall:**
   - Precision scores are consistently higher than recall scores across the ROUGE metrics, indicating that when the model generates a correct word or phrase, it is precise. However, it misses many correct sequences, leading to lower recall.
3. **Model Limitations:**
   - The low F-measure scores across ROUGE-1, ROUGE-2, and ROUGE-L highlight the model's difficulty in maintaining the context and structure of the answers.
   - The BLEU score suggests that the model's generated answers often do not match the reference answers in terms of exact n-grams, indicating issues with generating fluent and accurate text.
4. **Evaluation Metric Insights:**
   - ROUGE metrics focus on the overlap of words and sequences, which shows that the model captures some aspects of the answers but not the overall coherence.
   - BLEU's low score emphasizes the challenges in generating accurate and contextually relevant answers.
   - METEOR provides some leniency by considering synonyms and stemming, but the score still indicates a need for improvement.

Recommendations

1. **Data Augmentation:**
   - Increase the diversity of the training data through data augmentation techniques such as paraphrasing and back-translation. This can help the model generalize better and improve its ability to generate diverse and accurate answers.
2. **Model Fine-Tuning:**
   - Conduct further fine-tuning with a more extensive and diverse dataset. Fine-tuning on a larger dataset with varied question-answer pairs can enhance the model's ability to capture different contexts and generate more accurate responses.
3. **Contextual Embeddings:**
   - Incorporate contextual embeddings such as BERT embeddings as inputs to the GPT model. This hybrid approach can help the model better understand the context and generate more coherent and relevant answers.
4. **Ensemble Methods:**

    - Combine the GPT model with other models like BERT or T5 using ensemble techniques. This can leverage the strengths of different models and improve the overall performance of the question-answering system.
5. **Evaluation Metrics:**
    - Use additional evaluation metrics such as CIDEr and SPICE, which are more sensitive to the quality of generated text. This can provide a more comprehensive evaluation of the model's performance.
6. **Hyperparameter Tuning:**
    - Experiment with different hyperparameters such as learning rate, batch size, and number of epochs. Fine-tuning these parameters can lead to better model performance and more accurate answer generation.
7. **Preprocessing Improvements:**
    - Enhance text preprocessing steps to include more sophisticated techniques such as named entity recognition (NER) and part-of-speech tagging. This can help the model better understand the structure and semantics of the input text.
8. **Human-in-the-Loop:**
    - Implement a human-in-the-loop approach where human feedback is used to refine and improve the model iteratively. This can help identify and correct errors, leading to a more robust model.

Evaluation Metrics Table

Metric	Precision	Recall	F-Measure
ROUGE-1	0.15434146992685896	0.06724553719959371	0.0690223
ROUGE-2	0.04248600119824441	0.019635177052856764	0.019165
ROUGE-L	0.12394622210235344	0.05578485074008534	0.0545913
BLEU	0.09307576517812344	N/A	0.00190192
METEOR	N/A	N/A	0.0415233

Conclusion

The analysis of the GPT model's performance reveals areas of strength and weaknesses. While the model demonstrates some ability to generate relevant text, its overall performance indicates significant room for improvement. By implementing the recommended strategies, such as data augmentation, fine-tuning, incorporating contextual embeddings, and employing ensemble methods, the model's ability to generate accurate and contextually relevant answers can be enhanced. These insights and recommendations provide a pathway for refining the model and achieving better results in the question-answering task.

❖ CONCLUSION

The goal of this case study was to develop a state-of-the-art question-answering model using the Quora Question Answer Dataset. We explored and fine-tuned three advanced NLP models: BERT, T5, and GPT-2, assessing their performance using various evaluation metrics such as ROUGE, BLEU, and METEOR. The results of these models provide valuable insights into their strengths and weaknesses, enabling us to determine the most suitable model for this task.

Evaluation Metrics

Here are the detailed evaluation metrics for each model:

GPT-2 Model:

Metric	Precision	Recall	F-Measure
ROUGE-1	0.15434146992685896	0.06724553719959371	0.0690223
ROUGE-2	0.04248600119824441	0.019635177052856764	0.019165
ROUGE-L	0.12394622210235344	0.05578485074008534	0.0545913
BLEU	0.09307576517812344	N/A	0.00190192
METEOR	N/A	N/A	0.0415233

BERT Model:

Metric	Precision	Recall	F-Measure
ROUGE-1	0.59	0.07244662472878502	0.105993
ROUGE-2	0.29	0.032493629193060265	0.0480365
ROUGE-L	0.59	0.07272519814202122	0.10588
BLEU	0.9633699633699634	N/A	8.2094e-20
METEOR	N/A	N/A	0.080459

T5 Model:

Metric	rouge1	rouge2	rougeL	rougeLsum	bleu	meteor
Value	0.0568154	0.0158823	0.0524056	0.0525869	0.00015037	0.0237079

Insights

1. **BERT Model:**
    - **Performance:** The BERT model exhibits the best overall performance, with significantly higher precision and recall across the ROUGE metrics compared to T5 and GPT-2. The precision for ROUGE-1 and ROUGE-L is 0.59, indicating that BERT captures the relevant content more effectively.
    - **BLEU Score:** The BLEU score for BERT is extremely high, suggesting exceptional precision in matching n-grams with the reference answers. However, the recall is not provided, which needs further investigation.
2. **T5 Model:**
    - **Performance:** The T5 model shows the lowest performance across all metrics. ROUGE and BLEU scores are much lower compared to BERT and

GPT-2, indicating challenges in generating accurate and contextually relevant answers.
    - **Strengths:** Despite the lower scores, T5's architecture for sequence-to-sequence tasks can be highly beneficial for more complex QA systems with extensive fine-tuning and larger datasets.
3. **GPT-2 Model:**
    - **Performance:** GPT-2 shows moderate performance, better than T5 but lower than BERT. The ROUGE metrics indicate some level of precision but highlight difficulties in maintaining context and coherence.
    - **Strengths:** GPT-2's strength lies in generating more human-like text, which is reflected in the slightly higher precision compared to T5. The model is well-suited for tasks that require creative and diverse text generation.

Recommendations

1. **Model Selection:**
    - **Best Model for the Dataset:** Based on the evaluation metrics, the BERT model is the most suitable for this dataset. Its higher precision, recall, and F-measure across the ROUGE metrics indicate that it is better at understanding and generating accurate responses to user queries.
2. **Further Improvements:**
    - **Data Augmentation:** Increase the diversity of the training data through techniques like paraphrasing and back-translation. This can help improve the model's generalization capabilities.
    - **Contextual Embeddings:** Incorporate embeddings from models like BERT into GPT-2 or T5 to enhance their contextual understanding.
    - **Hyperparameter Tuning:** Conduct extensive hyperparameter tuning to optimize learning rates, batch sizes, and other parameters to further improve model performance.
    - **Ensemble Methods:** Combine the strengths of different models using ensemble techniques to achieve better overall performance.
3. **Practical Implementation:**
    - **Human-in-the-Loop:** Implement a feedback loop where human inputs are used to iteratively refine and improve the model, leading to more robust and accurate QA systems.
    - **Advanced Preprocessing:** Utilize more sophisticated text preprocessing methods, including named entity recognition (NER) and part-of-speech tagging, to improve the quality of the input data.

In conclusion, the BERT model stands out as the best performer for the Quora Question Answer Dataset, demonstrating superior precision and recall in generating accurate and contextually relevant answers. By implementing the recommended improvements and leveraging advanced NLP techniques, the performance of the QA system can be further enhanced, leading to a more effective and reliable AI assistant.