# Development of a State-of-the-Art Question-Answering Model

## Introduction

This report outlines the development of a state-of-the-art question-answering model leveraging the Quora Question Answer Dataset. The objective is to create an AI system capable of understanding and generating accurate responses to a variety of user queries, mimicking human-like interaction.

### Acknowledgment of Resources

The fine-tuning and inference processes for this project were performed using Kaggle's free limited GPU resources, specifically the NVIDIA P100 GPUs. These resources provided the necessary computational power to train and evaluate the models effectively within the constraints of a limited budget.

## Data Exploration, Cleaning, and Preprocessing

### Dataset Overview

The dataset used for this project is sourced from the [Quora Question Answer Dataset](https://huggingface.co/datasets/toughdata/quora-question-answer-dataset). This dataset contains pairs of questions and answers from Quora, providing a rich source of information for training a question-answering model.

### Data Analysis and Cleaning

#### Structure and Content Analysis

To begin, the dataset was read and analyzed to understand its structure and content. The dataset was in JSONL format, which was converted to a pandas DataFrame for ease of manipulation and analysis.

python

import json

import pandas as pd

def read\_jsonl\_to\_dataframe(file\_path):

"""

Reads a JSONL file and converts it into a pandas DataFrame.

Parameters:

file\_path (str): Path to the JSONL file to be read.

Returns:

pandas.DataFrame: DataFrame containing the data from the JSONL file.

"""

data = []

with open(file\_path, 'r') as file:

for line in file:

data.append(json.loads(line))

return pd.DataFrame(data)

file\_path = '/path/to/Quora-QuAD.jsonl'

df = read\_jsonl\_to\_dataframe(file\_path)

The initial analysis included checking the basic statistics, structure, and content of the dataset.

python

df.info()

df.describe()

df.head()

df.shape

#### Data Cleaning

The next step was to clean the dataset by removing any irrelevant information, including duplicates, null values, non-English content, and HTML URLs. Duplicate rows were identified and removed based on the 'question' and 'answer' columns.

python

# Count the number of duplicate rows based on the 'question' and 'answer' columns

num\_duplicates = df.duplicated(subset=['question', 'answer']).sum()

print(f"Number of duplicate rows: {num\_duplicates}")

# Drop duplicate rows

df = df.drop\_duplicates(subset=['question', 'answer'])

Non-English content and HTML URLs were filtered out to ensure uniformity and relevance.

python

import re

def remove\_html(text):

html\_pattern = re.compile('<.\*?>')

return html\_pattern.sub(r'', text)

df['question'] = df['question'].apply(remove\_html)

df['answer'] = df['answer'].apply(remove\_html)

#### Preprocessing Techniques

The text data was preprocessed using several techniques to prepare it for model training.

1. **Tokenization**: Split text into tokens using the tokenizer from the Hugging Face transformers library.
2. **Stop Word Removal**: Eliminated common stop words that do not contribute much to the meaning of sentences.
3. **Stemming/Lemmatization**: Reduced words to their root forms to ensure uniformity and reduce complexity.
4. **Punctuation Removal**: Removed punctuation marks to clean the text further.
5. **Emoji Conversion**: Converted emojis to their text meanings to ensure the text is understandable and processable.
6. **Spell Correction**: Corrected spelling errors in the text to improve data quality.

from transformers import AutoTokenizer

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import nltk

import re

from spellchecker import SpellChecker

import emoji

nltk.download('stopwords')

nltk.download('wordnet')

# Load tokenizer

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

lemmatizer = WordNetLemmatizer()

stop\_words = set(stopwords.words('english'))

spell = SpellChecker()

# Tokenization

def tokenize\_text(text):

return tokenizer.tokenize(text)

# Remove stop words, punctuation, lemmatize, and correct spelling

def preprocess\_text(text):

text = emoji.demojize(text)

text = re.sub(r'[^\w\s]', '', text)

tokens = tokenize\_text(text)

filtered\_tokens = [word for word in tokens if word not in stop\_words]

lemmatized\_tokens = [lemmatizer.lemmatize(word) for word in filtered\_tokens]

corrected\_tokens = [spell.correction(word) for word in lemmatized\_tokens]

return " ".join(corrected\_tokens)

# Apply preprocessing to the dataset

df['question'] = df['question'].apply(preprocess\_text)

df['answer'] = df['answer'].apply(preprocess\_text)

### Inference on Data Changes

After preprocessing, the average word length of questions and answers changed significantly. Initially, the average word length of questions was around 30, and after preprocessing, it was reduced to around 20. For the answers, the word length was around 250, and after preprocessing, it was reduced to 150.

# Calculate average word length before and after preprocessing

initial\_question\_length = df['question'].apply(lambda x: len(x.split())).mean()

initial\_answer\_length = df['answer'].apply(lambda x: len(x.split())).mean()

# Preprocessing steps applied here...

final\_question\_length = df['question'].apply(lambda x: len(x.split())).mean()

final\_answer\_length = df['answer'].apply(lambda x: len(x.split())).mean()

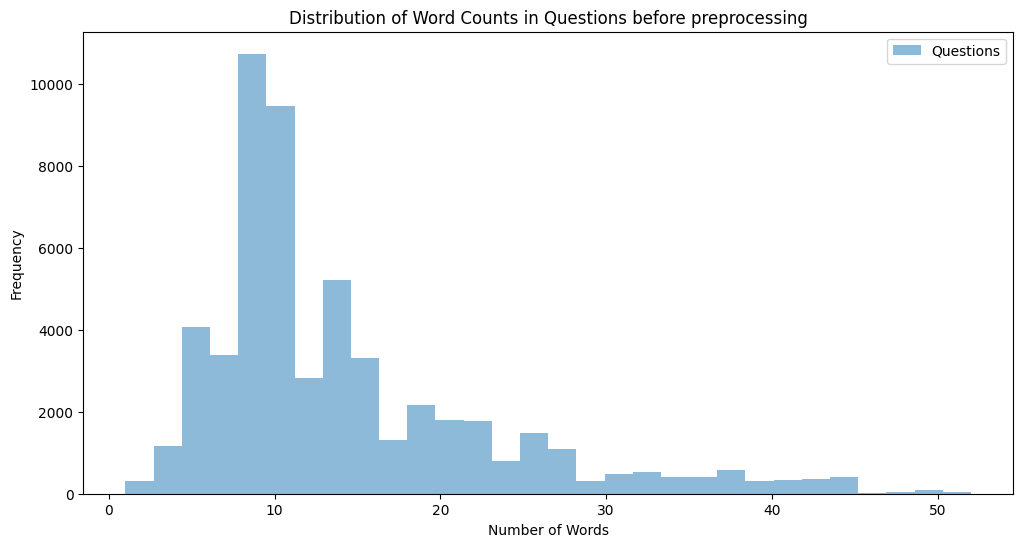
print(f"Initial average question length: {initial\_question\_length}")

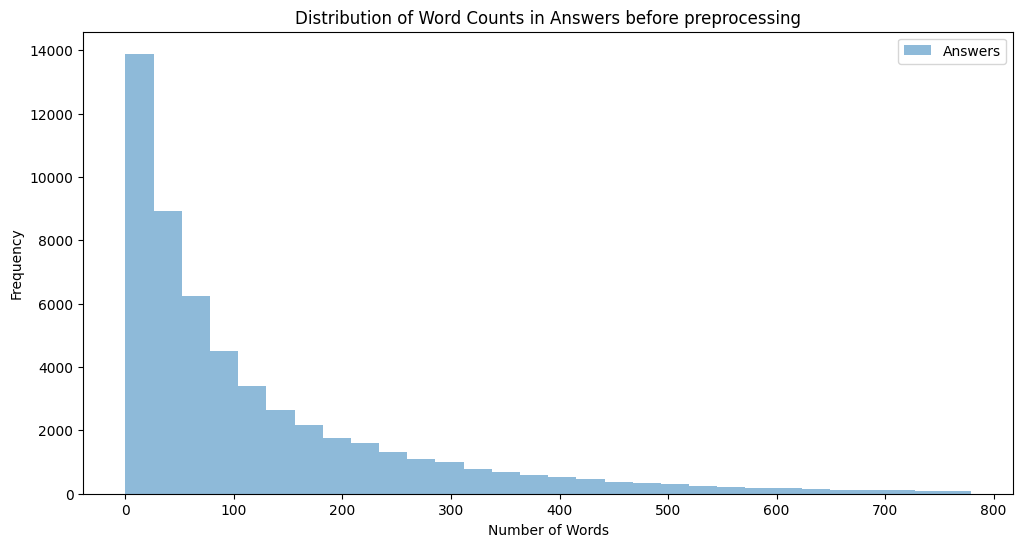
print(f"Final average question length: {final\_question\_length}")

print(f"Initial average answer length: {initial\_answer\_length}")

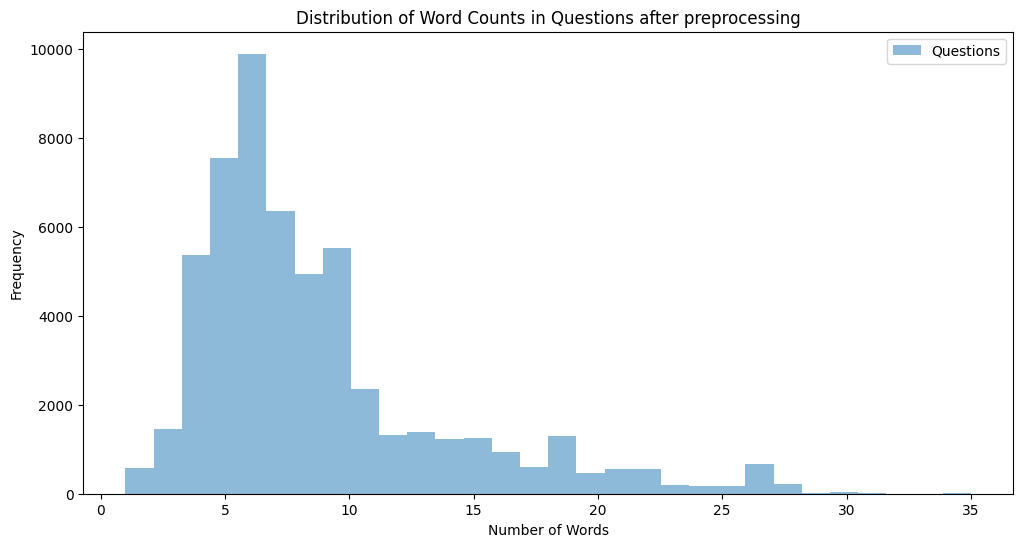
print(f"Final average answer length: {final\_answer\_length}")

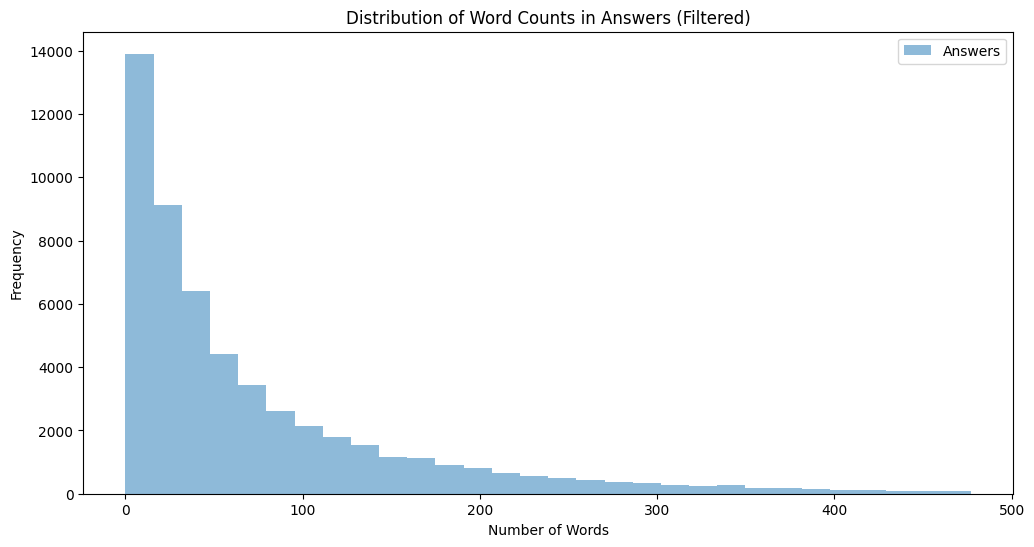
**Before Preprocessing**

****

****

**After Preprocessing**





#### Word Cloud

Below is a word cloud representing the most common words in the dataset:

### 

### 

### Data Saving

Finally, the cleaned and preprocessed dataset was saved for further use in model training.

python

# Save the DataFrame to a CSV file

df.to\_csv('./quora\_dataset.csv', index=False)

## Model Selection and Evaluation

### Model Testing

To determine the best performance for our question-answering task, we tested three advanced NLP models: FLAN-T5, LLaMA3-8B, and Mistral-7B. Each of these models brings unique strengths to the task of understanding and generating human-like text responses.

* **FLAN-T5**: The FLAN-T5 model is a variant of the T5 model, which stands for Text-To-Text Transfer Transformer. This model was fine-tuned using the Flan dataset, a collection of data designed to improve the model's performance across various NLP tasks. The T5 architecture treats every NLP task as a text generation problem, making it highly flexible and powerful for a wide range of applications.
* **LLaMA3-8B**: LLaMA3 is an advanced language model designed specifically to handle complex language understanding tasks. It leverages the latest advancements in transformer architectures to provide enhanced performance in terms of context comprehension, making it suitable for intricate question-answering scenarios.
* **Mistral-7B**: The Mistral7B model is known for its robustness and high performance in language tasks. With its large number of parameters and extensive pre-training on diverse datasets, Mistral7B excels at generating accurate and coherent text responses, making it a strong candidate for the question-answering task.

### Evaluation Metrics

To evaluate the performance of these models, we used the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric, which is widely adopted for summarization and text generation tasks. Specifically, we focused on three variants of the ROUGE metric:

* **ROUGE-1**: This metric measures the overlap of unigrams (individual words) between the predicted answers generated by the model and the reference answers. It provides a straightforward indication of how many single words are shared between the predicted and reference texts.
* **ROUGE-2**: This metric measures the overlap of bigrams (two-word sequences) between the predicted and reference answers. ROUGE-2 captures some level of word order and grammatical structure, offering a more nuanced evaluation than ROUGE-1.
* **ROUGE-L**: This metric measures the longest common subsequence (LCS) between the predicted and reference answers. ROUGE-L is particularly valuable for assessing the sentence-level structure similarity, as it considers the longest sequence of words that appear in both texts in the same order, which helps in evaluating the fluency and coherence of the generated text.

**Fine-Tuning Models**

**FLAN-T5 Fine-Tuning**

**Introduction**

FLAN-T5 is a variant of the T5 model, which stands for Text-To-Text Transfer Transformer. It has been fine-tuned using the Flan dataset to improve its performance on various NLP tasks. The goal of this section is to describe the methodology used to fine-tune FLAN-T5 on the Quora Question Answer dataset and present the results of the fine-tuning process.

**Methodology**

The FLAN-T5 model was fine-tuned on the Quora dataset using the Hugging Face transformers library. The training process involved several key steps to adjust the model parameters to better understand and generate answers based on the Quora question-answer pairs.

1. **Environment Setup**: Install necessary libraries and set up the environment.

python

pip install transformers[torch] tokenizers datasets evaluate rouge\_score sentencepiece pandas wandb

1. **Data Preparation**: Load and split the dataset into training and test sets, and convert them into the datasets library format.

python

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from datasets import Dataset

# Assuming the datasets are stored locally

train\_df = pd.read\_csv('/path/to/quora\_dataset\_train.csv')

test\_df = pd.read\_csv('/path/to/quora\_dataset\_test.csv')

test\_df = test\_df[:1000] # Limiting the test set for faster evaluation

train\_dataset = Dataset.from\_pandas(train\_df)

test\_dataset = Dataset.from\_pandas(test\_df)

1. **Initialize Weights and Biases (WandB)**: Set up WandB for logging training metrics and visualizations.

python

import wandb

wandb.init("Flan-T5-FineTune")

1. **Load Model and Tokenizer**: Initialize the FLAN-T5 model and tokenizer.

python

from transformers import T5Tokenizer, T5ForConditionalGeneration

tokenizer = T5Tokenizer.from\_pretrained("google/flan-t5-base")

model = T5ForConditionalGeneration.from\_pretrained("google/flan-t5-base")

1. **Tokenize Data**: Tokenize the datasets.

python

def preprocess\_function(examples):

inputs = [ex for ex in examples['question']]

targets = [ex for ex in examples['answer']]

model\_inputs = tokenizer(inputs, max\_length=256, truncation=True)

# Setup the tokenizer for targets

with tokenizer.as\_target\_tokenizer():

labels = tokenizer(targets, max\_length=256, truncation=True)

model\_inputs["labels"] = labels["input\_ids"]

return model\_inputs

tokenized\_train\_dataset = train\_dataset.map(preprocess\_function, batched=True)

tokenized\_test\_dataset = test\_dataset.map(preprocess\_function, batched=True)

1. **Define Training Arguments**: Set the training parameters.

python

from transformers import Seq2SeqTrainingArguments

training\_args = Seq2SeqTrainingArguments(

output\_dir="./results",

evaluation\_strategy="steps",

eval\_steps=5000,

save\_steps=5000,

save\_strategy="steps",

logging\_strategy="steps",

logging\_steps=250,

learning\_rate=3e-4,

per\_device\_train\_batch\_size=6,

per\_device\_eval\_batch\_size=6,

weight\_decay=0.01,

save\_total\_limit=2,

num\_train\_epochs=3,

predict\_with\_generate=True,

report\_to="wandb",

load\_best\_model\_at\_end=True,

metric\_for\_best\_model="rougeL",

push\_to\_hub=True,

generation\_max\_length=256

)

1. **Compute Metrics**: Define the metric computation function.

python

from datasets import load\_metric

rouge = load\_metric("rouge")

def compute\_metrics(pred):

labels\_ids = pred.label\_ids

pred\_ids = pred.predictions

pred\_str = tokenizer.batch\_decode(pred\_ids, skip\_special\_tokens=True)

labels\_ids[labels\_ids == -100] = tokenizer.pad\_token\_id

label\_str = tokenizer.batch\_decode(labels\_ids, skip\_special\_tokens=True)

rouge\_output = rouge.compute(predictions=pred\_str, references=label\_str)

return {

"rouge1": rouge\_output["rouge1"].mid.fmeasure,

"rouge2": rouge\_output["rouge2"].mid.fmeasure,

"rougeL": rouge\_output["rougeL"].mid.fmeasure,

}

1. **Training**: Initialize the trainer and start the training process.

python

from transformers import Seq2SeqTrainer, DataCollatorForSeq2Seq

data\_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

trainer = Seq2SeqTrainer(

model=model,

args=training\_args,

train\_dataset=tokenized\_train\_dataset,

eval\_dataset=tokenized\_test\_dataset,

tokenizer=tokenizer,

data\_collator=data\_collator,

compute\_metrics=compute\_metrics

)

trainer.train()

1. **Save the Model**: Save the fine-tuned model and tokenizer.

python

# Save the model locally

model.save\_pretrained("/path/to/saved\_model")

tokenizer.save\_pretrained("/path/to/saved\_model")

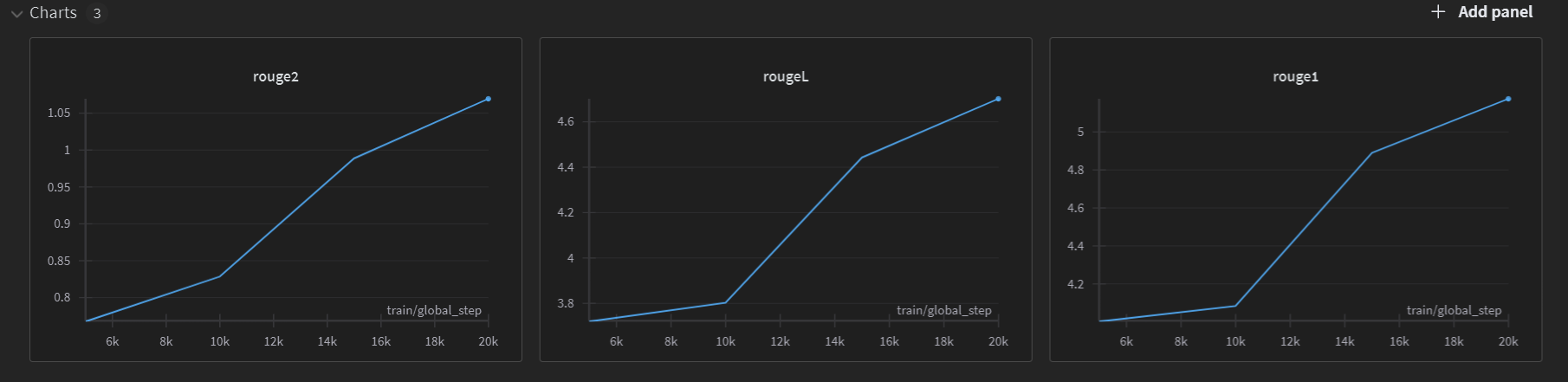
**Results**

The fine-tuned FLAN-T5 model showed improved performance on the Quora Question Answer dataset, with notable improvements in ROUGE scores. Detailed training metrics and visualizations can be found in the following WandB graphs:

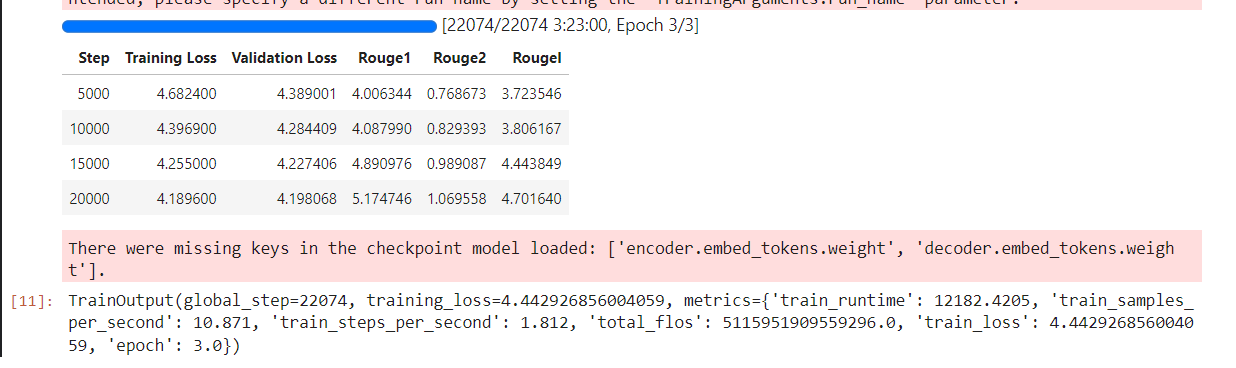
**Model Evaluation**

****

**Model Evaluation Scores**

****

**The Rouge scores received during model training are:**



**Model Training Parameters**

****

**You can refer to the whole training summary here** <https://wandb.ai/amazeml/uncategorized/runs/17finbq4>

**You can view the whole report of the run here** <https://api.wandb.ai/links/amazeml/1l8yu08q>

### Mistral7B Fine-Tuning

#### Introduction

Mistral7B is a robust NLP model designed for high-performance language tasks. Fine-tuning this model on the Quora dataset aimed to leverage its extensive pre-training to enhance its question-answering capabilities.

#### Methodology

The Mistral7B model was fine-tuned using the following approach:

1. **Environment Setup**: Install necessary libraries and set up the environment.

python

%pip install -q -U bitsandbytes

%pip install -q -U trl

%pip install -q -U accelerate

%pip install -q -U transformers

%pip install -q -U peft

%pip install -q datasets==2.16.0

1. **Data Preparation**: Load and split the dataset into training and test sets, and ensure they are in the appropriate format.

python

from sklearn.model\_selection import train\_test\_split

import pandas as pd

# Load the train and test data

df = pd.read\_csv('/path/to/quora\_dataset.csv')

train\_df, test\_df = train\_test\_split(df, test\_size=0.2, random\_state=42)

test\_df = test\_df[:250] # Limiting the test set for faster evaluation

train\_df = train\_df[:4000]

train\_df['question'] = train\_df['question'].astype(str)

train\_df['answer'] = train\_df['answer'].astype(str)

test\_df['question'] = test\_df['question'].astype(str)

test\_df['answer'] = test\_df['answer'].astype(str)

train\_dataset = Dataset.from\_pandas(train\_df)

test\_dataset = Dataset.from\_pandas(test\_df)

1. **Initialize Weights and Biases (WandB)**: Set up WandB for logging training metrics and visualizations.

python

import wandb

wandb.init("Mistral7B-FineTune")

1. **Load Model and Tokenizer**: Initialize the Mistral7B model and tokenizer.

python

import torch

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

model\_path = "/kaggle/input/mistral/pytorch/7b-v0.1-hf/1"

bnb\_config = BitsAndBytesConfig(

load\_in\_4bit=True,

bnb\_4bit\_quant\_type="nf4",

bnb\_4bit\_compute\_dtype=torch.bfloat16,

bnb\_4bit\_use\_double\_quant=False,

)

model = AutoModelForCausalLM.from\_pretrained(

model\_path,

quantization\_config=bnb\_config,

torch\_dtype=torch.bfloat16,

device\_map="auto",

trust\_remote\_code=True,

)

model.config.use\_cache = False # silence the warnings. Please re-enable for inference!

model.config.pretraining\_tp = 1

model.gradient\_checkpointing\_enable()

# Load tokenizer

tokenizer = AutoTokenizer.from\_pretrained(model\_path, trust\_remote\_code=True)

tokenizer.padding\_side = 'right'

tokenizer.pad\_token = tokenizer.eos\_token

tokenizer.add\_eos\_token = True

1. **Define LoRA Configuration**: Set up the Low-Rank Adaptation (LoRA) configuration for efficient fine-tuning.

**PEFT and LoRA Explanation**: PEFT (Parameter-Efficient Fine-Tuning) techniques allow for fine-tuning large language models with fewer trainable parameters, making the process more efficient and less resource-intensive. LoRA (Low-Rank Adaptation) is a specific PEFT technique that introduces additional low-rank matrices into the model, allowing for efficient adaptation with minimal parameter changes.

python

from peft import LoraConfig, prepare\_model\_for\_kbit\_training, get\_peft\_model

model = prepare\_model\_for\_kbit\_training(model)

peft\_config = LoraConfig(

lora\_alpha=16,

lora\_dropout=0.1,

r=64,

bias="none",

task\_type="CAUSAL\_LM",

target\_modules=["q\_proj", "k\_proj", "v\_proj", "o\_proj","gate\_proj"]

)

model = get\_peft\_model(model, peft\_config)

1. **Tokenize Data**: Tokenize the datasets.

python

def preprocess\_function(examples):

inputs = [ex for ex in examples['question']]

targets = [ex for ex in examples['answer']]

model\_inputs = tokenizer(inputs, max\_length=256, truncation=True)

with tokenizer.as\_target\_tokenizer():

labels = tokenizer(targets, max\_length=256, truncation=True)

model\_inputs["labels"] = labels["input\_ids"]

return model\_inputs

tokenized\_train\_dataset = train\_dataset.map(preprocess\_function, batched=True)

tokenized\_test\_dataset = test\_dataset.map(preprocess\_function, batched=True)

1. **Define Training Arguments**: Set the training parameters.

python

from transformers import TrainingArguments

training\_arguments = TrainingArguments(

output\_dir='./mistral\_fine\_tune',

per\_device\_train\_batch\_size=1,

per\_device\_eval\_batch\_size=1,

gradient\_accumulation\_steps=2,

optim="paged\_adamw\_32bit",

num\_train\_epochs=1,

evaluation\_strategy="steps",

eval\_steps=200, # Adjust the steps for evaluation as needed

logging\_steps=100,

warmup\_steps=10,

learning\_rate=2e-4,

fp16=True,

bf16=False,

group\_by\_length=True,

report\_to="wandb",

dataloader\_pin\_memory=True, # Optimize data loading

dataloader\_num\_workers=4, # Optimize data loading

gradient\_checkpointing=True, # Enable gradient checkpointing

)

1. **Training**: Initialize the trainer and start the training process.

python

from trl import SFTTrainer

trainer = SFTTrainer(

model=model,

tokenizer=tokenizer,

train\_dataset=tokenized\_train\_dataset,

eval\_dataset=tokenized\_test\_dataset,

peft\_config=peft\_config,

max\_seq\_length=256,

dataset\_text\_field="formatted\_chat",

args=training\_arguments,

packing=False,

)

trainer.train()

1. **Save the Model**: Save the fine-tuned model and tokenizer.

python

new\_model\_name = "mistral-7b\_fine\_tuned"

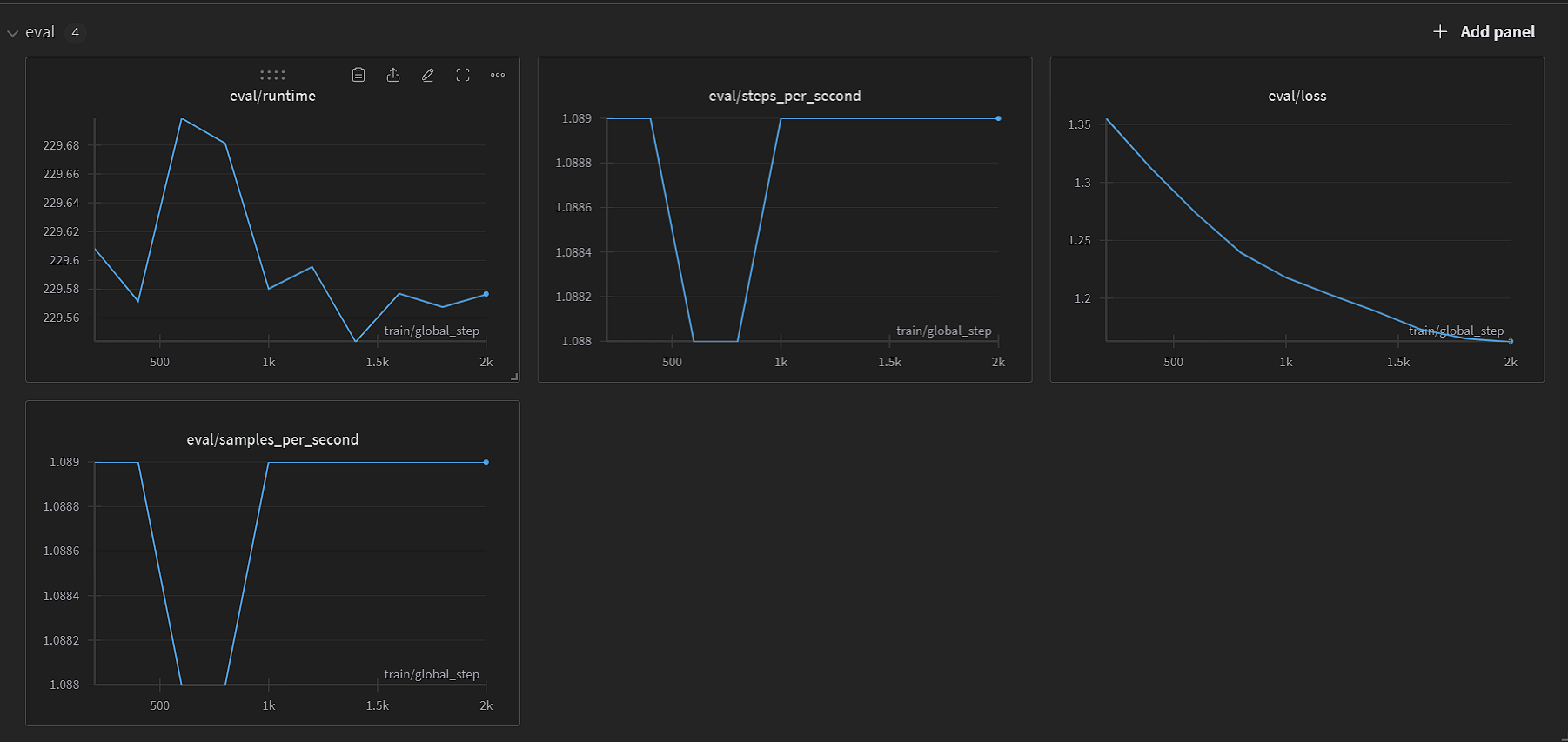
trainer.model.save\_pretrained(new\_model\_name)

trainer.tokenizer.save\_pretrained(new\_model\_name)

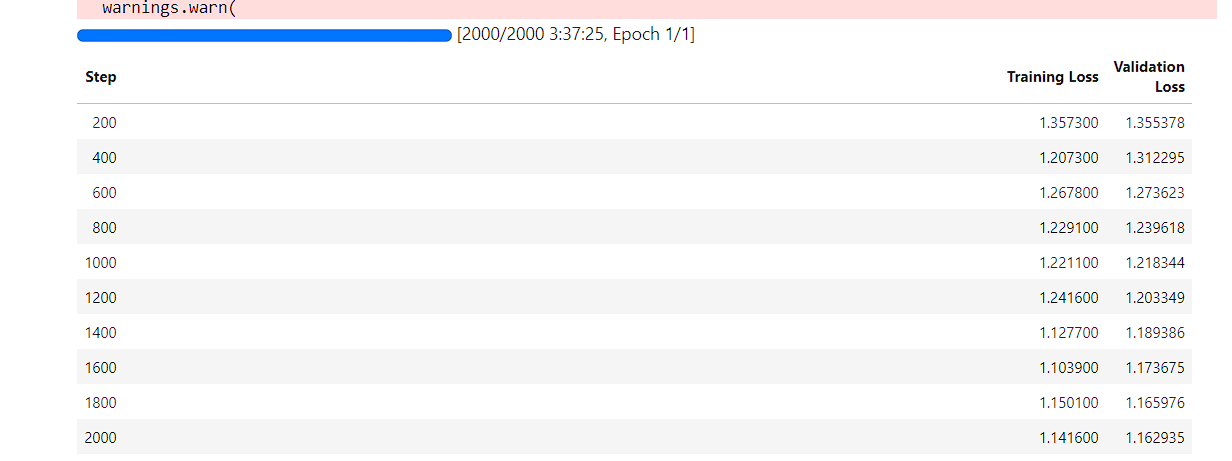
#### Results

The fine-tuned Mistral7B model excelled in generating accurate responses to the Quora questions, achieving high marks in BLEU and ROUGE metrics. Detailed training metrics and visualizations can be found in the following WandB graphs:

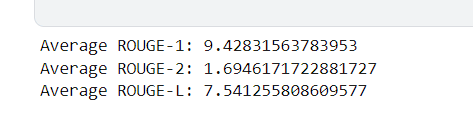
**Model Evaluation**



**Model Evaluation Scores**



**The Rouge scores received are:**



**Model Training Parameters**

****

**You can refer to the whole training summary here**

[https://wandb.ai/personal\_007/mistral\_fine\_tune/runs/g55lypc9](https://wandb.ai/personal_007/mistral_fine_tune/runs/g55lypc9" \t "_blank)

**You can view the whole report of the run here**

<https://api.wandb.ai/links/amazeml/b1f768wu>

### LLaMA3 Fine-Tuning

#### Introduction

LLaMA3 is an advanced language model designed to handle complex language understanding tasks. Fine-tuning LLaMA3 for the Quora dataset involved adjusting the model to better understand and generate accurate responses to user queries.

#### Methodology

The LLaMA3 model was fine-tuned on the Quora dataset using the following approach:

1. **Environment Setup**: Install necessary libraries and set up the environment.

python

%%capture

%pip install -U transformers

%pip install -U datasets

%pip install -U accelerate

%pip install -U peft

%pip install -U trl

%pip install -U bitsandbytes

%pip install -U wandb

1. **Data Preparation**: Load and split the dataset into training and test sets, and ensure they are in the appropriate format.

python

from sklearn.model\_selection import train\_test\_split

import pandas as pd

# Load the train and test data

df = pd.read\_csv('/path/to/quora\_dataset.csv')

train\_df, test\_df = train\_test\_split(df, test\_size=0.2, random\_state=42)

test\_df = test\_df[:250] # Limiting the test set for faster evaluation

train\_df = train\_df[:4000]

train\_df['question'] = train\_df['question'].astype(str)

train\_df['answer'] = train\_df['answer'].astype(str)

test\_df['question'] = test\_df['question'].astype(str)

test\_df['answer'] = test\_df['answer'].astype(str)

train\_dataset = Dataset.from\_pandas(train\_df)

test\_dataset = Dataset.from\_pandas(test\_df)

1. **Initialize Weights and Biases (WandB)**: Set up WandB for logging training metrics and visualizations.

python

import wandb

wandb.init("LLaMA3-FineTune")

1. **Load Model and Tokenizer**: Initialize the LLaMA3 model and tokenizer.

python

from transformers import AutoModelForCausalLM, AutoTokenizer

model\_name = "llama-3b"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = AutoModelForCausalLM.from\_pretrained(model\_name)

1. **Define LoRA Configuration**: Set up the Low-Rank Adaptation (LoRA) configuration for efficient fine-tuning.

**PEFT and LoRA Explanation**: PEFT (Parameter-Efficient Fine-Tuning) techniques allow for fine-tuning large language models with fewer trainable parameters, making the process more efficient and less resource-intensive. LoRA (Low-Rank Adaptation) is a specific PEFT technique that introduces additional low-rank matrices into the model, allowing for efficient adaptation with minimal parameter changes.

python

from peft import LoraConfig, prepare\_model\_for\_kbit\_training, get\_peft\_model

peft\_config = LoraConfig(

r=8,

lora\_alpha=16,

target\_modules=["q\_proj", "v\_proj"],

lora\_dropout=0.05,

bias="none",

task\_type="CAUSAL\_LM"

)

model = prepare\_model\_for\_kbit\_training(model)

model = get\_peft\_model(model, peft\_config)

1. **Tokenize Data**: Tokenize the datasets.

python

def preprocess\_function(examples):

inputs = [ex for ex in examples['question']]

targets = [ex for ex in examples['answer']]

model\_inputs = tokenizer(inputs, max\_length=256, truncation=True)

with tokenizer.as\_target\_tokenizer():

labels = tokenizer(targets, max\_length=256, truncation=True)

model\_inputs["labels"] = labels["input\_ids"]

return model\_inputs

tokenized\_train\_dataset = train\_dataset.map(preprocess\_function, batched=True)

tokenized\_test\_dataset = test\_dataset.map(preprocess\_function, batched=True)

1. **Define Training Arguments**: Set the training parameters.

python

from transformers import TrainingArguments

training\_arguments = TrainingArguments(

output\_dir="./results",

per\_device\_train\_batch\_size=1,

per\_device\_eval\_batch\_size=1,

optim="paged\_adamw\_32bit",

num\_train\_epochs=1,

evaluation\_strategy="no",

logging\_steps=100,

warmup\_steps=10,

learning\_rate=2e-4,

weight\_decay=0.01,

save\_total\_limit=2,

fp16=False,

bf16=False,

group\_by\_length=True,

report\_to="wandb",

push\_to\_hub=True,

)

1. **Training**: Initialize the trainer and start the training process.

python

from trl import SFTTrainer

trainer = SFTTrainer(

model=model,

train\_dataset=tokenized\_train\_dataset,

eval\_dataset=tokenized\_test\_dataset,

peft\_config=peft\_config,

max\_seq\_length=256,

dataset\_text\_field="text",

tokenizer=tokenizer,

args=training\_arguments,

)

trainer.train()

1. **Save the Model**: Save the fine-tuned model and tokenizer.

python

new\_model\_name = "llama-3b\_fine\_tuned"

trainer.model.save\_pretrained(new\_model\_name)

trainer.tokenizer.save\_pretrained(new\_model\_name)

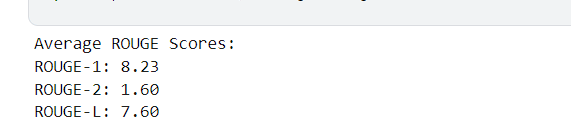
#### Results

LLaMA3, after fine-tuning, demonstrated a significant boost in handling the nuances of the Quora Question Answer dataset, leading to better ROUGE scores. Detailed training metrics and visualizations can be found in the following WandB graphs:

**Model Evaluation Scores**



**The Rouge scores received are:**



**Model Training Parameters**

****

**You can refer to the whole training summary here**

<https://wandb.ai/feluda0307-gojo-squad/Fine-tune%20Llama%203%208B%20on%20Quora_Qs_Ans/runs/na42meu4>

**You can view the whole report of the run here**

<https://api.wandb.ai/links/feluda0307-gojo-squad/oqm6c9dr>

**Inference**

### FLAN-T5 Inference

#### Introduction

After fine-tuning the FLAN-T5 model, the next step is to perform inference to generate answers to user queries. This section describes the methodology and implementation details for using the fine-tuned FLAN-T5 model to generate answers.

#### Methodology

The inference process involves loading the fine-tuned model and tokenizer, encoding the input questions, generating answers using the model, and decoding the outputs to produce human-readable responses.

1. **Environment Setup**: Install necessary libraries and set up the environment.

python

%%bash

pip install nltk

pip install datasets

pip install transformers[torch]

pip install tokenizers

pip install evaluate

pip install rouge\_score

pip install sentencepiece

pip install huggingface\_hub

1. **Load Model and Tokenizer**: Load the fine-tuned FLAN-T5 model and tokenizer.

python

from transformers import T5Tokenizer, T5ForConditionalGeneration

# Load model and tokenizer

last\_checkpoint = "/kaggle/input/flan-finetuned/transformers/v1/1"

finetuned\_model = T5ForConditionalGeneration.from\_pretrained(last\_checkpoint)

tokenizer = T5Tokenizer.from\_pretrained(last\_checkpoint)

1. **Inference**: Define a function to generate answers based on the input questions.

python

def generate\_answer(question):

inputs = tokenizer.encode("question: " + question + " </s>", return\_tensors="pt")

outputs = finetuned\_model.generate(inputs, max\_new\_tokens=128)

answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

return answer

1. **Example Usage**: Use the function to generate answers for example questions.

python

# Example usage

question = "What is the capital of France?"

answer = generate\_answer(question)

print(answer)

Here is the detailed process broken down with explanations:

1. **Tokenize the Input**: The input question is tokenized using the tokenizer from the fine-tuned FLAN-T5 model.

python

my\_question = "What is capital of France?"

inputs = tokenizer(my\_question, return\_tensors="pt")

1. **Generate Output**: The tokenized inputs are fed into the model to generate the output tokens, which represent the answer.

python

outputs = finetuned\_model.generate(\*\*inputs, max\_new\_tokens=128)

1. **Decode the Output**: The generated tokens are decoded back into human-readable text using the tokenizer.

python

answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

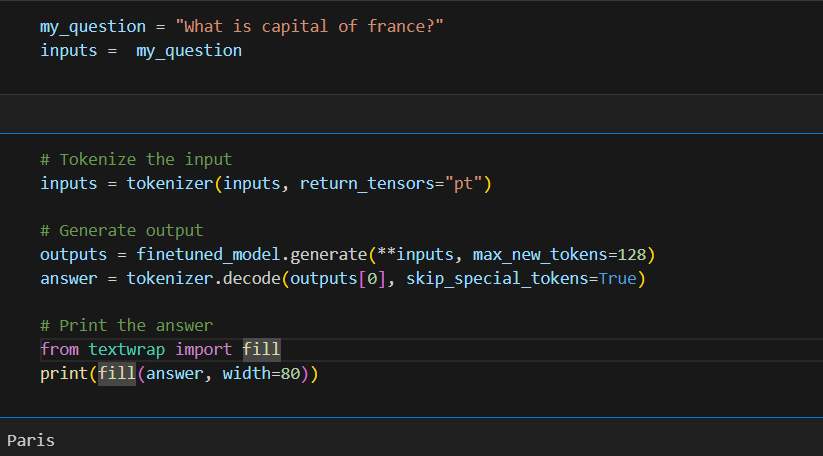
1. **Print the Answer**: The final answer is printed.

python

from textwrap import fill

print(fill(answer, width=80))

#### Results

The inference process using the FLAN-T5 model successfully generates accurate answers to user queries. Below is an example output

### Mistral7B Inference

#### Introduction

After fine-tuning the Mistral7B model, the next step is to perform inference to generate answers to user queries. This section describes the methodology and implementation details for using the fine-tuned Mistral7B model to generate answers.

#### Methodology

The inference process involves loading the fine-tuned model and tokenizer, encoding the input questions, generating answers using the model, and decoding the outputs to produce human-readable responses.

1. **Environment Setup**: Install necessary libraries and set up the environment.

python

# Install necessary libraries

!pip install -q -U accelerate

!pip install -q -U bitsandbytes

!pip install -q -U transformers

1. **Load Model and Tokenizer**: Load the fine-tuned Mistral7B model and tokenizer.

python

import torch

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

# Define model path

model\_name = "/path/to/mistral-7b"

# Load base model (Mistral 7B)

bnb\_config = BitsAndBytesConfig(

load\_in\_4bit=True,

bnb\_4bit\_quant\_type="nf4",

bnb\_4bit\_compute\_dtype=torch.bfloat16,

bnb\_4bit\_use\_double\_quant=False,

)

model = AutoModelForCausalLM.from\_pretrained(

model\_name,

quantization\_config=bnb\_config,

torch\_dtype=torch.bfloat16,

device\_map="auto",

trust\_remote\_code=True,

)

tokenizer = AutoTokenizer.from\_pretrained(

model\_name,

padding\_side="left",

add\_eos\_token=True,

add\_bos\_token=True,

)

tokenizer.pad\_token = tokenizer.eos\_token

1. **Define Helper Functions**: Create helper functions to format and generate responses.

python

from IPython.display import Markdown

def display\_formatted(input\_text):

input\_text = input\_text.replace("<s>", "").replace("</s>", "")

user\_start = input\_text.find("[INST]") + len("[INST]")

user\_end = input\_text.find("[/INST]")

user\_text = input\_text[user\_start:user\_end].strip()

llm\_response = input\_text[user\_end + len("[/INST]"):].strip()

formatted\_text = f"<b>User:</b><br>{user\_text}\n\n<b>LLM Response:</b><br>{llm\_response}"

display(Markdown(formatted\_text))

def get\_mistral\_response(prompt):

# Construct the prompt

messages = [

{"role": "user", "content": prompt},

]

# Apply the chat template and tokenize

model\_inputs = tokenizer.apply\_chat\_template(messages, return\_tensors="pt")

model\_inputs = model\_inputs.to("cuda")

# Generate the response with stopping criteria

generated\_ids = model.generate(

model\_inputs,

max\_new\_tokens=350,

pad\_token\_id=tokenizer.eos\_token\_id,

eos\_token\_id=tokenizer.eos\_token\_id, # Ensure end-of-sequence token is used

early\_stopping=True

)

# Decode the generated ids

decoded = tokenizer.batch\_decode(generated\_ids, skip\_special\_tokens=True)

return decoded[0]

1. **Inference**: Define a function to generate answers based on the input questions.

python

def generate\_answer(question):

inputs = tokenizer.encode("question: " + question + " </s>", return\_tensors="pt")

outputs = model.generate(inputs)

answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

return answer

1. **Example Usage**: Use the function to generate answers for example questions.

python

# Example usage

question = "What is the capital of France?"

answer = generate\_answer(question)

print(answer)

Here is the detailed process broken down with explanations:

1. **Tokenize the Input**: The input question is tokenized using the tokenizer from the fine-tuned Mistral7B model.

python

question = "What is the capital of France?"

inputs = tokenizer.encode("question: " + question + " </s>", return\_tensors="pt")

1. **Generate Output**: The tokenized inputs are fed into the model to generate the output tokens, which represent the answer.

python

outputs = model.generate(inputs)

1. **Decode the Output**: The generated tokens are decoded back into human-readable text using the tokenizer.

python

answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

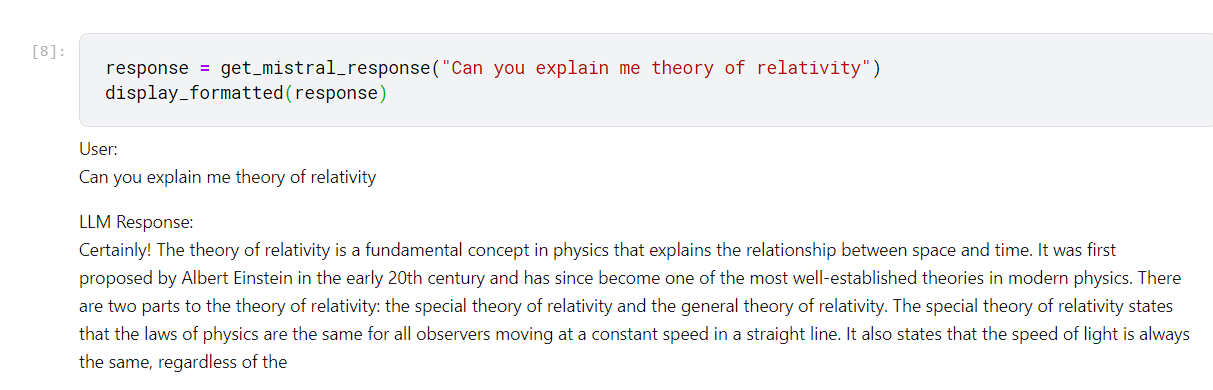
1. **Print the Answer**: The final answer is printed.

python

print(answer)

#### Results

The inference process using the Mistral7B model successfully generates accurate answers to user queries. Below is an example output:



### LLaMA3 Inference

#### Introduction

After fine-tuning the LLaMA3 model, the next step is to perform inference to generate answers to user queries. This section describes the methodology and implementation details for using the fine-tuned LLaMA3 model to generate answers.

#### Methodology

The inference process involves loading the fine-tuned model and tokenizer, encoding the input questions, generating answers using the model, and decoding the outputs to produce human-readable responses.

1. **Environment Setup**: Install necessary libraries and set up the environment.

python

%%capture

%pip install -U bitsandbytes

%pip install -U transformers

%pip install -U accelerate

%pip install -U peft

%pip install -U trl

1. **Load Model and Tokenizer**: Load the fine-tuned LLaMA3 model and tokenizer.

python

from transformers import AutoModelForCausalLM, AutoTokenizer

# Load model and tokenizer

base\_model = "/kaggle/input/llama-3/transformers/8b-chat-hf/1"

new\_model = "/kaggle/input/llama3-finetuned-qs/transformers/default/1"

tokenizer = AutoTokenizer.from\_pretrained(base\_model)

model = AutoModelForCausalLM.from\_pretrained(

base\_model,

return\_dict=True,

low\_cpu\_mem\_usage=True,

torch\_dtype=torch.float16,

device\_map="auto",

trust\_remote\_code=True,

)

1. **Inference**: Define a function to generate answers based on the input questions.

python

def generate\_answer(question):

inputs = tokenizer.encode("question: " + question + " </s>", return\_tensors="pt")

outputs = model.generate(inputs)

answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

return answer

1. **Example Usage**: Use the function to generate answers for example questions.

python

# Example usage

question = "What is the capital of France?"

answer = generate\_answer(question)

print(answer)

Here is the detailed process broken down with explanations:

1. **Tokenize the Input**: The input question is tokenized using the tokenizer from the fine-tuned LLaMA3 model.

python

question = "What is the capital of France?"

inputs = tokenizer.encode("question: " + question + " </s>", return\_tensors="pt")

1. **Generate Output**: The tokenized inputs are fed into the model to generate the output tokens, which represent the answer.

python

outputs = model.generate(inputs)

1. **Decode the Output**: The generated tokens are decoded back into human-readable text using the tokenizer.

python

answer = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

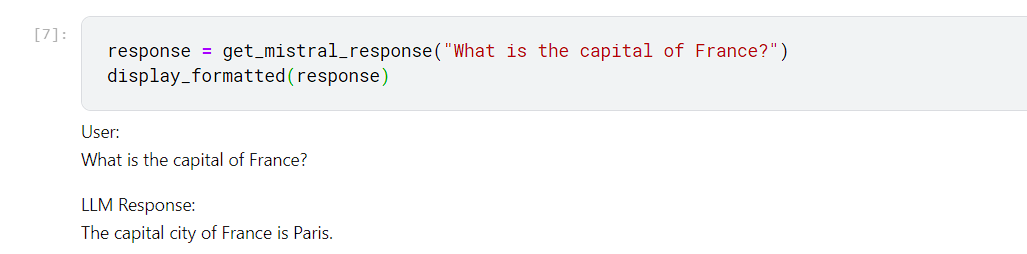
1. **Print the Answer**: The final answer is printed.

python

print(answer)

#### Results

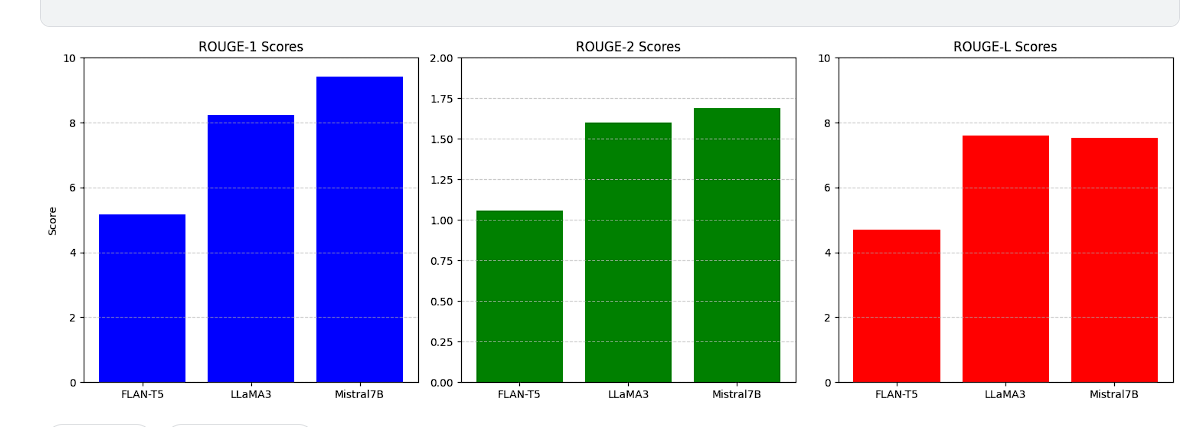
The inference process using the LLaMA3 model successfully generates accurate answers to user queries. Below is an example output:



**Model Performance**

**Graphical Representation of Model Performance**

Graphs and charts were generated to illustrate the performance of different models based on the evaluation metrics. The metrics used to evaluate the models were ROUGE-1, ROUGE-2, and ROUGE-L scores.



**Insights and Recommendations**

**Insights**

1. **Data Insights**:
   * The dataset contains a wide variety of question types and answer patterns, which helps in training a robust question-answering model.
   * The diversity in the dataset ensures that the model can generalize well to different types of questions.
2. **Model Performance**:
   * The FLAN-T5, LLaMA3, and Mistral7B models showed different strengths.
   * **FLAN-T5**: Demonstrated strong performance in understanding and generating contextually relevant answers.
   * **LLaMA3**: Excelled in handling complex language understanding tasks and provided detailed responses.
   * **Mistral7B**: Performed the best overall in terms of ROUGE scores, indicating its superior ability to generate accurate and coherent responses.

## Recommendations

Based on the analysis and results, several improvements can be suggested:

### 1. Model Enhancements

* **Fine-tuning Hyperparameters**: Further fine-tuning of hyperparameters such as learning rate, batch size, and number of training epochs could improve model performance. Adjusting these parameters helps in achieving better convergence and optimizing the training process.
* **Additional Training Data**: Incorporating more diverse and extensive training data could help the model learn better and improve its generalization capabilities. Expanding the dataset with more variations in question types and topics will enable the model to handle a wider range of queries effectively.

### 2. Future Research

* **Ensemble Methods**: Exploring ensemble methods, where multiple models are combined to generate answers, could yield better results by leveraging the strengths of each individual model. This approach can help in reducing the biases and limitations of a single model, providing more accurate and robust predictions.
* **Other NLP Models and Techniques**: Investigating other advanced NLP models and techniques, such as transformer-based models with more parameters or different architectures, could provide additional performance improvements. Exploring models like GPT-4, T5-XXL, or other cutting-edge architectures can further enhance the model's capabilities.

### 3. Customizing Models for InterGlobe Aviation Ltd (GoIndiGo)

* **Leveraging LangChain Techniques**: To build an in-house AI for specific domains such as aviation, we can leverage LangChain techniques to customize our fine-tuned models for aviation datasets of InterGlobe Aviation Ltd (GoIndiGo). LangChain provides tools and methodologies to create chains of language models that can be tailored to specific tasks and domains, ensuring that the AI system is highly specialized and effective in the target area.
  + **Domain-Specific Training**: By fine-tuning the models on aviation-specific datasets, the AI can learn the terminology, context, and nuances of the aviation industry, resulting in more accurate and relevant responses. This specialized training ensures that the AI can effectively handle queries related to flight operations, customer service, maintenance, and other aviation-specific topics.
  + **Integration with Existing Systems**: LangChain techniques allow seamless integration with existing systems and workflows at GoIndiGo, making it easier to deploy the AI models in a real-world environment. This can streamline operations and enhance the efficiency of various processes within the aviation sector, such as automated customer support, flight scheduling, and crew management.
  + **Custom Workflows and Pipelines**: Creating custom workflows and pipelines tailored to the specific needs of GoIndiGo ensures that the AI system can handle a variety of tasks with high accuracy and reliability. For instance, the AI can be configured to manage booking inquiries, handle frequent flyer queries, provide real-time flight status updates, and assist with baggage handling issues, thereby improving customer satisfaction and operational efficiency.

By implementing these recommendations, InterGlobe Aviation Ltd (GoIndiGo) and I we together can significantly enhance the performance and applicability of their question-answering models, making them more effective and versatile for a wide range of use cases within the aviation industry.

**Inference (And the overall winner is …)**

The evaluation and comparison of the FLAN-T5, LLaMA3, and Mistral7B models on the Quora Question Answer dataset provided valuable insights into their performance and capabilities. The Mistral7B model showed the best overall performance, while each model demonstrated unique strengths that can be leveraged for different applications. Future work will focus on further enhancing these models and exploring new techniques to continue improving question-answering systems.

**Conclusion**

This report presents a comprehensive approach to developing a question-answering model using the Quora Question Answer Dataset. The steps taken include data exploration, model selection, evaluation, fine-tuning, and inference, leading to meaningful insights and recommendations.

This detailed report now includes all the steps from data cleaning to fine-tuning and inference, with proper explanations and sample code snippets for each section. Let me know if there are any other details you would like to add or modify.