In [ ]:
```
!unzip -q /content/drive/MyDrive/Project_4_export.zip
```

In [ ]:
```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) h
ttps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.p
kg.dev/colab-wheels/public/simple/)
Collecting transformers
  Downloading transformers-4.29.2-py3-none-any.whl (7.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.1/7.1 MB 33.4 MB/s eta 0:
00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist
-packages (from transformers) (3.12.0)
Collecting huggingface-hub<1.0,>=0.14.1 (from transformers)
  Downloading huggingface_hub-0.14.1-py3-none-any.whl (224 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 16.4 MB/s eta
0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/d
ist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.
10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/d
ist-packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python
3.10/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist
-packages (from transformers) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylin
ux2014_x86_64.whl (7.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 24.8 MB/s eta 0:
00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/di
st-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-p
ackages (from huggingface-hub<1.0,>=0.14.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/l
ib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.14.1->transform
ers) (4.5.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/py
thon3.10/dist-packages (from requests->transformers) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pytho
n3.10/dist-packages (from requests->transformers) (2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/li
b/python3.10/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/
dist-packages (from requests->transformers) (3.4)
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.14.1 tokenizers-0.13.3 transform
ers-4.29.2
```

1. Long-Short Term Memory (LSTM) for Sentiment Analysis (20 points)

Your task is to create an LSTM network to predict sentiment from Yelp reviews (train_yelp_reviews.csv). A review is considered positive if labeled with a `1` and negative if labeled with a `0`. (Hint: Read the provided CSV files with pandas using sep = `\t` and engine= `python`)

You may use any packages or tools as you wish, however, the code that you submit must be written on your own. You are free to experiment with pre-processing, model architectures, training procedures, removing stop-words, or hyper-parameters, as long as your model contains at least one LSTM layer. You may find using a GPU to be beneficial, but we have made sure that good classification can be obtained using an average CPU.

1.1 Data Inspection (5 points)

Read through some of the reviews. Display the most 50 commonly occurring tokens for each class. Are there any patterns in reviews that Naive Bayes or other bag-of-word models may not be able to classify accurately?

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import re
warnings.filterwarnings('ignore')
from numpy.random import seed
seed(10)

import os
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import LabelEncoder
```

In [ ]:

```
train_data = pd.read_csv('/content/datasets/train_yelp_reviews.csv', sep='\
train_data.head(5)
```

Out[4]:

	text	label
0	Great time - family dinner on a Sunday night.	1
1	The classic Maine Lobster Roll was fantastic.	1
2	We won't be going back.	0
3	All I have to say is the food was amazing!!!	1
4	Food was good, service was good, Prices were g...	1

In [ ]:
```
#positive reviews and negative reviews
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
from nltk.probability import FreqDist
import re

nltk.download('punkt')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

punctuations = set(string.punctuation)

positive_reviews = []
negative_reviews = []

## implementation understanding taken from chatgpt
for idx, row in train_data.iterrows():
  tokens = word_tokenize(row['text'])
  tokens = [token.lower() for token in tokens if token.lower() not in stop_


  if row['label'] == 1:
    positive_reviews.extend(tokens)

  if row['label'] == 0:
    negative_reviews.extend(tokens)

## referenced from https://tedboy.github.io/nlps/generated/generated/nltk.F
positive_reviews_dict = FreqDist(positive_reviews)
negative_reviews_dict = FreqDist(negative_reviews)

positive_reviews_dict.most_common(50)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[5]:
```
[('good', 66),
 ('great', 64),
 ('food', 53),
 ('place', 51),
 ('service', 42),
 ('friendly', 22),
 ('delicious', 21),
 ('amazing', 20),
 ('back', 20),
 ('really', 20),
 ('best', 20),
 ('nice', 19),
 ('time', 18),
 ("'s", 18),
 ("n't", 17),
 ('go', 17),
 ('also', 16),
 ('like', 16),
 ('restaurant', 15),
 ('staff', 13),
 ('...', 13),
 ('fantastic', 12),
 ('always', 12),
 ('vegas', 12),
 ('love', 12),
 ('awesome', 12),
 ('pretty', 11),
 ('menu', 11),
 ('first', 10),
 ('fresh', 10),
 ('excellent', 10),
 ('steak', 10),
 ('pizza', 10),
 ('experience', 10),
 ('even', 10),
 ('perfect', 9),
 ('chicken', 9),
 ('atmosphere', 9),
 ('server', 9),
 ('one', 9),
 ('prices', 8),
 ('stars', 8),
 ('everything', 8),
 ('made', 8),
 ('spot', 8),
 ('loved', 8),
 ('definitely', 8),
 ('ever', 8),
 ('well', 8),
 ('come', 8)]
```

In [ ]:
```
negative_reviews_dict.most_common(50)
```

Out[6]:
```
[("n't", 68),
 ('food', 60),
 ('place', 40),
 ('back', 33),
 ('service', 31),
 ('like', 28),
 ('go', 25),
 ('would', 20),
 ('good', 20),
 ('time', 19),
 ("'s", 18),
 ('never', 17),
 ('bad', 16),
 ('minutes', 16),
 ('ever', 16),
 ('one', 15),
 ('...', 14),
 ("'ve", 14),
 ('disappointed', 14),
 ('got', 13),
 ('much', 12),
 ('think', 12),
 ('us', 12),
 ('really', 12),
 ("'m", 12),
 ('came', 12),
 ('wo', 11),
 ('going', 11),
 ('worst', 11),
 ('get', 10),
 ('better', 10),
 ('eat', 10),
 ('probably', 9),
 ('even', 9),
 ('``', 9),
 ("''", 9),
 ('bland', 9),
 ('slow', 9),
 ('wait', 9),
 ('terrible', 9),
 ('waited', 9),
 ('also', 8),
 ('burger', 8),
 ('way', 8),
 ('flavor', 8),
 ('restaurant', 8),
 ('experience', 8),
 ('coming', 8),
 ('ordered', 8),
 ('another', 8)]
```

The patterns in reviews that Naive Bayes or other bag-of-word models can find difficult to categorize correctly are frequently connected to the meaning of words in context. Bag-of-word models do not take into account the order or sequence of words in a sentence and instead treat each word as an independent property.

1.2 Model Training (15 points)

Implement your LSTM model and apply it to the training dataset. In order to obtain full credit, you must describe your text pre-processing procedure, describe your network in terms of the layers used, input/output dimensions at each layer, choice of word embedding dictionary, and training procedure.

In [ ]:

```
from sklearn.model_selection import train_test_split

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data['text'])
sequences = tokenizer.texts_to_sequences(train_data['text'])

# Pad sequences to ensure equal length
max_len_seq = 0
for seq in sequences:
  len_seq = len(seq)
  if max_len_seq < len_seq:
    max_len_seq = len_seq

padded_sequences = pad_sequences(sequences, maxlen=max_len_seq)

# Split the data into training and validation sets
xtrain, xval, ytrain, yval = train_test_split(padded_sequences, train_data[

xtrain.shape
```

Out[7]:

```
(720, 32)
```

In [ ]:
```
##LSTM
model = Sequential()
model.add(tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1
model.add(tf.keras.layers.SpatialDropout1D(0.4))
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_se
model.add(tf.keras.layers.BatchNormalization())
# model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.SpatialDropout1D(0.3))
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(Dense(1,activation='sigmoid'))


reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor
                              patience=2, cooldown=3, mode = 'min')
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2)
loss = tf.keras.losses.BinaryCrossentropy(
    from_logits = True
)

metric = tf.keras.metrics.BinaryAccuracy(
    name='accuracy',
)

model.compile(loss=loss,
              optimizer='adam',
              metrics=[metric])
```

In [ ]:
```
from sklearn.metrics import *

num_epochs = 10
history = model.fit(xtrain, ytrain,
                    epochs=num_epochs, verbose=1,
                    validation_data = (xval, yval),
                    batch_size = 32,
                    callbacks = [reduce_lr, early])

prediction = model.predict(xval)

# Get labels based on probability 1 if p>= 0.5 else 0
pred_labels = []
for i in prediction:
    if i >= 0.5:
        pred_labels.append(1)
    else:
        pred_labels.append(0)
print("Report of prediction on test set : \n ", classification_report(yval,
```

```
Epoch 1/10
23/23 [==============================] - 19s 239ms/step - loss: 0.6956 -
accuracy: 0.5347 - val_loss: 0.6922 - val_accuracy: 0.5056 - lr: 0.0010
Epoch 2/10
23/23 [==============================] - 3s 133ms/step - loss: 0.6820 - a
ccuracy: 0.5542 - val_loss: 0.6925 - val_accuracy: 0.5056 - lr: 0.0010
Epoch 3/10
23/23 [==============================] - 4s 168ms/step - loss: 0.6014 - a
ccuracy: 0.6861 - val_loss: 0.6811 - val_accuracy: 0.5167 - lr: 0.0010
Epoch 4/10
23/23 [==============================] - 3s 122ms/step - loss: 0.3343 - a
ccuracy: 0.8681 - val_loss: 0.6244 - val_accuracy: 0.7722 - lr: 0.0010
Epoch 5/10
23/23 [==============================] - 2s 72ms/step - loss: 0.1543 - ac
curacy: 0.9403 - val_loss: 0.5825 - val_accuracy: 0.7389 - lr: 0.0010
Epoch 6/10
23/23 [==============================] - 2s 78ms/step - loss: 0.1163 - ac
curacy: 0.9611 - val_loss: 0.5451 - val_accuracy: 0.7167 - lr: 0.0010
Epoch 7/10
23/23 [==============================] - 1s 38ms/step - loss: 0.0495 - ac
curacy: 0.9847 - val_loss: 0.5231 - val_accuracy: 0.7611 - lr: 0.0010
Epoch 8/10
23/23 [==============================] - 1s 51ms/step - loss: 0.0309 - ac
curacy: 0.9917 - val_loss: 0.4992 - val_accuracy: 0.7444 - lr: 0.0010
Epoch 9/10
23/23 [==============================] - 2s 72ms/step - loss: 0.0220 - ac
curacy: 0.9931 - val_loss: 0.5694 - val_accuracy: 0.7278 - lr: 0.0010
Epoch 10/10
23/23 [==============================] - 1s 60ms/step - loss: 0.0169 - ac
curacy: 0.9944 - val_loss: 0.6446 - val_accuracy: 0.7333 - lr: 0.0010
6/6 [==============================] - 1s 6ms/step
Report of prediction on test set :
              precision    recall  f1-score   support

           0       0.82      0.60      0.69        89
           1       0.69      0.87      0.77        91

    accuracy                           0.73       180
   macro avg       0.75      0.73      0.73       180
weighted avg       0.75      0.73      0.73       180
```

In [ ]:
```
print("F1 Score : {}".format(f1_score(yval, pred_labels)))
```

```
F1 Score : 0.7669902912621359
```

In [ ]:
```
print("Confusion Matrix : \n {}".format(confusion_matrix(yval, pred_labels)
```

```
Confusion Matrix :
 [[53 36]
 [12 79]]
```

1.3 Model Prediction

Apply your trained model to the test data (test_yelp_reviews.csv). Save your predictions as a new file in single column titled "prediction" and export as a CSV file. Please submit your saved CSV to Canvas with the name (_LSTM_predictions.csv). Your grade on the model will be based on the accuracy of your predictions on the unlabeled data. Below is an example output:

In [ ]:
```
stop_words = set(stopwords.words('english'))
test_data = pd.read_csv('/content/datasets/test_yelp_reviews.csv', sep='\t'

# test_data['text'] = test_data['text'].apply(lambda x: [token.lower() for
# test_data['text'] = test_data['text'].apply(lambda x: ' '.join(x))

# # Convert the text data to sequences or embeddings if required by your mo
test_sequences = tokenizer.texts_to_sequences(test_data['text'])

# # Pad sequences to a fixed length
test_sequences = pad_sequences(test_sequences, maxlen=max_len_seq)

# Make predictions
test_preds = model.predict(test_sequences)
test_preds = test_preds.flatten()

pred_labels = []
for i in test_preds:
    if i >= 0.5:
        pred_labels.append(1)
    else:
        pred_labels.append(0)
# Create a DataFrame with predictions
preds_df = pd.DataFrame({'predictions': pred_labels})

# Save predictions to a CSV file
preds_df.to_csv('/content/drive/MyDrive/f006d3c_LSTM_predictions.csv', inde
```

```
4/4 [==============================] - 0s 8ms/step
```

2. FineTune Bert Model (20 points)

Using the same dataset, fine-tune a pre-trained BERT model for sentiment analysis. You are recommended to use the Hugging Face implementation of BERT model and pre-trained weights.

• Preprocess the dataset into a format that can be used by BERT model.

In [ ]:
```
from transformers import BertTokenizer, TFBertForSequenceClassification, Be
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
from numpy.random import seed
seed(100)

xtrain, xtest = train_test_split(train_data, test_size=0.2, random_state=45

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the text and encode labels
train_encodings = tokenizer(list(xtrain['text']), truncation=True, padding=
val_encodings = tokenizer(list(xtest['text']), truncation=True, padding=Tru

# Create TensorFlow dataset
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    xtrain['label'].values
))

val_dataset = tf.data.Dataset.from_tensor_slices((
    dict(val_encodings),
    xtest['label'].values
))
```

Fine-tune the pre-trained BERT model using the training set and evaluate its performance on the validation dataset.

In [ ]:
```
# Load the pre-trained BERT model
bert_model = TFBertForSequenceClassification.from_pretrained('bert-base-unc

# Define the optimizer, loss function, and metrics optimizer, and loss sugg
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

#asked chatgpt about recommended metrics for BERT, and it suggested this
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
# callbacks = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', pati
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor
                                    patience=2, cooldown=3, mode = 'min')
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

# Compile the model
bert_model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
#bert_model.compile(optimizer=optimizer, loss=loss, metrics='accuracy')

# Train the model
bert_model.fit(
    train_dataset.shuffle(1000).batch(64),
    epochs=15,
    batch_size=32,
    validation_data=val_dataset.shuffle(1000).batch(64),
    callbacks = [reduce_lr, early]
)
```

```
All model checkpoint layers were used when initializing TFBertForSequence
Classification.

Some layers of TFBertForSequenceClassification were not initialized from
the model checkpoint at bert-base-uncased and are newly initialized: ['cl
assifier']
You should probably TRAIN this model on a down-stream task to be able to
use it for predictions and inference.
```

```
Epoch 1/15
12/12 [==============================] - 63s 858ms/step - loss: 0.6575 -
accuracy: 0.6347 - val_loss: 0.5395 - val_accuracy: 0.7611 - lr: 2.0000e-
05
Epoch 2/15
12/12 [==============================] - 6s 534ms/step - loss: 0.3973 - a
ccuracy: 0.9000 - val_loss: 0.3032 - val_accuracy: 0.9111 - lr: 2.0000e-0
5
Epoch 3/15
12/12 [==============================] - 7s 566ms/step - loss: 0.2237 - a
ccuracy: 0.9431 - val_loss: 0.3152 - val_accuracy: 0.8944 - lr: 2.0000e-0
5
Epoch 4/15
12/12 [==============================] - 6s 539ms/step - loss: 0.1476 - a
ccuracy: 0.9653 - val_loss: 0.2274 - val_accuracy: 0.9222 - lr: 2.0000e-0
5
Epoch 5/15
12/12 [==============================] - 7s 569ms/step - loss: 0.0720 - a
ccuracy: 0.9889 - val_loss: 0.2371 - val_accuracy: 0.9222 - lr: 2.0000e-0
5
Epoch 6/15
12/12 [==============================] - 6s 529ms/step - loss: 0.0500 - a
ccuracy: 0.9931 - val_loss: 0.2462 - val_accuracy: 0.9167 - lr: 2.0000e-0
5
Epoch 7/15
12/12 [==============================] - 7s 562ms/step - loss: 0.0279 - a
ccuracy: 0.9972 - val_loss: 0.2476 - val_accuracy: 0.9167 - lr: 4.0000e-0
6
Epoch 8/15
12/12 [==============================] - 6s 528ms/step - loss: 0.0216 - a
ccuracy: 1.0000 - val_loss: 0.2569 - val_accuracy: 0.9222 - lr: 4.0000e-0
6
Epoch 9/15
12/12 [==============================] - 7s 549ms/step - loss: 0.0194 - a
ccuracy: 1.0000 - val_loss: 0.2581 - val_accuracy: 0.9222 - lr: 4.0000e-0
6
```

```
Out[24]: <keras.callbacks.History at 0x7f548944c580>
```

Compare the performance of the BERT model with the LSTM model and comment on your findings.

Based on the initial analysis, I found that BERT model is performing much better than the LSTM model, the F1 Score of LSTM model was 0.785, however for BERT it was 0.92.

In [ ]:
```
val_predictions = bert_model.predict(val_dataset.batch(64))
val_pred_labels = np.argmax(val_predictions.logits, axis=1)

# Calculate classification report
print(classification_report(xtest['label'].values, val_pred_labels))
```

```
3/3 [==============================] - 4s 146ms/step
              precision    recall  f1-score   support

           0       0.93      0.91      0.92        90
           1       0.91      0.93      0.92        90

    accuracy                           0.92       180
   macro avg       0.92      0.92      0.92       180
weighted avg       0.92      0.92      0.92       180
```

In [ ]:
```
print("F1 Score : {}".format(f1_score(xtest['label'].values, val_pred_label
```

```
F1 Score : 0.923076923076923l
```

Apply your trained model to the test data (test_yelp_reviews.csv). Save your predictions as a new file in single column titled "classification" and export as a CSV file. Please submit a copy of your saved CSV as a separate file on Canvas with the format (_BERT_predictions.csv). Your grade on the model will be based on the F1-score of your predictions on the unlabeled data.

In [ ]:
```
test_data = pd.read_csv('/content/datasets/test_yelp_reviews.csv', sep='\t'

# Convert the text data to sequences or embeddings if required by your mode
test_sequences = tokenizer(list(test_data['text']), truncation=True, paddin

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_sequences)
))

# Make predictions
test_preds = bert_model.predict(test_dataset.batch(64))
# Create a DataFrame with predictions
test_preds = np.argmax(test_preds.logits, axis=1)

preds_df = pd.DataFrame({'predictions': test_preds})

# Save predictions to a CSV file
preds_df.to_csv('/content/drive/MyDrive/f006d3c_BERT_predictions.csv', inde
```

```
2/2 [==============================] - 4s 135ms/step
```

4. Large Language Models (30 points)

In this part, you will be asked several questions about large language models. No coding is needed. 6 points for each question.

• Why is it challenging to train language models using reinforcement learning with human feedback (RLHF)? Give three examples to demonstrate.

It is challenging to train language models using reinforcement learning with human feedback, as there are predominantly 3 issues that would occur if we were to follow these steps.

1. Lack of Scalability as the RLHF-based modeling approach would require a large dataset consisting of human-generated training datasets, this can be time intensive and computationally expensive process.
2. Delayed sparse rewards that are commonly received by the language models as feedback signals would make it difficult for the model to learn from its processes to receive rewards making it a slower and unstable process.
3. Trade-off between exploration and exploitation: In RLHF, it's critical to strike a balance between the two. Language models must investigate various actions to find superior tactics while making use of previously acquired knowledge. It can be difficult to strike the ideal balance between exploration and exploitation.

• What are pros and cons of instruction finetuning and RLHF?

Instruction Finetuning is very much interpretable as it makes decision based on the inputs provided by us providing us with precise control on how we would want to see the output of the model.

One of the drawbacks for this model is that it is a time consuming and laborious process to provide the model with instructions. It also lacks generalizability and that it would be required to be specifically trained for those scenarios.

RLHF on the other hand can be generalizable and would be able to adapt to different scenarios as it learns from feedbacks provided by us. The major drawbacks for RLHF would be that it is very expensive to train the model due to large number of parameters required to train the model.

• What are some of the challenges of evaluating the current generation of language models?

1. One of the main concerns and issues that plague the current generation of language models is the data quality and diversity of the dataset the model is being trained on. How well the data has been collected affects the model performance, poor quality will often lead the model to overfitting, bias and have generalizibility issues.
2. Ethical and social implications is another area of concern that can potentially validate the current generation of language models. These models can impact society, as it can generate misleading or unethical contents. It can also empower people by improving how people communicate with each others, and gain information.
3. Selecting adequate evaluation metrics and criteria is the third difficulty that has an impact on language models. The effectiveness and quality of the language models can be measured through evaluation measures. Conventional evaluation metrics, however, might not be adequate or appropriate for all models. For instance, certain metrics may ignore deeper-level factors like coherence, relevance, logic, and creativity in favor of focusing exclusively on the surface-level accuracy or fluency of the created content. Additionally, some criteria could be ambiguous.

• Compare three popular language models (you can choose ChatGPT and any other two, e.g., [https://www.perplexity.ai/ (https://www.perplexity.ai/)](https://www.perplexity.ai/), [https://you.com/ (https://you.com/)](https://you.com/), [https://nat.dev/ (https://nat.dev/)](https://nat.dev/), [https://bard.google.com/ (https://bard.google.com/)](https://bard.google.com/)). Come up with 5 challenging questions, and compare their output. Comment on the quality of the output and explain what is the possibly the reason.

I asked them about various challenging questions focusing on the realm of quantum computing, to applications of Natural language processing in resolving bias and eradicating iliteracy to whether god exists or not. What I found out that the response was similar for the Chatgpt, Perplexity, and you with changing in the way they phrased the sentences. The quality of output was good and it seems that though it seems complicated they were fairly able to avoid any bias or any kind of ethical issues with their responses.