1. Load Dataset:

```
!nvidia-smi
```

```
Tue May 13 06:00:11 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Ver
|-----------------------------------------+------------------------+-------
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volati
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Ut
|                                         |                        |
|=========================================+========================+=======
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
| N/A   70C    P0             30W /   70W |    2466MiB /  15360MiB |      0
|                                         |                        |
+-----------------------------------------+------------------------+-------

+-----------------------------------------------------------------------------------------+
| Processes:
|  GPU   GI   CI        PID   Type   Process name
|        ID   ID
|=========================================================================================
+-----------------------------------------------------------------------------------------+
```

```
!pip install transformers torch scikit-learn pandas
```

```
import re
import os
import json
import pandas as pd
import matplotlib.pyplot as plt
import torch
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
#Pre-train BERT:
from transformers import BertTokenizer, BertForSequenceClassification, Trai
from sklearn.preprocessing import LabelEncoder
from peft import get_peft_model, LoraConfig
#Confidence Score System:
import requests
from Bio import Entrez
from langchain import LLMChain, PromptTemplate
from langchain.chains import RetrievalQA
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
#Gemini Model:
from google import genai
from google.genai import types
import base64
import google.generativeai as genai
```

(1) Covid Fake News Dataset:

```
# List of JSON files to process
json_files = [
    'Cleaned_Covid19_Train.json',
    'Cleaned_Covid19_Dev.json',
]
data_dict = {}
# Process each JSON file
for json_file in json_files:
    # Load the dataset
    with open(json_file, 'r') as file:
        data = json.load(file)

    # Prepare a list to hold the processed data
    jsonl_data = []

    # Extract and process each entry
    for entry in data:
        # Extract the id, tweet, and label
        tweet = entry['tweet']
        label = entry['label']

        # Tokenize the tweet
```

```
        tokens = re.findall(r'\b\w+\b', tweet)  # Keep only words and numbe
        reconstructed_tweet = ' '.join(tokens)

        # Prepare the JSONL entry with the required structure
        jsonl_entry = {
            "systemInstruction": {
                "role": "assistant",  # Example role, adjust as needed
                "parts": [
                    {
                        "text": "Classification the content is Fake, Real,
                    }
                ]
            },
            "contents": [
                {
                    "role": "user",
                    "parts": [
                        {
                            "text": f"TRANSCRIPT: \n{reconstructed_tweet}\r
                        }
                    ]
                },
                {
                    "role": "model",
                    "parts": [
                        {
                            "text": label  # The label indicating the model
                        }
                    ]
                }
            ]
        }
        jsonl_data.append(jsonl_entry)

   # Write the processed data to a JSONL file
    output_file = json_file.replace('.json', '.jsonl')  # Change the extens
    with open(output_file, 'w') as outfile:
        for entry in jsonl_data:
            json.dump(entry, outfile)
            outfile.write('\n')  # Write each entry on a new line
    print(f"Processed {json_file} and saved to {output_file}.")
    data_dict[json_file] = jsonl_data
# Access the data using the correct keys - the original filenames
covid_train_data = data_dict['Cleaned_Covid19_Train.json']  # Corrected key
covid_dev_data = data_dict['Cleaned_Covid19_Dev.json']  # Corrected key
# Print the first few entries for verification
print(f"First few entries from claims_test_data:\n{covid_train_data[:5]}")

```

```
Processed Cleaned_Covid19_Train.json and saved to Cleaned_Covid19_Train.jso
Processed Cleaned_Covid19_Dev.json and saved to Cleaned_Covid19_Dev.jsonl.
First few entries from claims_test_data:
[{'systemInstruction': {'role': 'assistant', 'parts': [{'text': 'Classifica
```

(2) Health Fact Dataset:

```
import json
import re
import os

# List of JSON files to process
json_files = [
    'healthfact_traindata.json',
    'cleaned_healthfact_test.json',
    'cleaned_healthfact_dev.json'
]
data_dict = {}
# Process each JSON file
for json_file in json_files:
    # Prepare a list to hold the processed data
    jsonl_data = []
    # Load the dataset
    with open(json_file, 'r') as file:
        # Read each line as a separate JSON object
        for line in file:
            try:
                entry = json.loads(line)
                # Extract the claim, explanation, and label
                claim = entry['claim']
                explanation = entry['explanation']
                label = entry['label']

                # Tokenize the claim
                tokens = re.findall(r'\b\w+\b', claim)  # Keep only words a
                reconstructed_claim = ' '.join(tokens)

                # Prepare the JSONL entry in the required format
                jsonl_entry = {
                    "systemInstruction": {
                        "role": "assistant",  # Example role, adjust as nee
                        "parts": [
                            {
                                "text": "You are a helpful assistant."  # E
                            }
                        ]
                    },
                    "contents": [
                        {
```

```
                            "role": "user",
                            "parts": [
                                {
                                    "text": f"CLAIM: {reconstructed_claim}\
                                }
                            ]
                        },
                        {
                            "role": "model",
                            "parts": [
                                {
                                    "text": label  # The label indicating t
                                }
                            ]
                        }
                    ]
                }
                jsonl_data.append(jsonl_entry)
            except json.JSONDecodeError as e:
                print(f"Error decoding JSON: {e}")
    # Use the correct key to store the data in the dictionary - keep the or
    data_dict[json_file] = jsonl_data
# Access the data using the correct keys - the original filenames
healthfact_train_data = data_dict['healthfact_traindata.json']  # Correctec
healthfact_test_data = data_dict['cleaned_healthfact_test.json']  # Correct
healthfact_dev_data = data_dict['cleaned_healthfact_dev.json']  # Correctec
# Print the first few entries for verification
print(f"First few entries from healthfact_train_data:\n{healthfact_train_da
# Optionally, write the processed data to JSONL files
for json_file, jsonl_data in data_dict.items():
    output_file = json_file.replace('.json', '.jsonl')  # Change the extens
    with open(output_file, 'w') as outfile:
        for entry in jsonl_data:
            json.dump(entry, outfile)
            outfile.write('\n')  # Write each entry on a new line
    print(f"Processed {json_file} and saved to {output_file}.")
```

```
First few entries from healthfact_train_data:
[{'systemInstruction': {'role': 'assistant', 'parts': [{'text': 'You are a
Processed healthfact_traindata.json and saved to healthfact_traindata.jsonl
Processed cleaned_healthfact_test.json and saved to cleaned_healthfact_test
Processed cleaned_healthfact_dev.json and saved to cleaned_healthfact_dev.j
```

(3) Scifact Dataset:

```
import json
import re

# List of JSONL files to process
```

```
jsonl_files = [
    'dev_3class.jsonl',
    'train_3class.jsonl'
]
data_dict = {}
# Process each JSONL file
for jsonl_file in jsonl_files:
    # Prepare a list to hold the processed data
    processed_data = []

    # Load the dataset
    with open(jsonl_file, 'r') as file:
        for line in file:
            try:
                entry = json.loads(line)

                # Extract the claim, explanation, and label
                claim = entry['claim']
                explanation = entry['evidence_text']
                label = entry['label']

                # Tokenize the claim
                tokens = re.findall(r'\b\w+\b', claim)  # Keep only words a
                reconstructed_claim = ' '.join(tokens)

                # Prepare the JSONL entry in the required format
                jsonl_entry = {
                    "systemInstruction": {
                        "role": "assistant",  # Example role, adjust as nee
                        "parts": [
                            {
                                "text": "You are a helpful assistant."  # E
                            }
                        ]
                    },
                    "contents": [
                        {
                            "role": "user",
                            "parts": [
                                {
                                    "text": f"CLAIM: {reconstructed_claim}\
                                }
                            ]
                        },
                        {
                            "role": "model",
                            "parts": [
                                {
                                    "text": label  # The label indicating t
                                }
                            ]
```

```
                        }
                    ]
                }
                # Append the modified entry to the processed data list
                processed_data.append(jsonl_entry)  # Append the processed
            except json.JSONDecodeError as e:
                print(f"Error decoding JSON: {e}")
    # Store the processed data in the dictionary
    data_dict[jsonl_file] = processed_data
# Access the data using the correct keys - the original filenames
scifact_train_data = data_dict['train_3class.jsonl']  # Corrected key
scifact_test_data = data_dict['dev_3class.jsonl']  # Corrected key
# Print the first few entries for verification
print(f"First few entries from scifact_train_data:\n{scifact_train_data[:5]
# Optionally, write the processed data to new JSONL files
for jsonl_file, processed_data in data_dict.items():
    output_file = jsonl_file.replace('.jsonl', '_processed.jsonl')  # Chang
    with open(output_file, 'w') as outfile:
        for entry in processed_data:
            json.dump(entry, outfile)
            outfile.write('\n')  # Write each entry on a new line
    print(f"Processed {jsonl_file} and saved to {output_file}.")
```

```
First few entries from scifact_train_data:
[{'systemInstruction': {'role': 'assistant', 'parts': [{'text': 'You are a
Processed dev_3class.jsonl and saved to dev_3class_processed.jsonl.
Processed train_3class.jsonl and saved to train_3class_processed.jsonl.
```

2. Data Exploration

```
# Function to explore a dataset
def explore_dataset(data, dataset_name):
    print(f"Exploring dataset: {dataset_name}")
    print(f"Number of entries: {len(data)}")

    # Convert to DataFrame for easier analysis
    df = pd.DataFrame(data)

    # Display the first few entries
    print("First few entries:")
    print(df.head())

    # Display basic statistics
    print("\nBasic statistics:")
    print(df.describe(include='all'))

    # Check the distribution of labels (if applicable)
    if 'label' in df.columns:
```

```
        label_counts = df['label'].value_counts()
        print("\nLabel distribution:")
        print(label_counts)

        # Plot the label distribution
        label_counts.plot(kind='bar', title='Label Distribution')
        plt.xlabel('Labels')
        plt.ylabel('Counts')
        plt.show()

    print("\n" + "-" * 40 + "\n")

# Explore each dataset
explore_dataset(covid_train_data, "Cleaned Covid19 Train Data")
explore_dataset(healthfact_train_data, "Healthfact Train Data")
explore_dataset(scifact_train_data, "SciFact Train Data")
```

```
0  {'role': 'assistant', 'parts': [{'text': 'You ...
1  {'role': 'assistant', 'parts': [{'text': 'You ...
2  {'role': 'assistant', 'parts': [{'text': 'You ...
3  {'role': 'assistant', 'parts': [{'text': 'You ...
4  {'role': 'assistant', 'parts': [{'text': 'You ...

                                            contents
0  [{'role': 'user', 'parts': [{'text': 'CLAIM: T...
1  [{'role': 'user', 'parts': [{'text': 'CLAIM: A...
2  [{'role': 'user', 'parts': [{'text': 'CLAIM: S...
3  [{'role': 'user', 'parts': [{'text': 'CLAIM: S...
4  [{'role': 'user', 'parts': [{'text': 'CLAIM: S...

Basic statistics:
                                       systemInstruction  \
count                                               9804
unique                                                 1
top     {'role': 'assistant', 'parts': [{'text': 'You ...
freq                                                9804

                                                  contents
count                                                9804
unique                                               9803
top     [{'role': 'user', 'parts': [{'text': 'CLAIM: P...
freq                                                    2

----------------------------------------

Exploring dataset: SciFact Train Data
Number of entries: 1261
First few entries:
                                    systemInstruction  \
0  {'role': 'assistant', 'parts': [{'text': 'You ...
1  {'role': 'assistant', 'parts': [{'text': 'You ...
2  {'role': 'assistant', 'parts': [{'text': 'You ...
3  {'role': 'assistant', 'parts': [{'text': 'You ...
4  {'role': 'assistant', 'parts': [{'text': 'You ...

                                            contents
```

```
0  [{'role': 'user', 'parts': [{'text': 'CLAIM: 0...
1  [{'role': 'user', 'parts': [{'text': 'CLAIM: 1...
2  [{'role': 'user', 'parts': [{'text': 'CLAIM: 1...
3  [{'role': 'user', 'parts': [{'text': 'CLAIM: 1...
4  [{'role': 'user', 'parts': [{'text': 'CLAIM: 3...

Basic statistics:
                                          systemInstruction  \
count                                                  1261
unique                                                    1
top     {'role': 'assistant', 'parts': [{'text': 'You ...
freq                                                   1261

                                                    contents
count                                                   1261
unique                                                  1258
top     [{'role': 'user', 'parts': [{'text': 'CLAIM: A...
freq                                                       2

----------------------------------------
```

3. Training Strategy

```python
# Convert datasets to DataFrames for easier manipulation
healthfact_df = pd.DataFrame(healthfact_train_data)
scifact_df = pd.DataFrame(scifact_train_data)

# Combine HealthFact and SciFact datasets for pre-training
combined_pretrain_df = pd.concat([healthfact_df, scifact_df], ignore_index=

# Save the combined dataset for pre-training
combined_pretrain_df.to_json('combined_pretrain_data.jsonl', orient='recorc

# Convert COVID-19 dataset to DataFrame
covid_df = pd.DataFrame(covid_train_data)

# Save the COVID-19 dataset for fine-tuning
covid_df.to_json('covid_finetune_data.jsonl', orient='records', lines=True)

print("Datasets combined and saved for train dataset:")
print("1. Combined Pre-train Data: combined_pretrain_data.jsonl")
print("2. COVID-19 Fine-tune Data: covid_finetune_data.jsonl")
```

```
Datasets combined and saved for train dataset:
1. Combined Pre-train Data: combined_pretrain_data.jsonl
2. COVID-19 Fine-tune Data: covid_finetune_data.jsonl
```

```
# Convert datasets to DataFrames for easier manipulation
healthfact_df_test = pd.DataFrame(healthfact_test_data)
scifact_df_test = pd.DataFrame(scifact_test_data)

# Combine HealthFact and SciFact datasets for pre-training
combined_pretrain_df_test = pd.concat([healthfact_df_test, scifact_df_test]

# Save the combined dataset for pre-training
combined_pretrain_df_test.to_json('combined_pretrain_test_data.jsonl', orie

# Convert COVID-19 dataset to DataFrame
covid_df_test = pd.DataFrame(covid_dev_data)

# Save the COVID-19 dataset for fine-tuning
covid_df_test.to_json('covid_finetune_test_data.jsonl', orient='records', l

print("Datasets combined and saved for Test dataset:")
print("1. Combined Pre-train Data: combined_pretrain_test_data.jsonl")
print("2. COVID-19 Fine-tune Data: covid_finetune_test_data.jsonl")
```

```
Datasets combined and saved for Test dataset:
1. Combined Pre-train Data: combined_pretrain_test_data.jsonl
2. COVID-19 Fine-tune Data: covid_finetune_test_data.jsonl
```

4. Before Fine Tuning Gemini 2.0 Flash Model Prompt + Label

```
# Define a fine-tuning function using Gemini API
def generate_response(prompt):
    response = model.generate_content(prompt)
    return response.text

# Example few-shot training prompt
prompt = """
Claim: "6 10 Sky s EdConwaySky explains the latest COVID19 data and governm
"""

response = generate_response(prompt)
print(response)
```

This claim appears to be a tweet or social media post promoting a segment on Sky News with Ed Conway explaining the latest COVID-19 data and government announcement. It also provides links to further information.

Here's a breakdown of the elements:

- "**6 10**": This likely refers to the time the tweet was posted, possibly 6:10 AM or PM.
- "**Sky s**": This is likely a shortened form of "Sky News's".
- "**EdConwaySky**": This is probably the Twitter handle for Ed Conway, who is likely a Sky News correspondent.
- "**explains the latest COVID19 data and government announcement**": This describes the content of the segment being promoted.
- "**Get more on the coronavirus data here**": This is a call to action, encouraging viewers to click on the provided links.
- "**https t co jvGZlSbFjH https t co PygSKXesBg**": These are shortened URLs likely leading to Sky News's website or relevant articles. It's important to note that link shorteners like "t.co" can hide the true destination of the link.

Potential Issues and Things to Consider:

- **Data Accuracy:** While the claim itself isn't making a specific factual statement, the accuracy of the COVID-19 data presented in the Sky News segment would be dependent on the sources used by Ed Conway.
- **Bias:** It's important to be aware of potential biases. Sky News, like any news organization, has a perspective. Viewers should critically evaluate the information presented.
- **Outdated Information:** COVID-19 data and government announcements change rapidly. The information presented in the segment might be outdated by the time you see the tweet.
- **Link Safety:** Always be cautious when clicking on shortened links, especially from unfamiliar sources. While Sky News is a reputable organization, it's good practice to be vigilant. You can use a link expander to see the actual URL before clicking.

In Conclusion:

The claim itself is a straightforward promotion of a news segment. However, critical evaluation of the information presented in the segment is still necessary. Consider the source, potential biases, the date of the information, and always be cautious when clicking on links.

---

5. Model Initialization

```
!pip install --upgrade google-genai
!gcloud auth application-default login
!pip install --upgrade google-cloud-aiplatform
```

```

from google.colab import auth as google_auth
google_auth.authenticate_user()

import vertexai
from vertexai.generative_models import GenerativeModel
from vertexai.preview.tuning import sft

vertexai.init(project="sit319-25t1-nguyen-ae806d0", location="us-central1")#

gemini_pro = GenerativeModel("gemini-2.0-flash-lite-001")

sft_tuning_job = sft.train(
    source_model=gemini_pro,
    train_dataset="gs://daftt/Cleaned_Covid19_Train-7.jsonl",
    tuned_model_display_name="covid_tuning",
    epochs=100,
    learning_rate_multiplier=1,
)

```

/usr/local/lib/python3.11/dist-packages/google/auth/_default.py:76: UserWarning: Your application has authenticated using end user credentials from Google Cloud SDK without a quota project. You might receive a "quota exceeded" or "API not enabled" error. See the following page for troubleshooting: https://cloud.google.com/docs/authentication/adc-troubleshooting/user-creds. warnings.warn(_CLOUD_SDK_CREDENTIALS_WARNING)

INFO:vertexai.tuning._tuning:Creating SupervisedTuningJob /usr/local/lib/python3.11/dist-packages/google/auth/_default.py:76: UserWarning: Your application has authenticated using end user credentials from Google Cloud SDK without a quota project. You might receive a "quota exceeded" or "API not enabled" error. See the following page for troubleshooting: https://cloud.google.com/docs/authentication/adc-troubleshooting/user-creds. warnings.warn(_CLOUD_SDK_CREDENTIALS_WARNING)

INFO:vertexai.tuning._tuning:SupervisedTuningJob created. Resource name: projects/181085238689/locations/us-central1/tuningJobs/1012173862948831232

INFO:vertexai.tuning._tuning:To use this SupervisedTuningJob in another session:
INFO:vertexai.tuning._tuning:tuning_job = sft.SupervisedTuningJob('projects/181085238689/locations/us-central1/tuningJobs/1012173862948831232')

INFO:vertexai.tuning._tuning:View Tuning Job: https://console.cloud.google.com/vertex-ai/generative/language/locations/us-central1/tuning/tuningJob/1012173862948831232?project=181085238689

6. Training and Evaluation fine tuning: Gemini 2.0 Flash + BERT

A. Pre-train on HealthFact + SciFact for General fact-checking ability.

```
import pandas as pd
import torch
from sklearn.metrics import accuracy_score, precision_score, recall_score,
from transformers import BertTokenizer, BertForSequenceClassification, Trai
from sklearn.preprocessing import LabelEncoder
# Load the combined pre-training dataset (HealthFact + SciFact)
train_combined_data = pd.read_json('combined_pretrain_data.jsonl', lines=Tr
val_combined_data = pd.read_json('combined_pretrain_test_data.jsonl', lines

# Load the COVID-19 fine-tuning dataset
train_covid_data = pd.read_json('covid_finetune_data.jsonl', lines=True)
```

```
val_covid_data = pd.read_json('covid_finetune_test_data.jsonl', lines=True)

# Assuming the datasets have 'claim' and 'label' columns
# Extract claims and labels from nested structure for pre-training
train_claims = train_combined_data['contents'].apply(lambda x: x[0]['parts'
train_labels = train_combined_data['contents'].apply(lambda x: x[1]['parts'
val_claims = val_combined_data['contents'].apply(lambda x: x[0]['parts'][0]
val_labels = val_combined_data['contents'].apply(lambda x: x[1]['parts'][0]

# Convert string labels to integers
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_labels)
val_labels = label_encoder.transform(val_labels)

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the input data for pre-training
train_encodings = tokenizer(train_claims, truncation=True, padding=True, ma
val_encodings = tokenizer(val_claims, truncation=True, padding=True, max_le

# Create a dataset class
class ClaimsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

# Create datasets for pre-training
train_dataset = ClaimsDataset(train_encodings, train_labels)
val_dataset = ClaimsDataset(val_encodings, val_labels)

# Load the BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',

# Define training arguments for pre-training with validation loss logging
training_args = TrainingArguments(
    output_dir='./results/pretrain',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs/pretrain',
```

```
    logging_steps=10,
    eval_strategy="epoch",  # Updated to eval_strategy
)

# Create a Trainer instance for pre-training
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=lambda p: {
        'accuracy': accuracy_score(p.label_ids, p.predictions.argmax(-1)),
        'precision': precision_score(p.label_ids, p.predictions.argmax(-1),
        'recall': recall_score(p.label_ids, p.predictions.argmax(-1), avera
        'f1': f1_score(p.label_ids, p.predictions.argmax(-1), average='weig
        'roc_auc': roc_auc_score(p.label_ids, torch.softmax(torch.tensor(p.
    },
)

# Pre-train the model
trainer.train()
# Save the model and tokenizer
model_save_path = "./my_trained_model"  # Choose your desired save path
model.save_pretrained(model_save_path)
tokenizer.save_pretrained(model_save_path)

print(f"Model and tokenizer saved to: {model_save_path}")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
  warnings.warn(
```


```
Some weights of BertForSequenceClassification were not initialized from the
You should probably TRAIN this model on a down-stream task to be able to us
wandb: WARNING The `run_name` is currently set to the same value as `Traini
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: ht
wandb: You can find your API key in your browser here: https://wandb.ai/aut
wandb: Paste an API key from your profile and hit enter: ·····
wandb: WARNING If you're specifying your api key in code, ensure this code
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: hieunguyen23032001 (hieunguyen23032001-deaki
Tracking run with wandb version 0.19.11
Run data is saved locally in /content/wandb/run-20250513_052637-k7ad0pp8
Syncing run ./results/pretrain to Weights & Biases (docs)
View project at https://wandb.ai/hieunguyen23032001-deakin-university/huggingface
View run at https://wandb.ai/hieunguyen23032001-deakin-university/huggingface/runs/k7ad0pp8
```

[4152/4152 15:29, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1	Roc Auc
1	0.297300	0.288650	0.875817	0.885717	0.875817	0.877511	0.989003
2	0.357900	0.281554	0.890077	0.892447	0.890077	0.889494	0.991119
3	0.208000	0.349890	0.898990	0.902847	0.898990	0.900310	0.991414

```
Model and tokenizer saved to: ./my_trained_model
```

B. Training and Evaluation Covid Fake News Dataset

```
# Prepare the training and validation data
# Prepare the training and validation data
train_covid_claims = train_covid_data['contents'].apply(lambda x: x[0]['par
train_covid_labels = train_covid_data['contents'].apply(lambda x: x[1]['par
val_covid_claims = val_covid_data['contents'].apply(lambda x: x[0]['parts']
val_covid_labels = val_covid_data['contents'].apply(lambda x: x[1]['parts']

# Convert string labels to integers
label_encoder = LabelEncoder()
train_covid_labels = label_encoder.fit_transform(train_covid_labels)
val_covid_labels = label_encoder.transform(val_covid_labels)

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the input data for pre-training
train_encodings = tokenizer(train_covid_claims, truncation=True, padding=Tr
val_encodings = tokenizer(val_covid_claims, truncation=True, padding=True,

# Create datasets for pre-training
train_dataset = ClaimsDataset(train_encodings, train_labels)
val_dataset = ClaimsDataset(val_encodings, val_labels)

# Load the BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',

# Pre-train the model
trainer.train()
```

```
Some weights of BertForSequenceClassification were not initialized from the
You should probably TRAIN this model on a down-stream task to be able to us
```

[4152/4152 15:42, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1	Roc Auc
1	0.401200	0.396093	0.887701	0.890988	0.887701	0.888816	0.990407
2	0.177600	0.489731	0.888295	0.889070	0.888295	0.887639	0.990609
3	0.119300	0.606078	0.888889	0.892492	0.888889	0.890275	0.990721

```
TrainOutput(global_step=4152, training_loss=0.15090772818669906, metrics=
{'train runtime': 942.5088, 'train samples per second': 35.22,
```

7. Test Prompt and Label

```
def generate():
  client = genai.Client(
      vertexai=True,
      project="181085238689"
```

```
    project="181085238689",
    location="us-central1",
)

msg3_text1 = types.Part.from_text(text="""Clearly the Obama administration

model = "projects/181085238689/locations/us-central1/endpoints/5419770989
contents = [
  types.Content(
    role="user",
    parts=[
      types.Part.from_text(text="""Multiple Facebook posts claim that Auss
    ]
  ),
  types.Content(
    role="model",
    parts=[
      types.Part.from_text(text=label)
    ]
  ),
  types.Content(
    role="user",
    parts=[
      msg3_text1
    ]
  ),
]
generate_content_config = types.GenerateContentConfig(
  temperature = 0.2,
  top_p = 0.8,
  max_output_tokens = 1024,
  response_modalities = ["TEXT"],
  safety_settings = [types.SafetySetting(
    category="HARM_CATEGORY_HATE_SPEECH",
    threshold="OFF"
  ),types.SafetySetting(
    category="HARM_CATEGORY_DANGEROUS_CONTENT",
    threshold="OFF"
  ),types.SafetySetting(
    category="HARM_CATEGORY_SEXUALLY_EXPLICIT",
    threshold="OFF"
  ),types.SafetySetting(
    category="HARM_CATEGORY_HARASSMENT",
    threshold="OFF"
  )],
)

for chunk in client.models.generate_content_stream(
  model = model,
  contents = contents,
  config = generate_content_config,
  ):
```

```
    print(chunk.text, end="")

generate()
```

/usr/local/lib/python3.11/dist-packages/google/auth/_default.py:76: UserWarning: Your application has authenticated using end user credentials from Google Cloud SDK without a quota project. You might receive a "quota exceeded" or "API not enabled" error. See the following page for troubleshooting: https://cloud.google.com/docs/authentication/adc-troubleshooting/user-creds. warnings.warn(_CLOUD_SDK_CREDENTIALS_WARNING)

fake

---

8. Implement the Confidence Scoring System

```
!pip install transformers torch requests beautifulsoup4
```

Show hidden output

```
!pip install biopython
```

Show hidden output

```
!pip install langchain-community
```

Show hidden output

Retrieval article

```python
import requests
from bs4 import BeautifulSoup

def retrieve_articles(query):
    """
    Retrieve articles from PubMed based on a query.

    This function uses the PubMed API to search for relevant articles
    based on the provided query and parses the HTML response using Beautifu
    """
    base_url = "https://pubmed.ncbi.nlm.nih.gov/"
    search_url = f"{base_url}?term={query}"

    response = requests.get(search_url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        # Extract article titles and summaries (example, you may need to ad
        articles = []
        for article_tag in soup.find_all('div', class_='docsum'):  # Exampl
            title = article_tag.find('a', class_='docsum-title').text.strip
            summary = article_tag.find('div', class_='abstract').text.strip
            articles.append({'title': title, 'summary': summary})

        return articles
    else:
        return None
```

Implement "Trust Score" calculation

```python
def calculate_trust_score(prediction, retrieved_articles):
    """
    Calculate a trust score based on the LLM's prediction and the retrievec
    The trust score is determined by the number of articles that support or
    """
    support_count = 0
    contradict_count = 0

    for article in retrieved_articles:
        if prediction.lower() in article['title'].lower() or prediction.low
            support_count += 1
        else:
            contradict_count += 1

    total_articles = support_count + contradict_count
    if total_articles == 0:
        return 0.0  # No articles found

    trust_score = support_count / total_articles  # Simple ratio of support
    return trust_score
```

Implement LLM Responses

```
def predict_with_confidence(claim):
    """
    Predict the label for a claim and calculate the trust score based on re
    """
    model.eval()
    with torch.no_grad():
        inputs = prepare_input(claim)
        outputs = model(**inputs)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1).item()

    # Convert predicted label to word
    predicted_label_word = label_encoder.inverse_transform([predictions])[0

    # Retrieve articles related to the claim
    retrieved_articles = retrieve_articles(claim)

    # Calculate the trust score
    trust_score = calculate_trust_score(claim, retrieved_articles)

    # Flag low-confidence responses
    if trust_score < 0.5:  # Example threshold
        print(f"Low confidence for claim: '{claim}'. Trust score: {trust_sc
    else:
        print(f"High confidence for claim: '{claim}'. Trust score: {trust_s

    return predicted_label_word, trust_score
```

Execute

```
claim = "Study Vaccine for Breast Ovarian Cancer Has Potential"
predicted_label_word, trust_score = predict_with_confidence(claim)
print(f"Predicted label for the claim '{claim}': '{predicted_label_word}',
```

```
Low confidence for claim: 'Study Vaccine for Breast Ovarian Cancer Has Pote
Predicted label for the claim 'Study Vaccine for Breast Ovarian Cancer Has
```