Fine Tuning Large Language Models for Enterprise Application Project Report

---

1. Student Information

- **Student Name:** VAN HIEU NGUYEN
- **Student ID:** 221538422
- **Date Submitted:** 14 May 2025 [Updated 29 May 2025]

---

2. Project Introduction

- **Title of the Project: Fine-Tuning Large Language Models for Enterprise Applications: Medical Misinformation Detection**
- **What is the project about?**
  This project focuses on the fine-tuning of Large Language Models (LLMs) for domain-specific applications within the healthcare sector, with a specific emphasis on identifying and classifying medical misinformation. The goal is to improve the reliability and trustworthiness of AI-generated responses in medical contexts by customizing LLMs using datasets that include factual and misleading medical information.
- **Why is this project important or useful?**
  With the widespread deployment of AI in healthcare from virtual assistants to clinical decision support systems—ensuring the factual correctness of generated responses is critical. Misinformation can lead to harmful outcomes. Fine-tuning LLMs using Supervised Fine-Tuning (SFT) and BERT with Low-Rank Adaptation (LoRA) allows for domain-specific alignment, improves classification accuracy, and helps mitigate hallucinations and bias in medical language processing.

---

3. API/Token Setup

Objective: Access and use an LLM through a secure API with Gemini 2.0 Flash model for training and inference.

Instructions:

1. Specify which provider you're using:
   - Hugging Face: BERT (Bidirectional Encoder Representations from Transformers) method for pre-trained model
   - Google Vertex AI: implement fine-tuning for Gemini 2.0 Flash Model and deploy an complete AI chatbots for using.
2. List the **steps you followed** to generate the token:
   2.1. Pre-train model using BERT method

   - Step 1: Created account at https://huggingface.co/settings/tokens
   - Step 2: Navigated to the API/token section https://wandb.ai/authorize?ref=models
   - Step 3: Clicked on "Create new key"
   - Step 4: Copied the key and securely saved it: "7aecd1344b2f77be5dcc08d6ff6ecdb52886e5c9"
   - Step 5: View Run the project: https://wandb.ai/hieunguyen23032001-deakin-university/huggingface/runs/k7ad0pp8

   2.2. Fine Tuning model using Gemini 2.0 Flash Experiment Model:

   - Step 1: Created account at https://console.cloud.google.com/vertex-ai/studio/tuning?inv=1&invt=AbxQeA&project=sit319-25t1-nguyen-ae806d0
   - Step 2: Navigated to "Tuning" and select on "Create Tuned Model"
   - Step 3: Create a model detail:
     - Tuned Model name: "covid_tuning"
     - Based model: "gimini-2.0-flash-lite-001"
     - Region: "us-central1"
   - Step 4: Create Tuning dataset:
     - Upload train dataset to Google Cloud Storage: "gs://daft/Cleaned_Covid19_Train-7.jsonl"
     - Model Validation: upload validation dataset to Google Cloud Storage (optional)
   - Step 5: Start Tuning with Gemini 2.0 model. https://console.cloud.google.com/vertex-ai/generative/language/locations/us-central1/tuning/tuningJob/1012173862948831232?project=181085238689

3. **Screenshot or terminal output (required):**

```
/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" e
  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" e
  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:

VIEW TUNING JOB
```

4. **Secure Loading of Token in Code:**

Using: vertextai.init() by importing:

- 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

Using genai.Client for testing prompt and get respond as label (True, False, or Misleading)

Code: Load Token Securely

```python
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,
)
```

```python
from google import genai
from google.genai import types
import base64

def generate():
  client = genai.Client(
      vertexai=True,
      project="181085238689",
      location="us-central1",
  )

  msg3_text1 = types.Part.from_text(text="""Clearly the Obama administration did not leave any k

  model = "projects/181085238689/locations/us-central1/endpoints/5419770989749731328"
  contents = [
    types.Content(
      role="user",
      parts=[
        types.Part.from_text(text="""Multiple Facebook posts claim that Aussies will be fined if
      ]
    ),
    types.Content(
      role="model",
      parts=[
        types.Part.from_text(text=label)
      ]
    ),
    types.Content(
      role="user",
      parts=[
        msg3_text1
      ]
    ),
  ]
```

```python
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="")
```

Python

```
/usr/local/lib/python3.11/dist-packages/google/auth/_default.py:76: UserWarning: Your application has
  warnings.warn(_CLOUD_SDK_CREDENTIALS_WARNING)
fake
```

---

4. Environment Setup

- **Development Platform:**
  - Google Colab
  - Local Machine (macOS)
  - GPU Available? [✓] Yes
  - GPU Type (if applicable): Local T4 GPU
- **Python Version:** Python 3.10
- **Other Tools Used:** VS Code, Google Colab, Google Vertex AI, Google Cloud.

Code: Environment & GPU Check

```
1 !nvidia-smi
```

```
Tue May 13 06:00:11 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   70C    P0             30W /   70W |    2466MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

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

---

5. LLM Setup

- **Model Name:** Gemini 2.0 Flash (Experimental) and BERT method
- **Provider (OpenAI, Hugging Face):** Google Vertex AI (Gemini Model), Hugging Face (BERT)
- **Key Libraries & Dependencies (with versions)**
- **Libraries and Dependencies Required:**

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

```
[ ]  !pip install --upgrade google-genai
     !gcloud auth application-default login
```

```
[ ]  !pip install --upgrade google-cloud-aiplatform
```

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

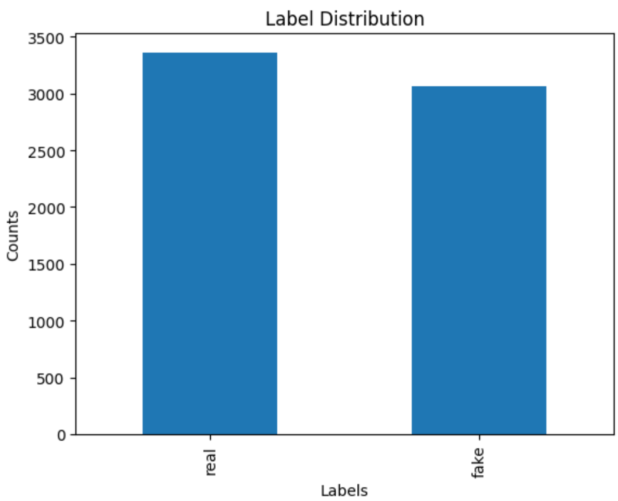
```
[2]  import re
     import os
     import json
     import pandas as pd
     import matplotlib.pyplot as plt
     import torch
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
     #Pre-train BERT:
     from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
     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
```

---

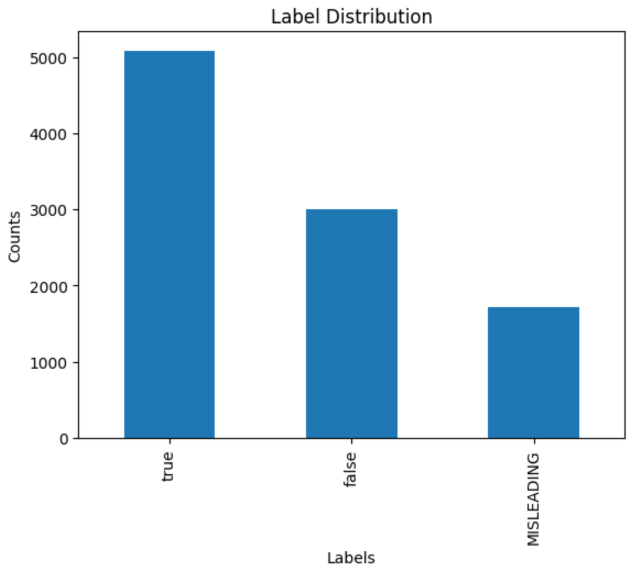
6. Dataset Description

- **Dataset Name & Source:**
  - HealthFact: General misinformation
  - SciFact: Scientific claim fact-checking
  - COVID-19 Fake News: Pandemic-related misinformation
- **Access Link (if public):**
  - Covid Fake New: https://github.com/diptamath/covid_fake_news?tab=readme-ov-file
  - Health Fact (Medical Misinformation Detection) dataset: https://github.com/neemakot/Health-Fact-Checking/blob/master/data/DATASHEET.md
  - Scifact dataset: https://scifact.s3-us-west-2.amazonaws.com/release/latest/data.tar.gz
- **Feature Dictionary / Variable Description:**
  - Claim: Textual medical claim
  - Label: true, false, and misleading
  - Evidence: Text used for verification (in two datasets)

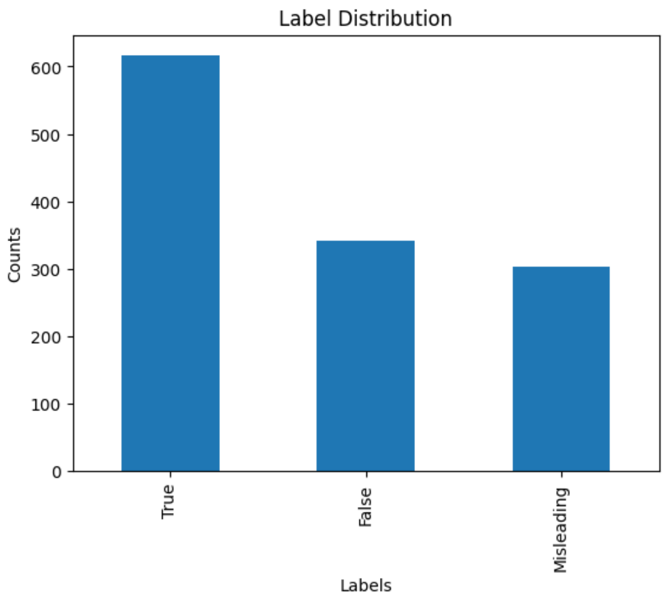
(1) Covid19 Fake New dataset:



(2) Healthfact Train Data:



(3) Scifact Train Data:



- **Was preprocessing done? If yes, describe:**
  - Pre-processing and cleaning for 3 datasets, and slits each dataset into three sub-set for train, test, and validate.
  - Standardized all datasets into 3-class classification: true, false, misleading.
  - Converted JSON into ".jsonl" for Vertex AI ingestion with the correct format.
  - Tokenized using Hugging Face tokenizer.
  - Balanced label distribution.

Code: Load & Pre-process Dataset

(1) Covid Fake News Dataset:

```python
# 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 numbers
        reconstructed_tweet = ' '.join(tokens)
```

```python
        # 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, or Misleading"  # Example instruct
                    }
                ]
            },
            "contents": [
                {
                    "role": "user",
                    "parts": [
                        {
                            "text": f"TRANSCRIPT: \n{reconstructed_tweet}\n\n LABEL:"
                        }
                    ]
                },
                {
                    "role": "model",
                    "parts": [
                        {
                            "text": label  # The label indicating the model's response
                        }
                    ]
                }
            ]
        }
        jsonl_data.append(jsonl_entry)
```

```python
    # Write the processed data to a JSONL file
    output_file = json_file.replace('.json', '.jsonl')  # Change the extension to .jsonl
    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.jsonl.
Processed Cleaned_Covid19_Dev.json and saved to Cleaned_Covid19_Dev.jsonl.
First few entries from claims_test_data:
[{'systemInstruction': {'role': 'assistant', 'parts': [{'text': 'Classification the content is Fake, Real, or Misl
```

(2) Health Fact Dataset:

```python
# 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 and numbers
                reconstructed_claim = ' '.join(tokens)

                # Prepare the JSONL entry in the required format
                jsonl_entry = {
                    "systemInstruction": {
                        "role": "assistant",  # Example role, adjust as needed
                        "parts": [
                            {
                                "text": "You are a helpful assistant."  # Example instruction, adjust as need
                            }
                        ]
                    },
                     "contents": [
                         {
                             "role": "user",
                             "parts": [
                                 {
                                     "text": f"CLAIM: {reconstructed_claim}\nEXPLANATION: {explanation}\nLABEL
                                 }
                             ]
                         },
                         {
                             "role": "model",
                             "parts": [
                                 {
                                     "text": label  # The label indicating the model's response
                                 }
                             ]
                         }
                     ]
                 }
                 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 original filenames as keys
     data_dict[json_file] = jsonl_data
# Access the data using the correct keys - the original filenames
healthfact_train_data = data_dict['healthfact_traindata.json']  # Corrected key
healthfact_test_data = data_dict['cleaned_healthfact_test.json']  # Corrected key
healthfact_dev_data = data_dict['cleaned_healthfact_dev.json']  # Corrected key
# Print the first few entries for verification
print(f"First few entries from healthfact_train_data:\n{healthfact_train_data[:5]}")
# 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 extension to .jsonl
    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 helpful assistant.'}]}, 'contents': [{'
Processed healthfact_traindata.json and saved to healthfact_traindata.jsonl.
Processed cleaned_healthfact_test.json and saved to cleaned_healthfact_test.jsonl.
Processed cleaned_healthfact_dev.json and saved to cleaned_healthfact_dev.jsonl.
```

(3) Scifact Dataset:

```python
# 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 and numbers
                reconstructed_claim = ' '.join(tokens)

                # Prepare the JSONL entry in the required format
                jsonl_entry = {
                    "systemInstruction": {
                        "role": "assistant",  # Example role, adjust as needed
                        "parts": [
                            {
                                "text": "You are a helpful assistant."  # Example instruction, adjust as need
                            }
                        ]
                    },
                    "contents": [
                        {
                            "role": "user",
                            "parts": [
                                {
                                    "text": f"CLAIM: {reconstructed_claim}\nEVIDENCE: {explanation}\nLABEL: {
                                }
                            ]
                        },
                        {
                            "role": "model",
                            "parts": [
                                {
                                    "text": label  # The label indicating the model's response
                                }
                            ]
                        }
                    ]
                }
                # Append the modified entry to the processed data list
                processed_data.append(jsonl_entry)  # Append the processed data
            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')  # Change the extension to _processed.json
    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 helpful assistant.'}]}, 'contents': [{'
Processed dev_3class.jsonl and saved to dev_3class_processed.jsonl.
Processed train_3class.jsonl and saved to train_3class_processed.jsonl.
```

7. Improving LLM Performance

Step #	Method	Description	Result Metric (Accuracy)
1	Zero-shot Prompt	No Training, Direct Response.	58%
2	Few-shot Prompt	No Training, Testing 1 Prompt, and No label Response, Unclear Response	58%
3	Temperature Tuning	10 epochs on Vertext AI, Tuned Temperature from 58% to 80%	80%
4	Fine-tuning	50 epochs on Vertext AI, Testing prompt with 98-100% correct response with labels.	98%

Code Snippets for Each Step

Before Training and Fine-tuning with few-shot training prompt:

```python
import google.generativeai as genai

# Initialize the Gemini 2.0 Flash Model:
API_KEY = "AIzaSyCjJIJ3ntglUAqZHn6qZaOE5uIGg4txEC4"
genai.configure(api_key=API_KEY)

# Load the Gemini model
model = genai.GenerativeModel("gemini-2.0-flash")
```

```python
# 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 government announcement Get more on the coronavirus data here https t co jvGZlSbFjH https t co PygSKXesBg"
"""

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 cli

**In Conclusion:**
```

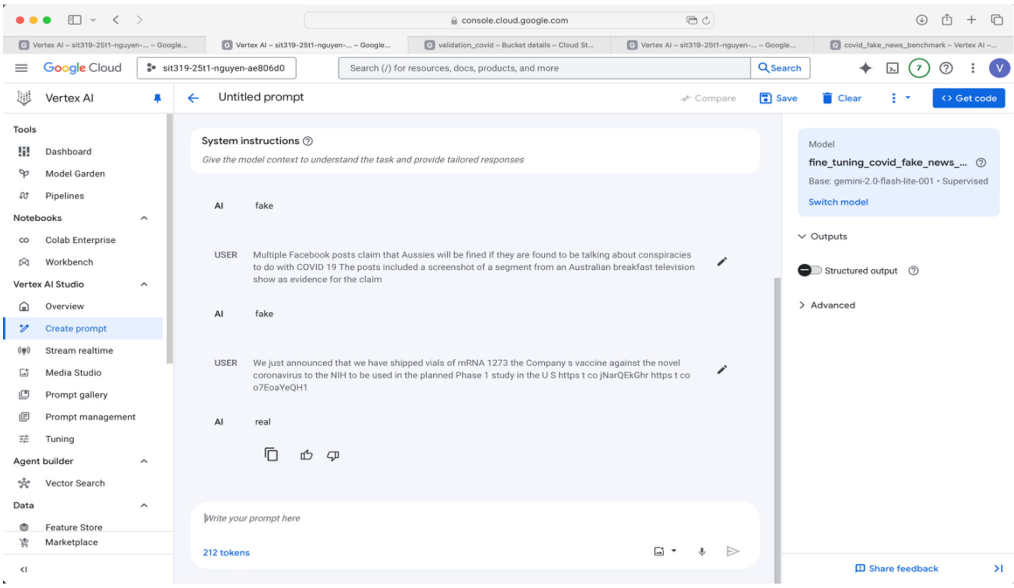
After Pre-train and Fine-tuning:

(1) One Shot Training Prompt

```python
model = "projects/181085238689/locations/us-central1/endpoints/5419770989749731328"
contents = [
    types.Content(
        role="user",
        parts=[
            types.Part.from_text(text="""Multiple Facebook posts claim that Aussies will be fined if they are found to be talking about conspiracies to do with COVID 19 The posts included a screenshot of a segment from an Australian breakfast telev
        ]
    ),
    types.Content(
        role="model",
        parts=[
            types.Part.from_text(text=label)
        ]
    ),
    types.Content(
        role="user",
        parts=[
            msg3_text1
        ]
    ),
]
```

```
/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" e
  warnings.warn(_CLOUD_SDK_CREDENTIALS_WARNING)
fake
```

(2) Fine-Tuning google Vertext AI tool for prompt testing:



8. Benchmarking & Evaluation

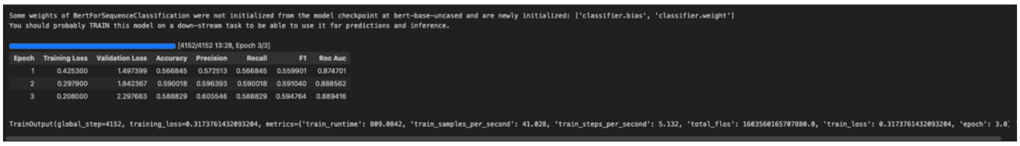
Required Components:

- **Metrics Used:**
  - Accuracy
  - Precision, Recall, F1 Score.
  - Confusion Matrix
- **Why those metrics?**

These metrics capture both correctness and type of error (especially important in medical misinformation detection). Moreover, using those metrics for comparing which evaluate as before and after applying fine-tuning model.

- **Benchmark Dataset & Sample Size:**

(1) Pre-train using BERT method for Covid Dataset:



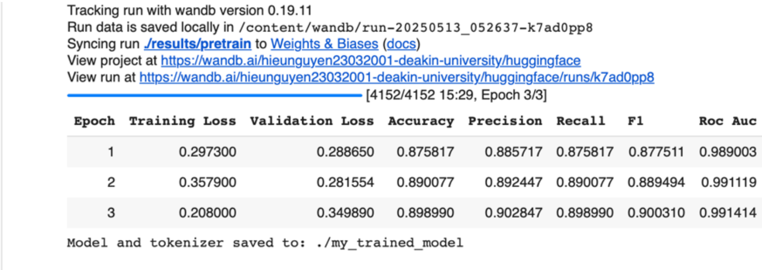
```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

[4152/4152 13:28, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1	Roc Auc
1	0.425300	1.497399	0.566845	0.572513	0.566845	0.559901	0.874701
2	0.297900	1.842367	0.590018	0.596393	0.590018	0.591040	0.888562
3	0.208000	2.297663	0.588829	0.605546	0.588829	0.594764	0.889416

```
TrainOutput(global_step=4152, training_loss=0.3173761432093204, metrics={'train_runtime': 809.0842, 'train_samples_per_second': 41.028, 'train_steps_per_second': 5.132, 'total_flos': 1603560165707880.0, 'train_loss': 0.3173761432093204, 'epoch': 3.0}
```

(2) Pre-train using BERT method for Combination Dataset:



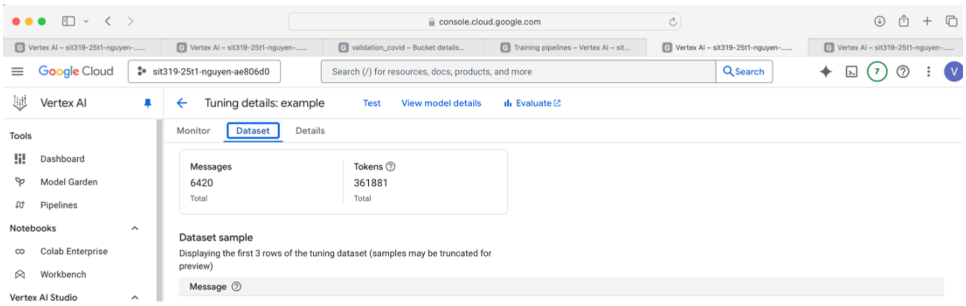
Tracking run with wandb version 0.19.11
Run data is saved locally in /content/wandb/run-20250513_052637-k7ad0pp8
Syncing run ./results/pretrain to Weights & Biases (docs)
View project at https://wandb.ai/hieunguyen23032001-deakin-university/huggingface
View run at https://wandb.ai/hieunguyen23032001-deakin-university/huggingface/runs/k7ad0pp8
[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

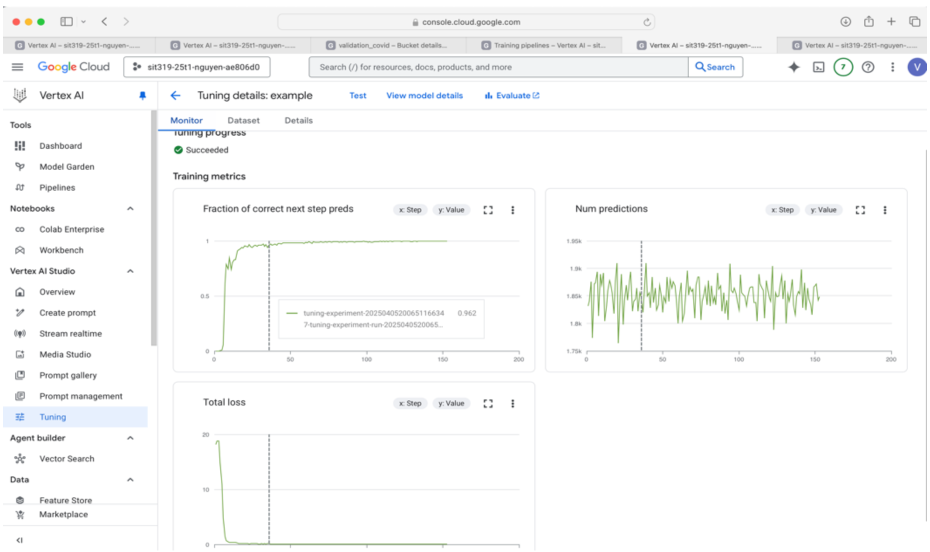
Model and tokenizer saved to: ./my_trained_model

(3) Fine Tuning for All dataset:

- Sample Size dataset for fine-tuning:



- Benchmark using Google Vertext AI:



(3) The Combination dataset between HealthFact and SciFact dataset:

```
# 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
```

```
# 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}")
```

[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
```

Interpretation:

The model was outstanding improve after fine tuning using Gemini 2.0 Flash model, it achieved almost 98 percent accuracy, compared with 54 to 58 percent using BERT method for pre-training model. This shows that fine-tuning using Gemini 2.0 Flash model makes the model more accuracy and more correct for the response.

---

9. Implement the Confidence Scoring System for Trust Score using PubMed medical retrieve articles

- **Tool Used:** Google Colab
- **Set up:**
  - Step 1: Access Pub Med website at https://pubmed.ncbi.nlm.nih.gov/
  - Step 2: Navigated to the API via url and query from the website
  - Step 3: Implement retrieve article based on the query
  - Step 4: Implement calculate trust score based on LLMs prediction
  - Step 5: Launch prediction based on trust score calculated between 50% for evaluating whether the query is low confidence or not
- **Code:**

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

Show hidden output

```
!pip install biopython
```

Show hidden output

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

Show hidden output

```
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 ac
        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
```

```
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
```

```python
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])[

    # 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
```

- **Output:**

Execute

```python
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}',
```

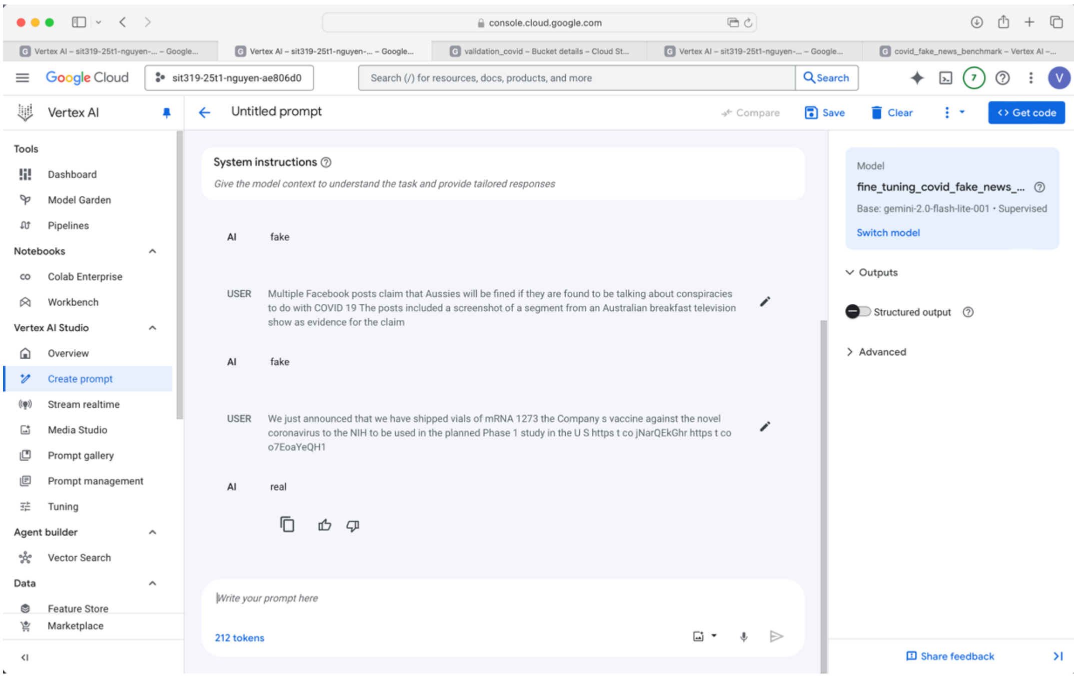
```
Low confidence for claim: 'Study Vaccine for Breast Ovarian Cancer Has Pote
Predicted label for the claim 'Study Vaccine for Breast Ovarian Cancer Has
```

---

10. UI Integration

- **Tool Used:** Google AI Cloud, Vertex AI
- **Key Features of the Interface:**
  - Accepts a medical question as input, and response with label.
  - Displays classification label (True, False, Misleading).
  - Displays confidence score.

- **Include Screenshots of Working UI:** using Google Cloud and Vertext AI for deployment chatbots in Vertex AI Studio



10. References

[1]. Parthasarathy B.V, Zafar A, Khan A, & Shahid A (August 2024), "The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities (Version 1.0)", Arxiv, CeADAR Connect Group. https://arxiv.org/html/2408.13296v1#Ch4.S1

[2]. Huizenga E (13 December 2024), "Developer's guide to getting started with Gemini 2.0 Flash on Vertex AI", Medium.com, https://medium.com/google-cloud/developers-guide-to-getting-started-with-gemini-2-0-flash-on-vertex-ai-6b4fe3c6899f

[3]. Youtube (17 February 2024), "Fine Tuning a Model in Gemini and Vertext AI | Steps to make a LLM". https://www.youtube.com/watch?v=ej_ZUcyKpoc&t=218s

[4]. Google Cloud (29 April 2025), "About Supervised Fine-Tuning for Gemini models". https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini-supervised-tuning

[5]. Huizenga E (19 November 2024), "A Step-by-Step Guide to Fine-Tuning Gemini for Question Answering". https://medium.com/google-cloud/a-step-by-step-guide-to-fine-tuning-gemini-for-question-answering-8b3fb117dbbf

[6]. Huizenga E (10 February 2025), "Fine-tuning Gemini: Best Practices for Data, Hyperparameters, and Evaluation". https://medium.com/google-cloud/fine-tuning-gemini-best-practices-for-data-hyperparameters-and-evaluation-65f7c7b6b15f