

# 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
- 

## 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: "7aecdl344b2f77be5dcc08d6ff6ecdb52886e5c9"
    - 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&inv=AbxQeA&project=sit319-25t1-nguyen-ac806d0>
    - 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:78: 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
Warning: warn(_E2002_SDK_QUOTA_PROJECT_WARNING)
INFO:vertexai.tuning._tuning:Creating SupervisedTuningJob
/usr/local/lib/python3.11/dist-packages/google/auth/_default.py:78: 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
Warning: warn(_E2002_SDK_QUOTA_PROJECT_WARNING)
INFO:vertexai.tuning._tuning:SupervisedTuningJob created. Resource name: projects/181085238689/locations/us-central1/tuningJobs/1812173862948831232
INFO:vertexai.tuning._tuning:To see this SupervisedTuningJob in another session:
INFO:vertexai.tuning._tuning:tuning_job = sft.SupervisedTuningJob("projects/181085238689/locations/us-central1/tuningJobs/1812173862948831232")
INFO:vertexai.tuning._tuning:View Tuning Job:
...
[VIEW TUNING JOB]
```

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

Using: `vertexai.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

```
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,
)

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
        ]
    ),
]
```

```
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	Perf	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp		Pwr:Usage/Cap		Memory-Usage	GPU-Util Compute M.
						MIG M.
0	Tesla T4	P0	Off	00000000:00:04:0	Off	0
N/A	70C		30W / 70W	2466MiB / 15360MiB		Default N/A

Processes:							GPU Memory Usage
GPU	GI	CI	PID	Type	Process name		
ID	ID						

#### 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
```

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

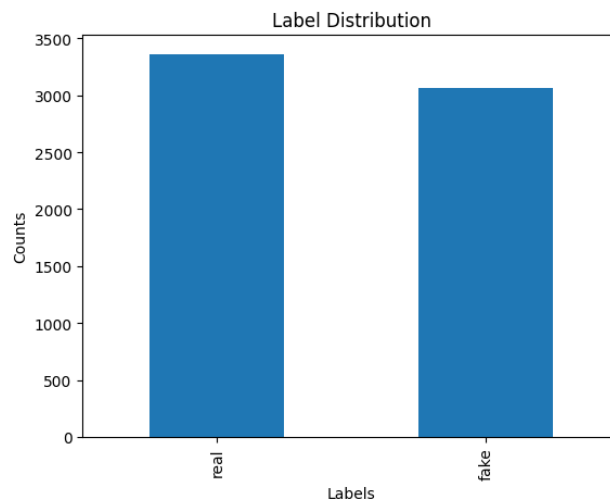
```
[ ] 1 !pip install --upgrade google-cloud-aiplatform
```

```
▶ 1 !b2b install tensorflow for CPU redness:22 penguin:2020
```

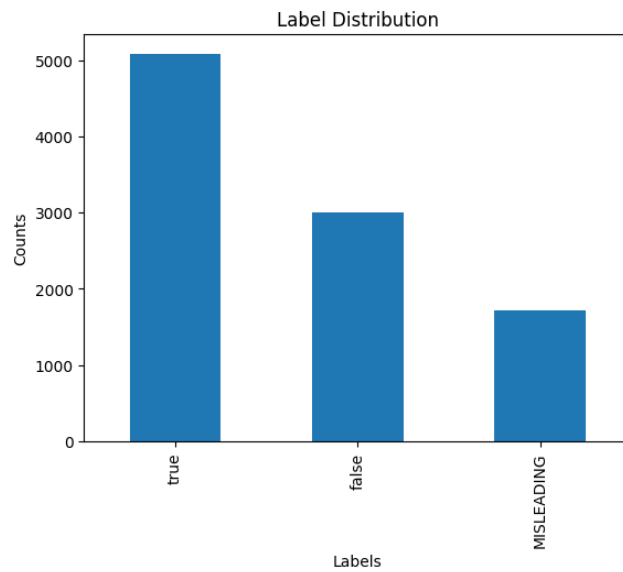
```
[2] 1 import re
    2 import os
    3 import json
    4 import pandas as pd
    5 import matplotlib.pyplot as plt
    6 import torch
    7 from sklearn.model_selection import train_test_split
    8 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
    9 #Pre-train BERT:
   10 from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
   11 from sklearn.preprocessing import LabelEncoder
   12 from peft import get_peft_model, LoraConfig
   13 #Confidence Score System:
   14 import requests
   15 from Bio import Entrez
   16 from langchain import LLMChain, PromptTemplate
   17 from langchain.chains import RetrievalQA
   18 from langchain.vectorstores import FAISS
   19 from langchain.embeddings import OpenAIEmbeddings
   20 #Gemini Model:
   21 from google import genai
   22 from google.genai import types
   23 import base64
   24 import google.generativeai as genai
```

## 6. Dataset Description

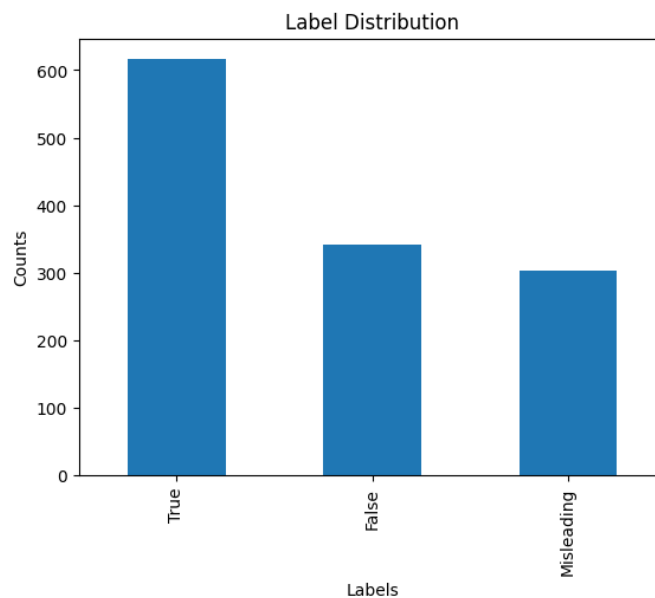
- **Dataset Name & Source:**
    - HealthFact: General misinformation
    - SciFact: Scientific claim fact-checking
    - COVID-19 Fake News: Pandemic-related misinformation
  - **Access Link (if public):**
    - Covid Fake New: [https://github.com/diptamath/covid\\_fake\\_news?tab=readme-ov-file](https://github.com/diptamath/covid_fake_news?tab=readme-ov-file)
    - 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 splits 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:**

```

1 # List of JSON files to process
2 json_files = [
3     'Cleaned_Covid19_Train.json',
4     'Cleaned_Covid19_Dev.json',
5 ]
6 data_dict = {}
7 # Process each JSON file
8 for json_file in json_files:
9     # Load the dataset
10    with open(json_file, 'r') as file:
11        data = json.load(file)
12
13    # Prepare a list to hold the processed data
14    jsonl_data = []
15
16    # Extract and process each entry
17    for entry in data:
18        # Extract the id, tweet, and label
19        tweet = entry['tweet']
20        label = entry['label']
21
22        # Tokenize the tweet
23        tokens = re.findall(r'\b\w+\b', tweet) # Keep only words and numbers
24        reconstructed_tweet = ' '.join(tokens)
25
26        # Prepare the JSONL entry with the required structure
27        jsonl_entry = {
28            "systemInstruction": {
29                "role": "assistant", # Example role, adjust as needed
30                "parts": [
31                    {
32                        "text": "Classification the content is Fake, Real, or Misleading" # Example instruct
33                    }
34                ]
35            },
36            "contents": [
37                {
38                    "role": "user",
39                    "parts": [
40                        {
41                            "text": f"TRANSCRIPT: \n{reconstructed_tweet}\n\n LABEL:"
42                        }
43                    ]
44                },
45                {
46                    "role": "model",
47                    "parts": [
48                        {
49                            "text": label # The label indicating the model's response
50                        }
51                    ]
52                }
53            ]
54        }
55        jsonl_data.append(jsonl_entry)
56
57    # Write the processed data to a JSONL file
58    output_file = json_file.replace('.json', '.jsonl') # Change the extension to .jsonl
59    with open(output_file, 'w') as outfile:
60        for entry in jsonl_data:
61            json.dump(entry, outfile)
62            outfile.write('\n') # Write each entry on a new line
63    print(f"Processed {json_file} and saved to {output_file}.")
64    data_dict[json_file] = jsonl_data
65
66 # Access the data using the correct keys - the original filenames
67 covid_train_data = data_dict['Cleaned_Covid19_Train.json'] # Corrected key
68 covid_dev_data = data_dict['Cleaned_Covid19_Dev.json'] # Corrected key
69 # Print the first few entries for verification
70 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 Misleading']}}]

## (2) Health Fact Dataset:

```

5 # List of JSON files to process
6 json_files = [
7     'healthfact_traindata.json',
8     'cleaned_healthfact_test.json',
9     'cleaned_healthfact_dev.json'
10 ]
11 data_dict = {}
12 # Process each JSON file
13 for json_file in json_files:
14     # Prepare a list to hold the processed data
15     jsonl_data = []
16     # Load the dataset
17     with open(json_file, 'r') as file:
18         # Read each line as a separate JSON object
19         for line in file:
20             try:
21                 entry = json.loads(line)
22                 # Extract the claim, explanation, and label
23                 claim = entry['claim']
24                 explanation = entry['explanation']
25                 label = entry['label']
26
27                 # Tokenize the claim
28                 tokens = re.findall(r'\b\w+\b', claim) # Keep only words and numbers
29                 reconstructed_claim = ' '.join(tokens)
30
31                 # Prepare the JSONL entry in the required format
32                 jsonl_entry = {
33                     "systemInstruction": {
34                         "role": "assistant", # Example role, adjust as needed
35                         "parts": [
36                             {
37                                 "text": "You are a helpful assistant." # Example instruction, adjust as need
38                             }
39                         ]
40                     },
41                     "contents": [
42                         {
43                             "role": "user",
44                             "parts": [
45                                 {
46                                     "text": f"CLAIM: {reconstructed_claim}\nEXPLANATION: {explanation}\nLABEL"
47                                 }
48                             ]
49                         },
50                         {
51                             "role": "model",
52                             "parts": [
53                                 {
54                                     "text": label # The label indicating the model's response
55                                 }
56                             ]
57                         }
58                     ]
59                 }
60                 jsonl_data.append(jsonl_entry)
61             except json.JSONDecodeError as e:
62                 print(f"Error decoding JSON: {e}")
63             # Use the correct key to store the data in the dictionary - keep the original filenames as keys
64             data_dict[json_file] = jsonl_data
65 # Access the data using the correct keys - the original filenames
66 healthfact_train_data = data_dict['healthfact_traindata.json'] # Corrected key
67 healthfact_test_data = data_dict['cleaned_healthfact_test.json'] # Corrected key
68 healthfact_dev_data = data_dict['cleaned_healthfact_dev.json'] # Corrected key
69 # Print the first few entries for verification
70 print(f"First few entries from healthfact_train_data:\n{healthfact_train_data[:5]}")
71 # Optionally, write the processed data to JSONL files
72 for json_file, jsonl_data in data_dict.items():
73     output_file = json_file.replace('.json', '.jsonl') # Change the extension to .jsonl
74     with open(output_file, 'w') as outfile:
75         for entry in jsonl_data:
76             json.dump(entry, outfile)
77             outfile.write('\n') # Write each entry on a new line
78     print(f"Processed {json_file} and saved to {output_file}.")
79     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': [{'role': 'user', 'parts': [{'text': 'CLAIM: {reconstructed_claim}\nEXPLANATION: {explanation}\nLABEL'}]}, {'role': 'model', 'parts': [{'text': label}]}]}]

```

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:

```
4 # List of JSONL files to process
5 jsonl_files = [
6     'dev_3class.jsonl',
7     'train_3class.jsonl'
8 ]
9 data_dict = {}
10 # Process each JSONL file
11 for jsonl_file in jsonl_files:
12     # Prepare a list to hold the processed data
13     processed_data = []
14
15     # Load the dataset
16     with open(jsonl_file, 'r') as file:
17         for line in file:
18             try:
19                 entry = json.loads(line)
20
21                 # Extract the claim, explanation, and label
22                 claim = entry['claim']
23                 explanation = entry['evidence_text']
24                 label = entry['label']
25
26                 # Tokenize the claim
27                 tokens = re.findall(r'\b\w+\b', claim) # Keep only words and numbers
28                 reconstructed_claim = ' '.join(tokens)
29
30                 # Prepare the JSONL entry in the required format
31                 jsonl_entry = {
32                     "systemInstruction": {
33                         "role": "assistant", # Example role, adjust as needed
34                         "parts": [
35                             {
36                                 "text": "You are a helpful assistant." # Example instruction, adjust as needed
37                             }
38                         ]
39                     },
40                     "contents": [
41                         {
42                             "role": "user",
43                             "parts": [
44                                 {
45                                     "text": f"CLAIM: {reconstructed_claim}\nEVIDENCE: {explanation}\nLABEL: {"
46                                 }
47                             ]
48                         },
49                         {
50                             "role": "model",
51                             "parts": [
52                                 {
53                                     "text": label # The label indicating the model's response
54                                 }
55                             ]
56                         }
57                     ]
58                 }
59
60                 # Append the modified entry to the processed data list
61                 processed_data.append(jsonl_entry) # Append the processed data
62             except json.JSONDecodeError as e:
63                 print(f"Error decoding JSON: {e}")
64
65     # Store the processed data in the dictionary
66     data_dict[jsonl_file] = processed_data
67
68     # Access the data using the correct keys - the original filenames
69     scifact_train_data = data_dict['train_3class.jsonl'] # Corrected key
70     scifact_test_data = data_dict['dev_3class.jsonl'] # Corrected key
71
72     # Print the first few entries for verification
73     print(f"First few entries from scifact_train_data:\n{scifact_train_data[:5]}")
74
75     # Optionally, write the processed data to new JSONL files
76     for jsonl_file, processed_data in data_dict.items():
77         output_file = jsonl_file.replace('.jsonl', '_processed.jsonl') # Change the extension to _processed.jsonl
78         with open(output_file, 'w') as outfile:
79             for entry in processed_data:
80                 json.dump(entry, outfile)
81                 outfile.write('\n') # Write each entry on a new line
82
83     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': [{'role': 'user', 'parts': [{'text': 'CLAIM: {reconstructed\_claim}\nEVIDENCE: {explanation}\nLABEL: {'

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 Vertex AI, Tuned Temperature from 58% to 80%	80%
4	Fine-tuning	50 epochs on Vertex 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:



```

import google.generativeai as genai

# Initialize the Gemini 2.0 Flash Model
API_KEY = "AIzaSyCj113hgUmg2m6z203u3Jg4t4dC4"
genai.configure(api_key=API_KEY)

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

# 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: "6:18 Sky's Ed Conway explains the latest COVID-19 data and government announcement. Get more on the coronavirus data here https://t.co/jvZL5BfJH https://t.co/Pyg0Kx5dy"
"""

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:

- "6:18 PM": This likely refers to the time the tweet was posted, possibly 6:18 AM or PM.
- "Sky News": This is likely a shortened form of "Sky News".
- "Ed Conway": This is likely the Twitter handle for Ed Conway, who is likely a Sky News correspondent.
- "explains the latest COVID-19 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/jvZL5BfJH https://t.co/Pyg0Kx5dy": 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:

- Misinformation: 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:

```

After Pre-train and Fine-tuning:

### (1) One Shot Training Prompt

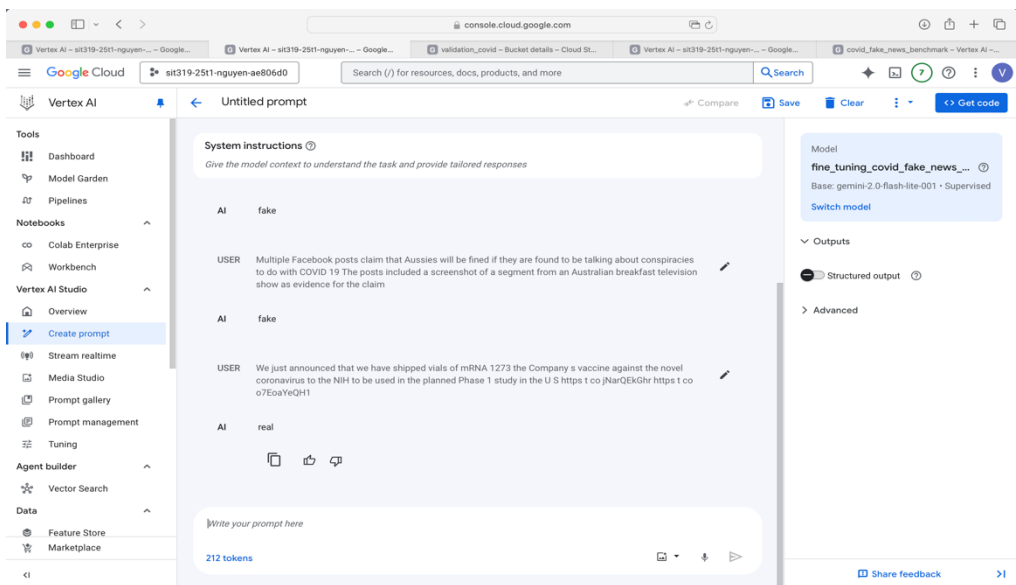
```

model = "projects/381885238689/locations/us-central1/endpoints/5413778089748731128"
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 television show as evidence for the claim."
        )
    ],
    types.Content(
        role="model",
        parts=[
            types.Part.from_text(text="fake")
        ]
    ),
    types.Content(
        role="user",
        parts=[
            types.Part.from_text(text="We just announced that we have shipped vials of mRNA 1273 the Company's vaccine against the novel coronavirus to the NIH to be used in the planned Phase 1 study in the U.S. https://t.co/jvZL5BfJH https://t.co/Pyg0Kx5dy")
        ]
    ),
    types.Content(
        role="model",
        parts=[
            types.Part.from_text(text="real")
        ]
    )
]

... /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.
... warnings.warn(_CLOUD_SDK_CREDENTIALS_WARNING)

```

### (2) Fine-Tuning google Vertex 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.
```

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1	Roc Auc
1	0.426300	1.497399	0.566845	0.572113	0.566845	0.569901	0.874701
2	0.297900	1.842387	0.590018	0.596393	0.590018	0.591543	0.888562
3	0.208000	2.297663	0.588929	0.605646	0.588929	0.594704	0.889416

```
trainOutput[global_step=4152, training_loss=0.3173761432893286, metrics={'train_runtime': 889.8842, 'train_samples_per_second': 61.828, 'train_steps_per_second': 5.132, 'total_flos': 1683168165787888.0, 'train_loss': 0.3173761432893286, 'epoch': 3.6
```

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

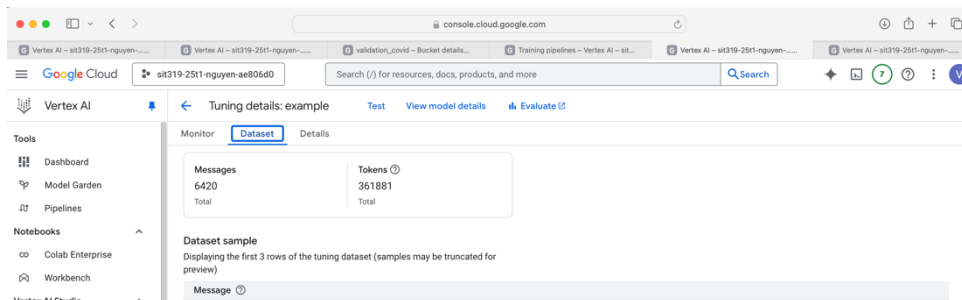
Tracking run with wandb version 0.19.11  
 Run data is saved locally in /content/wandb/run-20250513\_052637-k7ad0pp8  
 Syncing run [/results/pretrain](#) to [Weights & Biases \(docs\)](#)  
 View project at <https://wandb.ai/hieunguyen23032001-deakin-university/huggingface>  
 View run at <https://wandb.ai/hieunguyen23032001-deakin-university/huggingface/runs/k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

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

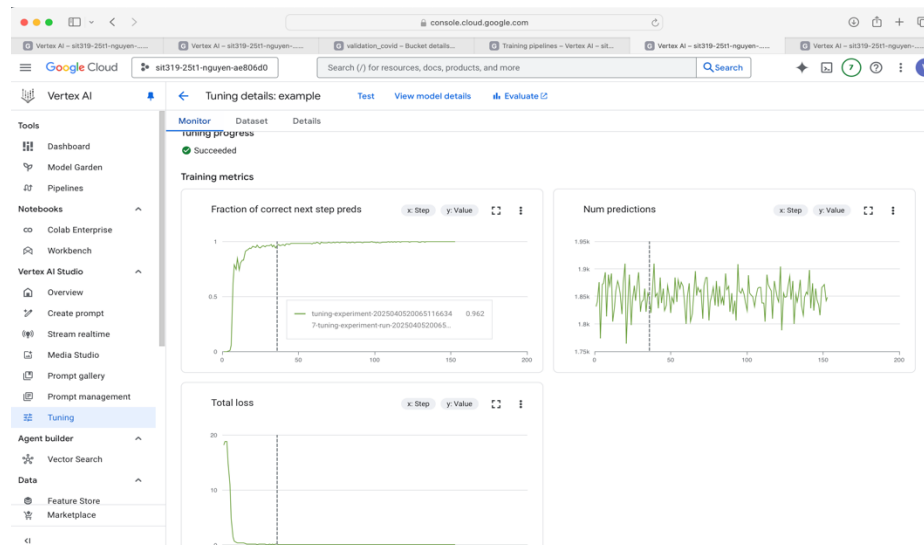
Model and tokenizer saved to: ./my\_trained\_model

(3) Fine Tuning for All dataset:

- Sample Size dataset for fine-tuning:



- Benchmark using Google Vertex AI:

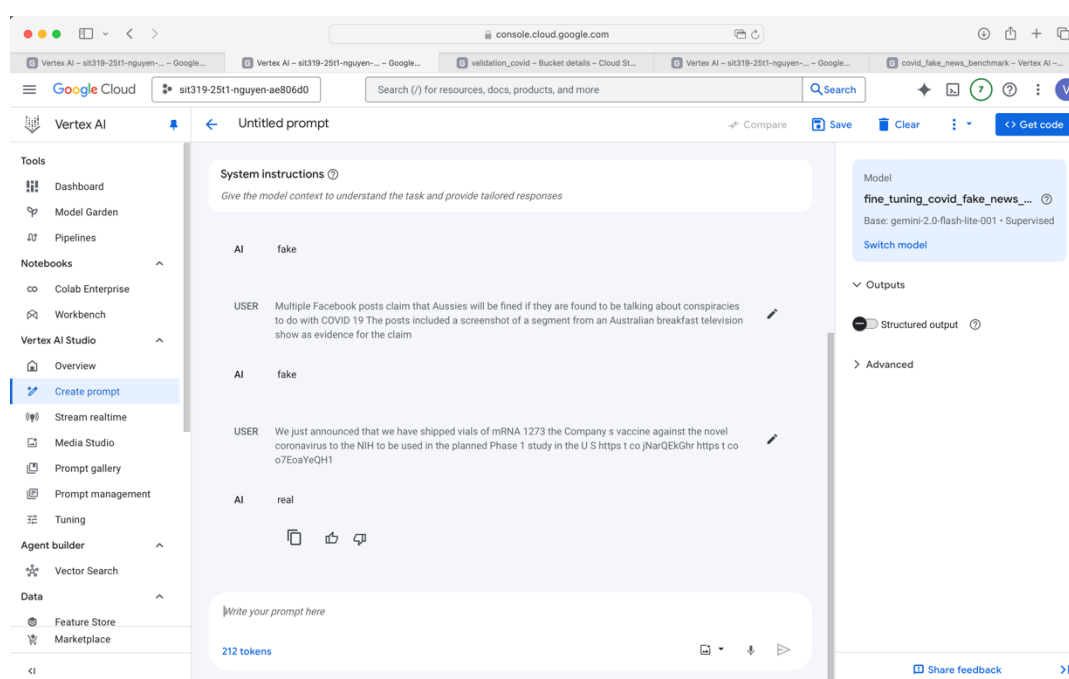


## Interpretation:

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. 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 Vertex 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 Vertex AI | Steps to make a LLM". [https://www.youtube.com/watch?v=ej\\_ZUcyKpoc&t=218s](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>