

# Google AI Course

## Part 2

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

### LLM OPS

AWS Sagemaker → can be used for Request, Train, Evaluate, Deploy



can be used

to use deep learning packages

→ API Gateway → API Handler

User

→ cost

will charge huge amount of money

LLMops platforms

Google → [Vertex AI] ← MLOps  
290M \$

AWS → [Bedrock] ← LLMops  
→ Hosted

LLM API → [LLM]

Platform API  
Response

First used for MLOps but as openAI emerged they introduced Model Garden for LLMops

Previous

ML Platform

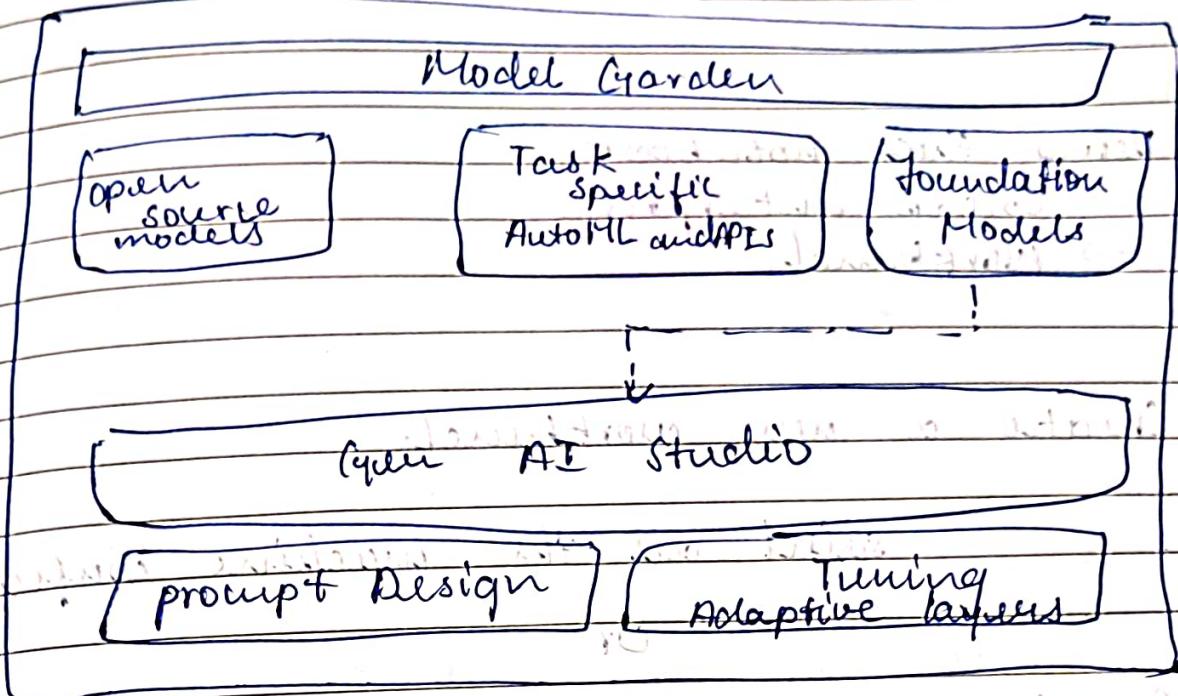
Data Science Workbench

[Experiment]

[Train]

[Deploy]

ML OPS

Nowpracticed

# Search for "gcp console"

create a project account

Search for "Vertx AI" tutorial

choose Model Garden on the left bar

# you can checkout everything

training, fine-tuning, generating, summarizing, etc.

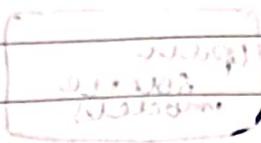
fine-tuned

workshop

## Vertebral A7 Handbook

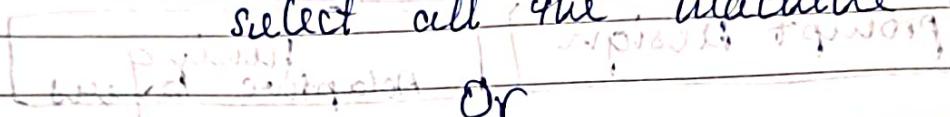
→ Enable all recommended APIs

- \* Using their notebook
- Collab enterprise
  - Workbench



→ Create a new workbench

select all the machine configs



Or

→ Create a collab entpr notebook which is collab like interface.

→ connect the notebook

can do all the coding

→ !pip install --upgrade google-cloud-aiplatform

→ Fix with no existing Jupyter kernel

→ Import vertebral

pathlib, random and math

\* Authenticate your notebook environment (Collab only)

→ Import sys

get from  
chatGPT

Link: [https://colab.research.google.com/drive/1JGKXWzvDyfjwCQHgkVYIwLcOOGdPmBx](#)

+ if using from colab enterprise you don't need to authenticate it will work.

To load a model

→ from `vertexai.preview.generative_models`  
`import (`  
`GenerativeModel)`

→ `model = GenerativeModel("gemini-1.0-pro")`

→ our model is loaded

→ can use any model from  
 model园地

→ `responses = model.generate_content("Why is`

`the sky blue?", stream=True)`

→ `print(responses)`

→ for response in responses:  
`print(response.text, end="")`

→ and can customize as we want

Model parameters

support this

→ generation\_config = GenerationConfig(  
 temperature=0.9,  
 top\_p=1.0,  
 top\_k=32,  
 candidatc\_count=1  
 max\_tokens=8192,

)

responses = model.generate\_content(  
 "why is the sky blue?",  
 generation\_config=generation\_config,  
 stream=True,

for res in res:  
 print(res.text, end="")  
 → "Hello world"

for res in res:  
 print(res.text, end="")  
 → "Hello world"

\* To make the chat remember all previous messages

→ chat = model.start\_chat()

prompt = " " "

responses = chat.send\_message(prompt, stream=True)

for res

Now we will remember that prompt:

Now if we ask something related to that

→ prompt = "chat.send.message(p)"

for student

and it will continue from same

content from chat of prev

\* lets use vision model

→ multimodal-model = Generative Model ("Gemini-1.0")

\* Define helper functions to load and display images

\* you can get the code from

github.com/charleschat/gpt

and can do many stuff with  
videos and images

f How to use Vertex AI from your local machine VSC

→ first install gcp cli from google

→ Now open VSC  
setup virtual env  
install the requirements  
google-cloud-aiplatform  
streamlit  
python-dotenv

Then do these commands

→ gcloud init

then do some required connection

you will get project ID and email  
again save it and at  
paste in the .env file

app.yaml choose region

Now lets run and paste the  
region in .env too

main.py

```
import tensorflow as tf  
import streamlit as st  
import tensorflow  
from vertebrae import preview_generative_models  
import (  
    GenerationConfig,  
    GenerativeModel  
)  
from dotenv import load_dotenv  
load_dotenv()
```

bind on port 8000

project\_id = os.getenv("PROJECT\_ID")

project\_region = os.getenv("REGION")

```
vertebrae.lit(project_id=_____, location  
= project_region)
```

model = GenerativeModel("generative-0-0-prj")

model.train(1000000, train\_steps=1000000)

this is how we do it

project\_id = "generative-0-0-prj"  
model = GenerativeModel(  
 project\_id="generative-0-0-prj",  
 region="us-central1",  
 model\_name="generative-0-0-prj",  
 model\_version="1",  
 max\_parallelism=1, max\_retries=1, max\_workers=1, timeout=600)

# RAG using VertexAI

→ Chatgpt or watch the video at 19:30:00 to understand

What is prompting and co-pilot  
↳ Trap  
↳ original prompting  
↳ fine-tuning

## Fine-tuning with VertexAI

everything is done on cloud  
↳ you should just prepare your dataset training

must be in json format

format:

{'input\_text': text, 'label': label}

output\_text:

if we want this is the data format

can convert using  
many converted  
in python

classmate

Date  
Page

run colab enterprise

→ ! pip install google-cloud-aiplatform  
--user datasets  
--user gcp-cloud-pipeline-components

pip install google-cloud-aiplatform

(optional)

If not in enterprise run only colab

→ import IPython  
from google.cloud import aiplatform  
from google.colab import auth as google\_auth

google.auth.authenticate\_user()

token saved in colab

token saved in colab

\* login and pair token and stuff

Select the project

→ import vertexai

PROJECT\_ID = "PROJECT\_ID"

vertexai.init(project=PROJECT\_ID)

regions

project\_id

cloudRegion

→ ! gcloud config set project \$project\_id

Import some necessary libs

ChatGPT or at 19:51:00

2023-09-20 Dallas 10:13

→ df = pd.read\_json('data/df.json')  
df['fine\_tuned'] = True

→ df.shape  
(744, 2)

after tuning  
a name

→ model.display\_name = "bbc-finetuned-model"  
finetuned\_model = TextGenerationModel.from\_pretrained("bbc/bbc-fineturned",  
weights\_path="models/bbc-finetuned",  
model\_type="text-generation", revision="@002")

finetuned\_model.tune\_model(

training\_data=df,  
train\_steps=100, epochs=1)

training\_job\_location="europe-west4",  
finetuned\_model\_location="models/bbc-finetuned",  
) # this takes

trials\_left=1000  
initial\_trial=1000

→ "will take some" → AT TAKES

you will see a URL

open it →

and you can see

the pipeline

of training, testing the, fit and logs

it will store everything in bucket of  
and pre-trained eval + google cloud

0:17:19 to this bucket will go to  
the endpoint

and we click the model garden  
 and click on MY  
 ENDPOINTS AND  
 MODELS

we see the model present

Now we can use the trained model

→ response = tuned\_model.predict(  
 I have another in my way)  
 print(response[0])

→



Or if we want to load the model from  
 the ENDPOINT

→ Tuned\_model\_name = tuned\_model.ENDPOINT.give  
 resource.deployedModel[D].model  
 tuned\_model\_1 = TextGenerationModel.get\_tuned-  
 model(tuned\_model\_name)

response = tuned\_model\_1.predict()

print(response[0])

# AWS Bedrock

Via

API

GPT, TTS, QnA

UI

in AWS Console

LMOps

Platform

→ Get Started

\* If Using Lambda request for model Access

\* In the Foundation model you can see custom model you can fine tune in the UI itself

\* To Set up in on VS Code

\* Setup virtual environment

\* Requirements.txt

language development environment

language community

aws-sdk-lambda

Boto3

python-dotenv

we will

make connection

to AWS

(AWS Lambda) API

- Create IAM user  
and in policy choose  
AmazonS3FullAccess
  - In secret key option create an access  
key in terminal keygen  
use cmd again until you  
create access key
  - Search for Ahk cli and install → ~~Terminal~~
- In terminal
- Ahk configure
- Copy that access key  
and paste it here  
and give the secret access key too  
and give the region  
mention these in the .env file too

After that

Linux terminal = jkl...jkl...jkl...

↳ download = w

bilbo = k...k...k...k...

gandalf = t...t...t...t...

Profanity = proximal...distal

main.py

```

→ from langchain.llms import Bedrock
" " " " .chatus import LMChain
" " " " .prompts import PromptTemplate
import boto3
import os
import streamlit as st
from sloten import load_slots

```

load\_slots()

```

aws_access_key_ids
aws_secret
region_name

```

## # boto3 client

```

bedrock = boto3.client(
    service_name="Bedrock-runtime",
    region_name="us-east-1",
    aws_access_key_id="",
    aws_secret_access_key="",
    config=Config()
)

```

This will auth

```
model_id = "mistral-nlp"
```

```
llm = Bedrock(
```

```
model_id= model_id
```

```
client= Bedrock
```

```
model_kwarg= {"temperature": 0.4}
```

def my\_chatbot (language, user\_tent):

new\_message = PromptTemplate("You are a chatbot. You are in language {language}. In user tent {user\_tent}"))

input\_variables = ["language", "user\_tent"]

(The template is: "You are a chatbot. You are in language {language}. In user tent {user\_tent}")

return message

Bedrock\_chain = CLMChain (llm = llm, prompt = prompt)

response = Bedrock\_chain({"language": language, "user\_tent": user\_tent})

return response

this function

will take lang  
and tent

and give response

sf . title ("Bedrock Chat Demo")

language = sf . sidebar . selection ("language",

[ "english", "spanish",  
"hindi" ] )

if language:

user\_tent = sf . sidebar . tent area (label =

"What is your ques?",  
max\_chars = 100)

if user enters text "pingus") it takes user response from my chatbot (language, user it was, appropriate) and shows two in turn)

so, code starts with response ("friend")  
as a command in the  
"if friend run"

## # In terminal

→ Streamlit) reads main.py which is  
responsible for displaying dashboard & implementing  
functionality.

so we see our responses  
are very fast  
as we are using  
Cloud platforms

(can implement RQ from  
this too

("push check" after "file.H.P. + 2.  
video  
at the

Suppose) method. and this is end up original

platform, "J.  
thing"

So now first, and then file = first run  
keep in mind (code = class, so we