Gen AI Course
Part 2

LLMOPS

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

DLC ← in AWS can be used to use deep learning packages

API Gateway → API → user

→cost

will charge huge amount of money

LLMops platforms

Google → Vertex AI ← MLops

AWS → Bedrock ← LLMops

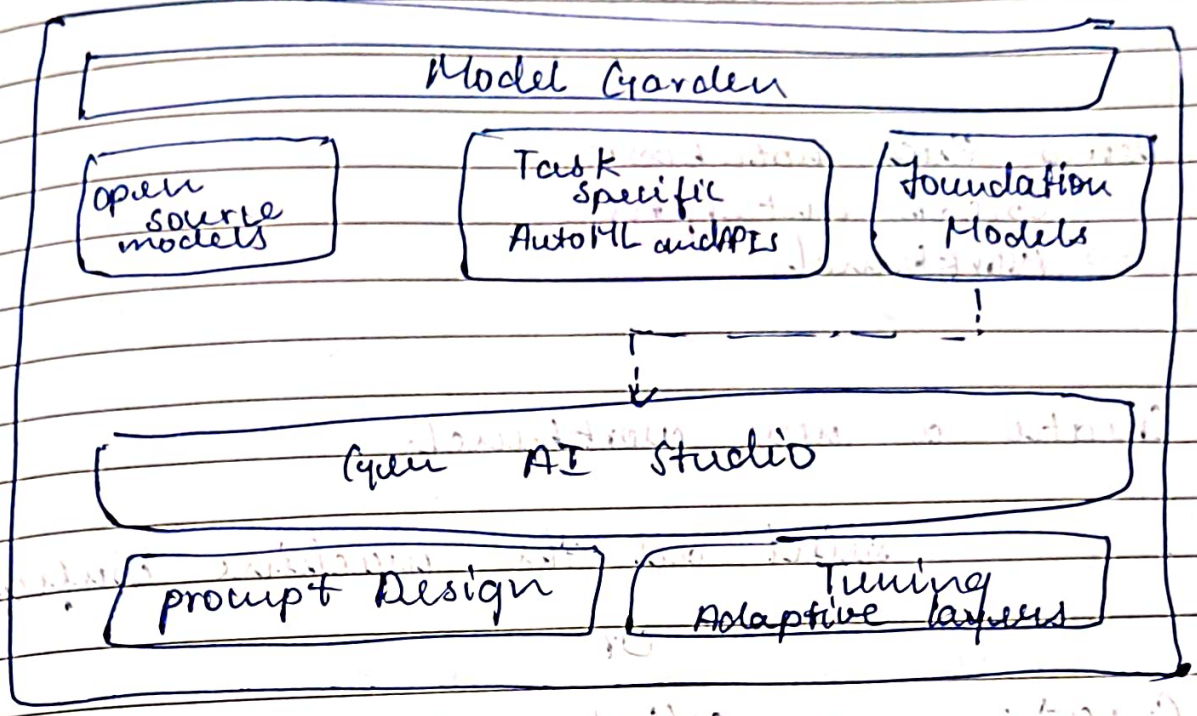
→ hosted

Vertex AI

First used for MLops but as Gen AI emerged they introduced Model Garden for LLMops

Previous

Now



Practical

* Search for "gcp console"

Create a gcp account

search for Vertex AI

choose Model Garden on the left bar

* you can checkout everything

Vertex AI Hands-on

→ Enable all recommended APIS

* Using their notebook
  → Colab enterprise
  → Workbench

→ Create a new workbench

select all the machine configs

Or

→ Create a colab entr— notebook which is collab like interface.

→ connect the notebook

can do all the coding

→ !pip ins— --upgrade --user google-cloud-aiplatform

→ import vertexai

* Authenticate your notebook environment (Colab only)

→ import sys

quit from ChatGPT

* If using from colab enterprice you dont need to authenticate it

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 Garden

→ 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

→ generation_config = GenerationConfig (← import this

temperature = 0.9,
top_p = 1.0,
top_k = 32,
candidate_count = 1
max_output_tokens = 8192,

)

response = model.generate_content(
"Why is the sky blue?",
generation_config = generation_config,
stream = True,
)

for res in res:
pr (res.text, end = " ")

→

* To make the chat remember all prev chats:

→ chat = model.start_chat()

prompt = " "

response = chat.send_message(prompt, stream = true)

for

Now it will remember that prompt :)

Now if we ask something related to that

→ prompt = "_____"

res_____ = chat.send_message(p_____)

for _____

→ * It will continue from same content from chat of prev

lets use vision model

→ multimodal_model = GenerativeModel("gemini-1.0-pro-vision")

Define helper functions to load and display images

you can get the code from chat gpt

and can do many stuff with videos and images

How to use vertex AI from our local machine VSC

→ first install gcp cli from google

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

Then do these commands

→ gcloud init

then do some required connection

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

choose region

and paste the
region in
.env too

main.py

```
import vertexai
import streamlit as st
import os
from vertexai.preview.generative_models
                    import (
    GenerationConfig
    GenerativeModel
)
from dotenv import load_dotenv

load_dotenv()

project_id = os.getenv("pro")
project_region = os.getenv("region")

vertexai.init(project_id, location
                    = pro)

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

this is how we do it

RAG using VertexAI

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

Finetuning with VertexAI

everything is done on cloud
you should just prepare your dataset

must be in .jsonl format

format:

{"input_text": "..."

"output_text": "..."

}

this is the data format

can convert using many converters in python

vin collab enterprise

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

If not enterprise only colab

→ import IPython
from google.cloud import aiplatform
from google.colab import auth as google_auth

google_auth.authenticate_user()

* login and pass and stuff

Select the project

→ import vertexai
PROJECT_ID = "_____"
vertexai.init(project = PROJ_____)

→ region = "_____"
project_id = "_____"

→ ! gcloud config set project {project_id}

→ #import some necessary libs

Chatgpt or at 19:51:00

→ df = pd.read_json("/——————.json1", lines = True)

→ df.shape
(744, 2)

→ model_display_name = "bbc-finetuned-model"
tuned_model = TextGenerationModel.from_pre-trained("text-bison@002")
* which model to finetune
after tuning name

tuned_model.tune_model(
training_data = df,
train_steps = 100, → epochs
tuning_job_location = "europe-west4",
tuned_model_location = "us-central1",
)

→ it will take time

* you will see a URL
open it
and you can see the pipelining and logs

* it will store everything in bucket of google Cloud

→ this bucket will give 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("prompt")

print(response.text)

→

Or if we want to load the model from the ENDPOINT

→ Tuned_model_name = tuned_model.endpoint.gca resource.deployed_model[0].model

tuned_model_1 = TextGenerationModel.get_tuned_model(tuned_model_name)

response = tuned_model_1.predict("")

print(response.text)

AWS Bedrock

LLMOps Platform

* in AWS console
→ Get started
* If using 1st time request for model Access
* In the Foundation model you can use custom model you can finetune in the UI itself

* to setup in VSC
* setup virtual environ
* Requirements.txt
  langchain
  langchain-community
  Streamlit
  Boto3 ——— with this we will make connection to AWS
  python-dotenv

→ Create IAM user
and in policy choose
AmazonBedrockFullAccess

→ In secret key option create an access key

→ CMD
create access key

→ Search for AWS cli and install

In terminal
→ AWS configure

Copy that access key and paste it here and the secret Access key too and give the region mention there in the .env file too

main.py

→ from langchain_community.llms.bedrock import Bedrock
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
import boto3
import streamlit as st
from dotenv import load_dotenv

load_dotenv()

aws_access_key_id = os.
aws_secret
region_name

# boto3 client
bedrock = boto3.client(
    service_name="bedrock-runtime",
    region_name
    aws_ac
    aws_secret
)
→ This with auth

model_id = "mistral.mis"

llm = Bedrock(
    model_id = model_id
    client = Bedrock
    model_kwargs = {"temp": 0.8})

```
def my_chatbot (language, user_text):
    prompt = PromptTemplate(
        input_variables = ["language", "user_text"],
        template = "You are a chatbot. You
                    are in {language}. \n\n
                    {user_text}"
    )

Bedrock_chain = LLMChain (llm = llm, prompt = prompt
response = Bedrock_chain({"language": language,
                          "user_text": user_text})
return response
```

this function will take lang and text and give respons

```
st.title("Bedrock Chat Demo")

language = st.sidebar.selectbox("language",
                                ["english", "spanish"
                                 "hindi"])

if language:
    user_text = st.sidebar.text_area(label=
    max_chars = 100) "what is your ques?",
```

if user_text:
    response = my_chatbot(language, user_text)
    st.write(response['text'])

* In terminal

→ streamlit run main.py

* we see our responses are very fast as we are using LLMops platforms

Can Implement RAG from this too

Check out the video at the