Implementing an Oracle OCI APEX-based Retrieval-Augmented Generation (RAG) application with GPT-2

CREATED BY JERRY BLAIR AND CHATGPT

OCTOBER 19, 2024

This document contains ChatGPT-generated content as a base augmented with content from the author (JERRY BLAIR). The content has been customized and reviewed by the author there are no plagiarism implications. Thie above statement is intended to provide transparency regarding the use of AI to avoid any ethical questions.

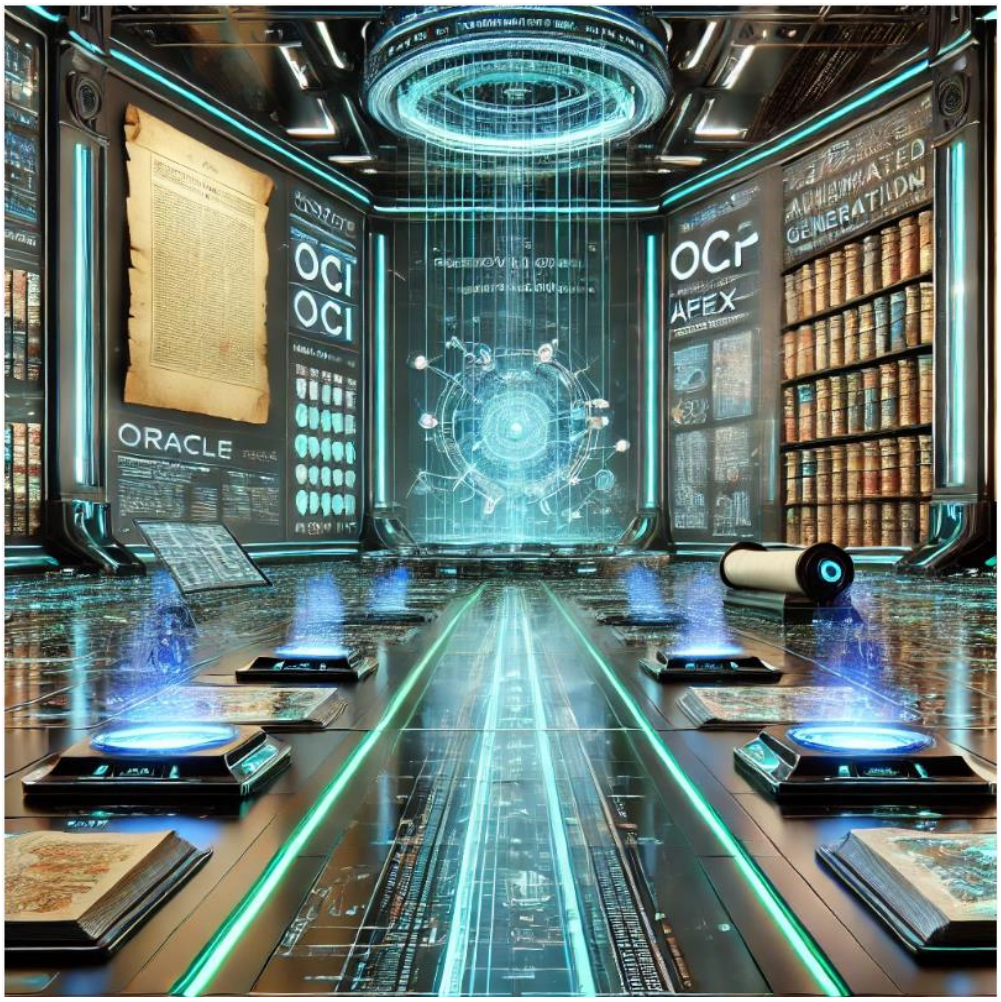

Table of Contents

Implementing an Oracle OCI APEX-based **Retrieval-Augmented Generation (RAG)** application with **GPT-2**

Here is a completely set of detailed instructions, starting from the beginning, to create an Oracle APEX RAG application using GPT-2. I will include explanations for why each component and step is required and what it does.

Step 1: Creating the Oracle 23c AI OLTP Database on OCI *LOE: 10 Minutes*

Purpose:

We need an OLTP (Online Transaction Processing) database as the backbone for storing data and handling transactional requests in the application. Oracle 23c AI provides enhanced support for AI and machine learning, making it ideal for integrating AI capabilities with GPT-2.

Detailed Steps:

Log in to OCI:

- Visit Oracle Cloud and log in using your credentials.
- Navigate to the **Oracle Database** section from the OCI dashboard.
- Select **Autonomous Database** from the options.

Create an Autonomous Transaction Processing (ATP) Database:

- Click **Create Autonomous Database**.
- Select **Autonomous Transaction Processing (OLTP)** from the database type options. OLTP is designed for high-volume, short transactions, which will handle all the incoming requests, data storage, and retrieval.
- Choose **Oracle 23c AI** as the database version. The AI capabilities in 23c will allow efficient integration with the GPT-2 model and RAG functionalities.
- **Database Name:** Choose an appropriate name (e.g., `RAG_Application_DB`).
- **Workload Type:** Select **Transaction Processing** for optimal performance.
- **Region & Availability Domain:** Select the region where you want the database to be hosted (choose the one closest to you for lower latency).
- **CPU & Storage Configuration:** Allocate CPUs and storage (start small if you're just testing, with 1-2 OCPUs and around 20 GB storage).
- **Auto Scaling:** Enable auto-scaling to allow the database to adjust resources based on usage needs automatically.

Database Password Setup:

  - Set up a **password** for the Admin account. This will be the main account used to access and configure the database.

Create Database:

- Click **Create** to launch the database. It will take a few minutes for the database to be provisioned.

Download the Wallet:

- Once the database is ready, download the **Database Wallet**. This wallet contains credentials to securely connect to the database from various tools (like SQL Developer, APEX, and others).
- Store the wallet securely on your local machine.

Why this step is important:

The OLTP database is the foundational data layer for the application. It will store transactional data, such as user inputs, model outputs, and interaction logs. Oracle 23c AI's capabilities allow for efficient integration with machine learning and AI tools like GPT-2.

Step 2: Create a Pingable OCI Compute Instance

Purpose:

The OCI Compute instance will be used to run your GPT-2 model and manage the necessary retrieval operations. This compute instance will interact with Object Storage, the database, and APEX to provide a seamless flow of data between the model and your Oracle APEX application.

A separate **OCI Compute Instance** is recommended instead of using the **Oracle Cloud Autonomous Database (ADB) code editor** for running and fine-tuning models like GPT-2. A detailed explanation for why a separate compute instance is provided at Appendix E.

Detailed Steps:

Navigate to the OCI Compute Section:

- From the OCI dashboard, click on Compute > Instances.
- Click Create Instance.

Instance Configuration:

- **Name the Instance:** Choose a meaningful name (e.g., `RAG_Model_Instance`).
- **Choose an Image:** Select **Oracle Linux 8** for the operating system. Oracle Linux is a stable and secure platform, and version 8 is optimized for modern workloads like machine learning.
- **Shape:** Select **VM.Standard.E3.Flex** or a similar shape. Flex shapes allow you to customize the number of OCPUs and memory. For GPT-2, start with around 2-4 OCPUs and 16 GB of memory (this will be sufficient for testing).

Configure Networking:

1. Choose a **Virtual Cloud Network (VCN)**. If you don't have one already, click **Create New VCN**.
    - **VCN CIDR Block:** Specify a private IP range (e.g., `10.0.0.0/16`).
    - **Subnet:** Create a subnet (e.g., `10.0.1.0/24`) that will host the compute instance.
    - This VCN isolates your instance from the public internet while allowing it to securely communicate with other OCI services.

Attach an Internet Gateway:

- Navigate to **Networking** > **VCNs** > **Create Internet Gateway**.
- This allows your instance to communicate with the outside world, which is necessary for downloading the GPT-2 model and libraries.
- **Add Route Rule:**
    - Destination CIDR: `0.0.0.0/0`
    - Target: **Internet Gateway**.

Configure Security Lists:

- The **Security List** defines the inbound/outbound traffic allowed for your instance.
- **Inbound Rules:**
    - Allow SSH:
        - **Source CIDR:** `0.0.0.0/0`
        - **Protocol:** TCP
        - **Port:** 22 (for SSH)
    - Allow ICMP for Ping:
        - **Protocol:** ICMP
        - **Source CIDR:** `0.0.0.0/0`
- **Outbound Rules:**
    - Allow all outbound traffic (e.g., HTTP requests for library downloads).

SSH Key Pair:

- Upload your **public SSH key** (typically `id_rsa.pub`) during the instance creation process.
- The corresponding private key (`id_rsa`) will be used later to connect via SSH.

Create the Instance:

- Click **Create** to spin up the compute instance.
- It will take a few minutes for the instance to be provisioned.

Test Ping and SSH Access:

- Once the instance is running, verify connectivity by pinging its public IP:

```
bash
Copy code
```

```
ping <instance-public-ip>
```

- Use SSH to connect:

```
bash
Copy code
ssh -i path_to_private_key opc@<instance-public-ip>
```

Why SSH is required:

Secure Shell (SSH) provides encrypted access to the server's command line, allowing you to install the necessary tools (Python, TensorFlow, etc.) to run the GPT-2 model.

Why this step is important:

A compute instance is necessary for running the GPT-2 model, which is computationally intensive. It needs to be externally accessible (SSH) for easy management but secured within a Virtual Cloud Network (VCN) for privacy and security.

Step 3: Install the GPT-2 Model on the OCI Compute Instance

Purpose:

GPT-2 will be the model used to generate text outputs based on the input data. We need to set up Python and TensorFlow, and then load GPT-2 for initial testing.

Detailed Steps:

1. **Install Python and Dependencies:**
   - SSH into the compute instance and update the package manager:

```
bash
Copy code
sudo yum update -y
```

   - Install Python 3 and pip:

```
bash
Copy code
sudo yum install python3
sudo yum install python3-pip
```

2. **Install GPT-2 and TensorFlow:**
   - Install TensorFlow and transformers (Hugging Face) libraries:

```
bash
Copy code
```

```
pip3 install tensorflow transformers
```

- **Why TensorFlow?** TensorFlow is a deep learning framework that allows you to efficiently run the GPT-2 model.
- **Why transformers library?** The `transformers` library from Hugging Face provides an easy-to-use interface for loading and using pre-trained models like GPT-2.

3. **Load GPT-2 Model:**
   - Create a Python script (`gpt2_model.py`) to load the GPT-2 model:

```
python
Copy code
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load the GPT-2 model and tokenizer
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Example generation
input_text = "In the beginning God created"
inputs = tokenizer(input_text, return_tensors="pt")
outputs = model.generate(inputs['input_ids'], max_length=100)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

   - Run the script:

```
bash
Copy code
python3 gpt2_model.py
```

   - This will output a text sequence generated by GPT-2.

Why this step is important:

Loading and testing the GPT-2 model ensures that the core component of the RAG system is ready. GPT-2 will handle all text generation, making it essential to set up properly.

Step 4: Adding Retrieval Functionality for RAG (Retrieval-Augmented Generation)

Purpose:

The retrieval functionality is critical for RAG, as it enhances the GPT-2 model by allowing it to retrieve relevant documents or texts from a database or file storage system, such as Oracle Object Storage. The GPT-2 model will use this retrieved information as additional context for generating more accurate and contextually relevant responses.

We will set up OCI Object Storage for storing documents, create a connection between the storage and the compute instance, and ensure that GPT-2 can retrieve the relevant documents when needed.

Step 4.1: Set Up OCI Object Storage for Document Storage

1. **Navigate to Object Storage:**
   - In the OCI Console, go to **Storage** > **Object Storage**.
   - Click **Create Bucket** to create a new storage bucket.
2. **Configure the Object Storage Bucket:**
   - **Bucket Name:** Choose a meaningful name (e.g., `RAGBibleDocuments`).
   - **Storage Tier:** Select **Standard** for frequent access to the data.
   - **Encryption:** Use the default encryption settings (Object Storage data is encrypted by default).
   - Click **Create** to create the bucket.
3. **Upload Documents to Object Storage:**
   - After creating the bucket, upload your documents (e.g., Bible text, interpretations, historical commentaries) as individual files. You can do this manually by clicking **Upload Object** in the bucket, or automate the process via the OCI CLI.
   - **Example Document Names:**
     - `genesis.txt`
     - `exodus.txt`
     - `commentary_genesis.txt`
     - `historical_context_exodus.txt`
4. **Enable Pre-Authenticated Request (PAR):**
   - If you prefer not to use credentials every time you access the Object Storage, you can create a **Pre-Authenticated Request (PAR)**.
     - Go to **Object Storage** > **Bucket** > **Pre-Authenticated Requests**.
     - Click **Create PAR**.
     - Set the **Object Name Prefix** as `/` to cover all objects.
     - Set expiration (based on how long you need the access).
     - The PAR URL will be used for accessing files in Object Storage without providing credentials every time.

Explanation:

- **Why Object Storage?** Object Storage in OCI provides secure, durable, and scalable storage for unstructured data (like text files). This is where you will store all the documents you want GPT-2 to access and retrieve for enhanced generation.
- **Why Pre-Authenticated Request (PAR)?** PAR allows secure access to the bucket without handling authentication programmatically, simplifying access from your application.

Step 4.2: Create a Python Script to Retrieve Documents from Object Storage

1. **Install OCI SDK for Python on the Compute Instance:**
   - SSH into your compute instance and install the OCI SDK:

```
bash
Copy code
```

```
pip3 install oci
```

2. **Set Up OCI Configuration File:**
   - If you are using credentials (instead of a PAR), you need to set up an OCI config file to authenticate your compute instance.
   - In your home directory (`~`), create an `.oci` directory:

```
bash
Copy code
mkdir ~/.oci
vi ~/.oci/config
```

   - Add the following configuration details to the `config` file:

```
ini
Copy code
[DEFAULT]
user=ocid1.user.oc1..your_user_ocid
fingerprint=your_fingerprint
key_file=/path/to/your/private_key.pem
tenancy=ocid1.tenancy.oc1..your_tenancy_ocid
region=us-ashburn-1
```

   - You can find these values in the OCI console (for user, fingerprint, tenancy) under **Identity** > **Users**.

3. **Python Script to Access Object Storage:**
   - Now, write a Python script (`retrieve_document.py`) that retrieves documents from your Object Storage bucket and feeds the content into the GPT-2 model.

```
python
Copy code
import oci
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# OCI Object Storage configuration
config = oci.config.from_file("~/.oci/config")  # or use PAR for
easier access
object_storage_client =
oci.object_storage.ObjectStorageClient(config)
namespace = "your_namespace"
bucket_name = "RAGBibleDocuments"

# Function to retrieve a document from Object Storage
def get_file_content(file_name):
    obj = object_storage_client.get_object(namespace,
bucket_name, file_name)
    return obj.data.content.decode('utf-8')

# Load the GPT-2 model
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Retrieve a specific document (e.g., Genesis)
```

```
document_content = get_file_content("genesis.txt")

# Tokenize and generate text using GPT-2
inputs = tokenizer(document_content, return_tensors="pt")
outputs = model.generate(inputs['input_ids'], max_length=150)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

4. **Run the Python Script:**
   - Run the script on your compute instance:

```
bash
Copy code
python3 retrieve_document.py
```

5. **Check Output:**
   - The script will retrieve the text from `genesis.txt` stored in Object Storage, pass it through GPT-2, and output a generated response. The GPT-2 model now has access to real-world data stored in Object Storage.

Explanation:

- **Why retrieve documents from Object Storage?** Since the GPT-2 model doesn't have real-world knowledge post-2019, retrieving contextually relevant documents (like biblical texts and interpretations) helps enhance its generation capabilities. Object Storage serves as a large-scale, external knowledge source for the model.
- **Why OCI SDK?** The SDK allows secure, programmatic interaction with Oracle services (like Object Storage) from your compute instance.

Step 4.3: Connect Retrieval to Oracle APEX

1. **Create a REST Data Source in APEX:**
   - In Oracle APEX, navigate to **Shared Components** > **REST Data Sources**.
   - Click **Create** and choose the option for **Oracle Cloud Infrastructure (OCI)**.
   - Enter the base URL for your Object Storage's REST API, or use the PAR URL for easy access.
   - Test the connection to ensure it's set up correctly.
2. **Create a Page for Retrieval in APEX:**
   - Create a new **Interactive Report** page in your APEX application.
   - Use the **REST Data Source** you created to display a list of files (like Bible texts) stored in Object Storage.
   - Customize the report to allow users to select a document for retrieval.
3. **Create a Button to Retrieve and Generate Text:**
   - Add a button to the interactive report that calls a **PL/SQL dynamic action**.
   - The PL/SQL code can execute a server-side process that makes a REST call to your Python script running on the compute instance to retrieve and process the document.
4. **Example PL/SQL Code to Trigger Python Script:**
   - In the dynamic action, write the following PL/SQL code to make a REST call:

```
plsql
Copy code
DECLARE
  l_http_request  UTL_HTTP.req;
  l_http_response UTL_HTTP.resp;
  l_url           VARCHAR2(4000) := 'http://<compute-instance-
ip>/retrieve_document.py';
  l_result        CLOB;
BEGIN
  l_http_request := UTL_HTTP.begin_request(l_url);
  l_http_response := UTL_HTTP.get_response(l_http_request);

  UTL_HTTP.read_text(l_http_response, l_result);
  UTL_HTTP.end_response(l_http_response);

  -- Output the result
  htp.p(l_result);
END;
```

5. **Test in APEX:**
   - Run your APEX application and navigate to the new page.
   - Choose a document from the list and click the button to retrieve and process the document using GPT-2.

Explanation:

- **Why REST Data Source in APEX?** The REST Data Source allows APEX to interact with external systems (in this case, Object Storage) to retrieve data for the application.
- **Why Dynamic Actions in APEX?** Dynamic Actions are used to trigger server-side processes (like retrieving and processing documents) based on user interaction, enhancing the application's interactivity and functionality.

Summary of Step 4:

You have now successfully set up the retrieval functionality in your Oracle APEX RAG application. The GPT-2 model can retrieve relevant documents from Oracle Object Storage and use them as context for generating enhanced text. The integration between APEX, Object Storage, and the compute instance running GPT-2 provides the full flow of data retrieval and processing.

Step 5: Fine-Tuning GPT-2 with Biblical Content, History, and Interpretations

Purpose:

Fine-tuning GPT-2 involves taking the pre-trained model and adjusting it with domain-specific data (in this case, the Bible, historical interpretations, and related documents). Fine-tuning helps the model better understand the context of your application and generate more accurate

responses. This step will provide detailed instructions on how to fine-tune GPT-2 using biblical content stored in OCI Object Storage.

Step 5.1: Prepare Data for Fine-Tuning

1. **Format the Data:**
   - Fine-tuning GPT-2 requires the data to be in a plain text format. Ensure that all biblical content and related documents are saved as `.txt` files.
   - Each file should contain a structured block of text, such as:
     - `genesis.txt` (contains the entire Book of Genesis).
     - `interpretation_genesis.txt` (contains a verse-by-verse interpretation).
     - Other documents should follow a similar pattern for easy access and organization.
2. **Upload the Data to OCI Object Storage:**
   - Ensure that all files to be used for fine-tuning are uploaded to your Object Storage bucket (created in Step 4). This makes them accessible for retrieval during the fine-tuning process.
   - Verify that the filenames are descriptive and easy to reference.

Explanation:

- **Why plain text format?** GPT-2 processes text data in a sequence. By providing the documents in `.txt` format, you ensure that the model can read, tokenize, and fine-tune using the content efficiently.

Step 5.2: Set Up Python Environment for Fine-Tuning

1. **Install Required Packages:**
   - SSH into your compute instance and ensure all required packages for fine-tuning are installed. If you haven't already, install the following:

     ```
     bash
     Copy code
     pip3 install transformers datasets torch
     ```

   - **transformers**: Provides the GPT-2 model and tokenizer functionality.
   - **datasets**: Helps in managing and loading large datasets efficiently for fine-tuning.
   - **torch**: Core deep learning framework for running and fine-tuning the GPT-2 model.
2. **Download the Fine-Tuning Script:**
   - Create a new Python script (`fine_tune_gpt2.py`) that will handle the fine-tuning process.

Explanation:

- **Why `datasets` and `torch` libraries?** These libraries optimize the process of handling large datasets and make model fine-tuning computationally efficient.

Step 5.3: Implement the Fine-Tuning Script

1. **Fine-Tuning Script:**
   - Below is a full Python script to fine-tune GPT-2 using your biblical content and interpretations stored in OCI Object Storage.

```
python
Copy code
import oci
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer,
TrainingArguments, TextDataset, DataCollatorForLanguageModeling

# Load GPT-2 model and tokenizer
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# OCI Object Storage configuration
config = oci.config.from_file("~/.oci/config")  # Ensure this
path matches your configuration
object_storage_client =
oci.object_storage.ObjectStorageClient(config)
namespace = "your_namespace"
bucket_name = "RAGBibleDocuments"

# Function to retrieve a file from Object Storage and save it
locally for fine-tuning
def retrieve_and_save_file(file_name, save_path):
    obj = object_storage_client.get_object(namespace,
bucket_name, file_name)
    with open(save_path, "w") as file:
        file.write(obj.data.content.decode('utf-8'))

# Retrieve and prepare text files for fine-tuning
retrieve_and_save_file("genesis.txt", "genesis.txt")
retrieve_and_save_file("interpretation_genesis.txt",
"interpretation_genesis.txt")

# Load dataset for fine-tuning
def load_dataset(file_path, tokenizer, block_size=128):
    return TextDataset(
        tokenizer=tokenizer,
        file_path=file_path,
        block_size=block_size
    )

# Prepare data collator (this groups text into blocks for model
training)
```

```
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False,  # GPT-2 doesn't use masked language modeling
)

# Load datasets
train_dataset = load_dataset("genesis.txt", tokenizer)
interpretation_dataset =
load_dataset("interpretation_genesis.txt", tokenizer)

# Training Arguments (adjust epochs, batch size, etc. based on
your instance capabilities)
training_args = TrainingArguments(
    output_dir="./fine_tuned_model",
    overwrite_output_dir=True,
    num_train_epochs=3,            # You can increase the number
of epochs for better fine-tuning
    per_device_train_batch_size=4,  # Adjust batch size based on
instance memory
    save_steps=10_000,
    save_total_limit=2,
)

# Create Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=train_dataset,
)

# Start fine-tuning
print("Fine-tuning GPT-2 model...")
trainer.train()

# Save the fine-tuned model
trainer.save_model("./fine_tuned_gpt2_bible")
tokenizer.save_pretrained("./fine_tuned_gpt2_bible")

print("Model fine-tuning complete!")
```

2. **Explanation of Key Elements in the Script:**
   - **retrieve_and_save_file:** This function retrieves a file from OCI Object Storage and saves it locally for fine-tuning.
   - **load_dataset:** This function prepares the text data (Bible and interpretations) for model training by tokenizing and organizing it into sequences.
   - **TrainingArguments:** This object defines key hyperparameters, such as the number of epochs, batch size, and where to save the fine-tuned model.
   - **Trainer:** This is the core component of the `transformers` library that handles the fine-tuning process.
3. **Run the Fine-Tuning Script:**
   - To fine-tune GPT-2 with your biblical data, run the script on your compute instance:

```
bash
```

```
Copy code
python3 fine_tune_gpt2.py
```

4. **Monitor Training Progress:**
    - The training process may take several minutes to hours depending on your compute instance and the size of your dataset.
    - The script will periodically save checkpoints (every 10,000 steps), so you can monitor the progress.

Explanation:

- **Why fine-tune GPT-2?** Fine-tuning the model allows you to specialize GPT-2 for your specific use case (biblical texts and interpretations). This process enhances the model's understanding and ability to generate relevant responses based on the specific knowledge you've provided.
- **Why use `Trainer` class?** The `Trainer` class simplifies the training process, handling most of the complexity related to batching, loss computation, and optimization.

Step 5.4: Store and Deploy the Fine-Tuned Model

1. **Save the Fine-Tuned Model in OCI Object Storage:**
    - After fine-tuning is complete, save the model in Object Storage for future use:

```
python
Copy code
fine_tuned_model_path = "./fine_tuned_gpt2_bible"
oci.util.upload_file(
    object_storage_client,
    namespace,
    bucket_name,
    "fine_tuned_gpt2_bible.zip",
    fine_tuned_model_path
)
```

2. **Deploy the Fine-Tuned Model:**
    - The fine-tuned model can now be reloaded and used for text generation in the same way as the base GPT-2 model. In your application or scripts, simply load the fine-tuned model:

```
python
Copy code
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load the fine-tuned model
model =
GPT2LMHeadModel.from_pretrained("./fine_tuned_gpt2_bible")
tokenizer =
GPT2Tokenizer.from_pretrained("./fine_tuned_gpt2_bible")

# Example of generating text with fine-tuned model
```

```
input_text = "In the beginning, God created the heaven and the
earth"
inputs = tokenizer(input_text, return_tensors="pt")
outputs = model.generate(inputs['input_ids'], max_length=150)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Explanation:

- **Why save and reload the fine-tuned model?** Once the model is fine-tuned, it can be deployed for use in applications. Saving the model ensures it can be reused and distributed without needing to re-train it from scratch.

Step 5 Summary:

By completing this step, you have fine-tuned GPT-2 using biblical content and related interpretations, preparing it to generate text based on domain-specific knowledge. The fine-tuned model can now be stored in OCI Object Storage for use in your Oracle APEX application.

Step 6: Creating an Oracle APEX 24.1.1 RAG Application

Purpose:

The goal of this step is to integrate the fine-tuned GPT-2 model into your Oracle APEX application, enabling it to retrieve documents from Oracle Object Storage, generate text based on those documents, and continuously fine-tune the model with additional data. We will create a fully functional APEX app that provides an interface for uploading files, running the GPT-2 model, and interacting with the retrieval-augmented generation (RAG) functionality.

Step 6.1: Create a New Oracle APEX Application

1. **Log into Oracle APEX:**
   - Use the browser to access the APEX instance running on your Oracle 23c AI database.
   - Log in as an APEX admin or workspace developer.
2. **Create a New APEX Application:**
   - From the APEX dashboard, click **App Builder** > **Create**.
   - Select **New Application** and give the application a name (e.g., `BibleRAGApp`).
   - Choose the **Theme** (Universal Theme is recommended).
   - Select any default pages you might need, such as a **Home** page, but we'll primarily focus on custom functionality.
3. **Create an Interactive Report for Document Management:**
   - **Interactive Report**: Go to **Create Page** > **Report** > **Interactive Report**.
   - Set the page name as **Document Management**.
   - This page will display the list of documents stored in OCI Object Storage, allowing users to interact with the data.

Explanation:

- **Why an Interactive Report?** Interactive reports in APEX allow users to view, search, and interact with data in a tabular format. In this case, it will allow them to see all the documents uploaded to Object Storage and select files to be used for text generation.

Step 6.2: Integrate OCI Object Storage with APEX

1. **Create a REST Data Source to Access Object Storage:**
   - Go to **Shared Components** > **REST Data Sources**.
   - Click **Create** > Select **Oracle Cloud Infrastructure (OCI)**.
   - Enter the base URL for your Object Storage bucket. This is either the **PAR URL** (if you set up Pre-Authenticated Requests) or the REST API endpoint if you are using an API key/OCI credentials.
   - **Example Base URL:**
     `https://<namespace>.objectstorage.<region>.oraclecloud.com/n/<bucketname>/`
   - Enter any required authentication information (e.g., PAR URL or API keys).
2. **Test the REST Data Source:**
   - After setting up the data source, click **Test** to ensure APEX can successfully retrieve the list of files from the bucket.
3. **Bind REST Data Source to the Interactive Report:**
   - Open the **Document Management** page and link it to the REST Data Source.
   - Configure the columns to show the file name, file size, and other relevant metadata (e.g., `name`, `size`).
4. **Add a Button for File Retrieval:**
   - On the **Document Management** page, add a **Button** that will trigger the retrieval of the selected file from Object Storage and pass it to the GPT-2 model for text generation.
   - **Example Button Name: Generate Text**.
5. **Create a PL/SQL Dynamic Action to Trigger Text Generation:**
   - Create a **Dynamic Action** triggered by the **Generate Text** button.
   - In the **PL/SQL Code** section of the dynamic action, add the following:

```
plsql
Copy code
DECLARE
  l_http_request  UTL_HTTP.req;
  l_http_response UTL_HTTP.resp;
  l_url           VARCHAR2(4000);
  l_result        CLOB;
BEGIN
  -- Prepare the URL to call the GPT-2 script on the compute
instance
  l_url := 'http://<compute-instance-
ip>:<port>/run_gpt2.py?file=' || :P1_FILE_NAME;  -- Replace
:P1_FILE_NAME with the name of the file selected in the report.

  -- Make HTTP request to retrieve document and run GPT-2
  l_http_request := UTL_HTTP.begin_request(l_url);
```

```
    l_http_response := UTL_HTTP.get_response(l_http_request);
    UTL_HTTP.read_text(l_http_response, l_result);
    UTL_HTTP.end_response(l_http_response);

    -- Output the result on a new page or display in a region
    :P1_RESULT := l_result;
END;
```

- **Explanation:**
  - The PL/SQL code sends an HTTP request to your Python script running on the compute instance, passing the selected file name as a parameter. The Python script will retrieve the file from Object Storage, run the GPT-2 model, and return the generated text to APEX.

6. **Create a Text Area to Display the Generated Text:**
   - Create a **Text Area** region on the same page or a new page to display the result of the GPT-2 generation.
   - Set the **Source** of this text area to the variable `:P1_RESULT`, which will hold the generated text returned by the PL/SQL block.

Explanation:

- **Why REST Data Source?** The REST Data Source allows APEX to dynamically interact with OCI services (like Object Storage) without needing hard-coded file paths or manual uploads.
- **Why PL/SQL Dynamic Action?** PL/SQL is used to trigger the external process (calling the Python GPT-2 script) while keeping the data flow within APEX's framework.

Step 6.3: Create a File Upload Page to Continuously Add Data

1. **Create a File Upload Page in APEX:**
   - Go to **Create Page** > **File Upload**.
   - Name the page **Upload New Documents** and ensure it stores the uploaded files in your OCI Object Storage bucket.
2. **Link the File Upload Page to Object Storage:**
   - Under **Shared Components**, create a **File Browser** item.
   - Set the **File Browser** to use the **REST Data Source** for your Object Storage bucket.
   - This will allow users to upload files directly to the Object Storage bucket from APEX.
3. **Process to Fine-Tune GPT-2 Automatically:**
   - When a new file is uploaded, trigger a **PL/SQL Process** to call your fine-tuning script on the compute instance, ensuring the new data is added to the model.

```
plsql
Copy code
DECLARE
  l_http_request  UTL_HTTP.req;
  l_http_response UTL_HTTP.resp;
  l_url           VARCHAR2(4000);
  l_result        CLOB;
BEGIN
```

```
  -- Prepare the URL to trigger fine-tuning process
  l_url := 'http://<compute-instance-
ip>:<port>/fine_tune_gpt2.py?file=' || :P2_FILE_NAME;  -- Replace
:P2_FILE_NAME with the name of the uploaded file.

  -- Make HTTP request to trigger fine-tuning
  l_http_request := UTL_HTTP.begin_request(l_url);
  l_http_response := UTL_HTTP.get_response(l_http_request);
  UTL_HTTP.read_text(l_http_response, l_result);
  UTL_HTTP.end_response(l_http_response);

  -- Output the result (confirmation that fine-tuning has started)
  htp.p(l_result);
END;
```

4. **Display Fine-Tuning Status:**
    - Create a **region** on the upload page that shows the status of fine-tuning. This could either be a **confirmation message** or a **status bar** that updates as the new file is incorporated into the model.

Explanation:

- **Why a File Upload Page?** This allows continuous addition of new biblical content or interpretations to the model. By automating the fine-tuning process, the model remains up-to-date with the latest information.
- **Why trigger fine-tuning dynamically?** Automating this process ensures the model continues learning and improving without requiring manual intervention.

Step 6.4: Deploy the Application

1. **Deploy APEX Application:**
    - Once the application is built, deploy it within your Oracle APEX instance.
    - You can set appropriate user roles (e.g., admin roles for those who can upload data and general roles for those who just generate text).
2. **Monitor Application Usage:**
    - Track usage metrics via APEX's built-in reporting features.
    - Ensure that the GPT-2 model is fine-tuned periodically with new content.
3. **Test the Full Workflow:**
    - Test the application end-to-end by uploading new biblical content, retrieving it, and generating text using the fine-tuned GPT-2 model. Ensure that the generated text reflects the most recently uploaded documents.

Explanation:

- **Why deploy on APEX?** Oracle APEX provides a secure, scalable platform to build and deploy web applications quickly, making it ideal for integrating your RAG model and fine-tuned GPT-2.

- **Why monitor application usage?** Monitoring helps you understand how users interact with the system and how well the model is performing, allowing for adjustments and improvements.

Final Summary of Step 6:

In this step, you have created a fully functional Oracle APEX application that integrates with your fine-tuned GPT-2 model. The application allows users to retrieve documents from OCI Object Storage, generate text using the model, upload new documents, and continuously fine-tune the model. This workflow enables the model to remain dynamic, improving as more data is added.

Appendix A: OCI Configurations for Database, APEX, Users, & Buckets

COMPONENT	ATTRIBUTE	ATTRIBUTE VALUE
OCI Instance		
	Tenancy	jerryblairconsulting
	URL	https://cloud.oracle.com/?region=us-chicago-1
	Console Username	jerry.blair.consulting@gmail.com
	Password	Jims#10
OLTP Database		
	Name	Live-Labs-OlTP-Database-Instance
	OCID	ocid1.autonomousdatabase.oc1.us-chicago-1.anxxeljsnmapb7qaunsdhjmamgh7ag7kboszbbjrgktjdt7mshbmv4tfzynq
	Compartment	jerryblairconsulting (root)/LiveLabs
	DB Version	23ai
	Admin Username	Admin
	Admin Password	Jims#10
Group		
	Domain Name	LiveLabsDomain
	Group Name	RAGAPPOSGroup
	OCID	ocid1.group.oc1..aaaaaaaahmhrikyaidfqljbbyz2d5tu6y7zkrqk7od2kiwty4ub3ahomxe6q
Group Policy		
	Group Policy Name	RAGAPPOSGroupPolicy
	OCID	ocid1.policy.oc1..aaaaaaaawzovk2qolvrseecd2ar7i5gq2akts3medya4p6sfzunqzul6jb5a
	Compartment	jerryblairconsulting (root)/LiveLabs
	Policy Statements	allow group RAGAPPOSGroup to manage object-family in compartment LiveLabs
User		
	User Group	RAGAPPOSGroup
	User Name	RAGAPPGroupUser
	OCID	ocid1.user.oc1..aaaaaaaazoiuaw3pd7ktcg6rhwzd5htsii2tfjstyqqjnzsajlrrwlxotjma
	User Groups	RAGAPPGroupUser Domain Administrators
	First and Last Name	Jerry Blair
	Recovery Email	436jtb@gmail.com

COMPONENT	ATTRIBUTE	ATTRIBUTE VALUE
	Private Key	"-----BEGIN PRIVATE KEY----- MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCQcNUYqUAh6SHE jzj5wkL0Tj5wR9RgavpMeRmbChNy817qHPGDQPkURPRO84wjPTKIsLy0zaRhWASz S9SWsjH1oR0wysdXG9J4nCnkQoEKSpQT/+EBLhvwaTvqKGR8HPwuey7UcyHkrp4r APSMNusd+D1ek6fxq8tKJ1qkhIY2/n9rz9ZqVz1T+a7onANGn42lGVvNbJ7Ryv6D gKvkdFg9Oph0OWiXviadKfZqd082L9xANWDK6LMkxkHOK11op2oCCg/HMHs6KAkR wILOkiKbW2mn5iJ5jo6VKjzL2j9rtDAjhqtKhfYPwePfAxJFHR7QkjEhnu08jC8m 7Dooo679AgMBAAECggEAGFa+/b0VVrF9VG0K3YEnvZmKqWpdJdMY8rZsQ2UHAnVy frIaWzLhVdJJtZ0R9eUrxP89CR1mz9VgU2MZgzwrMXfw3oYo4WuJg3PTFkjDjDul aJpTLJr7r77Xzvz387cBooZ0zW1KHonzEo2CZ7cgVsg79Rod8Fnxs0De4/6FzDHE 8iMdIIS1rLO6vv6OiZQ/XRMuXUNq+CF2pM9oYOOiEBNzAmj4zNn722MRZlVvaf5Y 09IRjmq9n2BHyBzcQO41jvc+38wrvSFdQdGIHFs0HX3NQ3Q7EGyVcdrYsGdZEnyU oHjpD2DLFVIuodAKY+0hWBNG/QFynBNADi4pCVv3eQKBgQDAuW4lx3NTNjofvkfU aETWAV5nqwAJQlVizLD0ZgaE90Fro4ksHw6MJ450WHPKggTAmmcu9hlyBGzNZtQb AJCOv9sqv1GJfsM0zw1SIBAo9rnUSkFEnz89gx8KtLuS32rpbaEHT99WS+0/TY85 Lg2kYoraXNEWLlGY3DdOwk1yxQKBgQC/3ST6It+w5P8Gm+rYYXf02U7EQRNEF+hi r9adXfxx/iaV9tz1z/DNcco/Qe2YLY74/MGUt+ybbuz3j86LF2yNoYWgXyd++Gw7 g0tm7d4GKxAnWItQdXvhaXpYM2l+PaaYyhxKVcT8r4jo4mkC/1JxjWJ+RPiAymQm DFfgDsYu2QKBgQCdzbeRG5Ukh3xkTYvHMnFawdgpDm2DJ1t/AJfMJlYkPcjFeCxz giKJREJypZJ6OKfnhQYD/+Kp51GnhhEa4wV6vUGZ6Pm8mZ0A+qCvpC1XPV6/ouV6 aOKBXYHnZEFKa8HAzNFTiWUX5iem87zYVws4lK+ZSPKzNM3/WiG3TYXY+QKBgQC+ iaP7NT7oscICkjz7fnBenKB4jhfP+94wg2ejCH93xWoUbACrrUH7BRJRF+wYwMXH x1uHkdBL3DFA+XOo9i47yTinN1hF4/e4cn8jTP69KW71ZB2WXdYU2WdCyGvvjtIv 0ieeOudR+hEBFjgrXdxGIJ67v0r6vAQRNRjfzkxKoQKBgEnddgZeSq1pWdR/dnfk

COMPONENT	ATTRIBUTE	ATTRIBUTE VALUE
		25aNoyYNf5zgCKMP5Xkg5uyMXUkNI0WRvfEFmkCByjFKUn5D/ wCDK9QZXl7dWpU5 5phEiFqMcZZWfp0LaNmFDk3CvSbGP01O4aPCjoGC9AE3dZjdy uJEvILeAv92d7EN 0z0b2SmSgynOaypm0Ty7u0Ev -----END PRIVATE KEY-----
	Public Key	"-----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkHDV GKlAIekhxl84+cJC 9E4+cEfUYGr6THkZmwoTcvNe6hzxg0D5FET0TvOMIz0yiLC8tM 2kYVgEs0vUlrIx 9aEdMMrHVxvSeJwp5EKBCkqUE//hAS4b8Gk76ihkfBz8Lnsu1H Mh5K6eKwD0jDbr Hfg9XpOn8avLSidapISGNv5/a8/Walc9U/mu6JwDRp+NpRlbzW ye0cr+g4Cr5HRY PTqYdDlol74mnSn2andPNi/cQDVgyuizJMZBzitdaKdqAgoPxzB7 OigJEcCCzpIi m1tpp+YieY6OlSo8y9o/a7QwI4arSoX2D8Hj3wMSRR0e0JIxIZ7t PIwvJuw6KKOu /QIDAQAB -----END PUBLIC KEY-----
	Fingerprint	76:06:0b:2f:5e:7c:74:b5:e8:3f:2c:45:dd:98:5b:7c
	Tenancy	ocid1.tenancy.oc1..aaaaaaaa6qhnofktohk6c3qnpuxb72nxtq4yi meor5sna4wfjrxcq6uborea
	Region	us-chicago-1
Bucket	Bucket Name	RAGAPPWebSiteBucket
	Namespace	axtb5v7whbot
	Compartment	LiveLabs
	OCID	ocid1.bucket.oc1.us-chicago- 1.aaaaaaaa2krvuioq5dx627torxlsu3fzbtz3q2yujfi6ltg2ziqxlooqy 3za
	PAR Name	
	PAR URL (old)	https://objectstorage.us-chicago-1.oraclecloud.com/p/nG8e- 2S-G07e5RQHYi- myPA_ZB7KbF5nd0Es5Sqjvmg8Z3sdOmqSgiLfYEwuE7Kw/n/axt b5v7whbot/b/RAGAPPWebSiteBucket/o/
	PAR URL (new)	https://axtb5v7whbot.objectstorage.us-chicago- 1.oci.customer-oci.com/p/nG8e-2S-G07e5RQHYi- myPA_ZB7KbF5nd0Es5Sqjvmg8Z3sdOmqSgiLfYEwuE7Kw/n/axt b5v7whbot/b/RAGAPPWebSiteBucket/o/
APEX Instance		

COMPONENT	ATTRIBUTE	ATTRIBUTE VALUE
	Database Name	LiveLabsOLTPDatabase
	Database Type	Transaction Processing
	Compartment	jerryblairconsulting (root)/LiveLabs
	APEX Version	24.1.1
	ORDS Version	24.3.0.262.0924
	OCID	ocid1.autonomousdatabase.oc1.us-chicago-1.anxxeljsnmapb7qaunsdhjmamgh7ag7kboszbbjrgktjdt7mshbmv4tfzynq
	Workspace	APEX
	Default Schema	WKSP_APEX
	Username	APEX
	Password	Jms#10
APEX APPLICATION	Application Id	2024
	Application Name	Exploring the Non-Technical through Technology
	Workspace	APEX
	Username	APEX
	Password	Jms#10
OCI Compute Instance	Instance Name	GPT2ComputeInstance
	Availability Domain	AD-1
	OCID	ocid1.instance.oc1.us-chicago-1.anxxeljtnmapb7qciyvkdxder3y7cwmrnsjyuk23dyhpp7zrws5ugu35r73a
	Virtual Cloud Network	GPT2ComputeInstance-vcn
	Public IP Address	164.152.22.23
	Public IPv4 address	164.152.22.23
	Private IPv4 Address	10.0.0.101
	SSH Private Key	-----BEGIN RSA PRIVATE KEY----- MIIEogIBAAKCAQEAsENbc5JabCmIUSTJJl9IubICcqHOU+4IqDAxKG9a6NUvZ92X D7Wpd7g1kdPfvBT2AfaTq13lC9DD48W8FP1L4MzXH9H8B+An8ToVcWRAFTg1PkaP B8eEUcdNQUXdMsFt6JeDq827hpYZckqaqsHJmfzfcC8hTwP6FzrCMO6+zVrtTbO7 0PdkgF0vglEPH3D68cvWy3tjim+X1lN2SpGQWPVGkWgS9WH9GvNANrShjbvXrvLa

COMPONENT	ATTRIBUTE	ATTRIBUTE VALUE
		TbMy7wMS+8mZ2E1MAM2MltlkkA+IZPIgZQzv1Jct5uEgJRjzeShZZ2K0Vx+7s78W h5zCxSb1glf2dUT75SY2mrTaUTenUKcwI7tjmwIDAQABAoIBAA7E/h8/hrMkgccm 2XINLudceqcHVuYifng1XZEhT5FN98RwxW0sEDC3b0myYr98VCbS9gm4eu5UBm4K DhoGJXsHPwlsC33PcxlcABr5xwLgLFn1CJwixDwkAp2zEHdlzONd+ssFGK8M2cIB bVdcmJt+HGzmjhh0YVrb8hSulK9ZSQePSUa1fxLHTbcRcomGSas+MVt5fWisuBrU Qy/KSptURlh3qIazyv1mH+HlQpd+YVbSNlqlLv9l87NbjL6mQjYaHx6vprX/gJam jvOld7sAGbVtcSZ4cwVDcCD/nkPhR2ov0wtZM3CcKbZ7DO+IUknpQ1e4IBNI1bCI ukAkUPECgYEA2hjDeTAg2aDPVroTljLEoFTKzxwZOLtGPe1GHsAzQUsn5Q/4MPvL hPb11vn/gjk9eruMnaNavn37cXhM2WF1eNjFdUNN1z8sVhlxeFyEichedC30qT7F JLDS5rwR2v2Hy+9WE5Sh3bdAnTw9p3M5DTkhzOGzHkJPqKpJJbv7kSsCgYEAzuVk 85apqPO4zhzd2t6Ce4AhhIrRtHyAswpvujov89dV0njKEIduEPeU1b0J8Yoywd5H fAxtTzyoSyPo5k7wNldzC2yuwMZUkPHTs2AJoMmLCs/sIm7RmwD/0QyeSq1c86NY 5PdN1k8FBwNJoRj4bmSewG6bU1YScPBUco4K31ECgYBAZ1mp+t1ohsVTjgVksD9C PWfaeyiziRH4DY5MXGOGX4Q7b1jSVjjGXgtA3XhnYzHVgXbHwkp0wE01N9oxw0bm VdURiLt+2aflEQRiD6gP6/yAgGWkaOXcrN7KxjTbn0yvm1Cm3ZhjGc9Z30UxoqPx VLK3htrXD6voWcjZ/MYQGwKBgCfx2AzDgNwSuhJpNlgkb5LCiTYayyQKiZqHoEyX IVz1rScmlPSeAR0bf8vIZJwSr3wIm0qS56PNtTxUoYmdofkL5zVEdXTYlckqN1JB b3pwjvy2agO325YOmlVVF1aZpAGR/v6t1IRU20agVt3YgVZm7NOAR1LOol7Vt6gq TmURAoGANaODGV6QzKvfrKxivcWmZdXAkGq8lZyGoOXUZQ+p3bcCDVqTlMQgvTmQ k0d7pnB0S/tr63Gkau8O+454+f4woQvHFQxQ+o7AH5pmukG1FJ90VDLHC7tsHQgs eRkUmsWTOprBr25l8k6ucxQivu/O0yBJLz0A1lMdw6Z7VH3BDO4= -----END RSA PRIVATE KEY-----
	SSH Public Key	ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCwQ1tzklpsKYhRJMkmX0i5sgJyoc5T7gioMDEob1ro1S9n3ZcPtal3uDWR09+8FPYB9pOrXeUL0MPjxbwU/UvgzNcf0fwH4CfxOhVxZEAVODU+Ro8Hx4RRx01BRd0ywW3ol4OrzbuGlhlySpqqwcmZ/N9wLyFPA/oXOsIw7r7NWu1Ns7vQ92SAXS+CUQ8fcPrxy9bLe2OKb5fWU3ZKkZBY9UaRaBL1Yf0a80A2tKGNu9eu8tpNszLvAxL7yZnYTUwAzYyW2WSQD4hk8iBlDO/Uly3m4SAl

COMPONENT	ATTRIBUTE	ATTRIBUTE VALUE
		GPN5KFlnYrRXH7uzvxaHnMLFJvWCV/Z1RPvlJjaatNpRN6dQpzAju2Ob ssh-key-2024-10-14
	FDA Laptop Path to Keys	C:\GPT2ComputeInstance\SSHKeys

Appendix B: A Note about Dynamic Groups

What is a Dynamic Group?

A **Dynamic Group** in OCI is a group that contains **resources** (such as compute instances, databases, etc.) that match specific rules. These rules are based on the resource's metadata and allow OCI to dynamically assign resources to groups. This is especially useful when you want to programmatically grant certain OCI resources (like an instance running an application) access to OCI services (e.g., Object Storage).

When is a Dynamic Group Required?

A **Dynamic Group** is typically required when:

1. **You want to programmatically manage access to OCI services**, such as when an OCI compute instance or an APEX application needs to interact with OCI Object Storage, databases, or other resources **without user intervention**.
2. **You do not want to rely on Pre-Authenticated Requests (PARs)** for security reasons, as PARs can be shared externally, and you need more granular control over what actions can be performed.
3. **You prefer to manage permissions using OCI policies** and not rely on manually creating PARs for each bucket or object.

With a **Dynamic Group**, you create policies to give permissions to the group, and the resources in that group inherit those permissions, making it ideal for applications that need continuous, automated access to resources (like Oracle APEX or custom OCI applications).

Why You Don't Need a Dynamic Group with a Bucket-Level PAR

When you use a **Pre-Authenticated Request (PAR)** for the **entire bucket**:

- The PAR acts like a URL that anyone with the link can use (depending on the permissions set when creating the PAR).
- You can perform operations such as uploading, downloading, or listing objects in the bucket directly using the PAR URL.
- Since the PAR grants access, the **APEX application** or any other resource does not need additional **permissions or policies** through a **Dynamic Group**. You just need the **PAR URL** to interact with the bucket.

Use Case Comparison:

1. **Using PAR**:
    - You created a **PAR for the bucket**, allowing users or applications to upload files to the bucket using the provided PAR URL.
    - **Simpler**: You don't need a Dynamic Group or policies.

    - **Manual**: You'll need to create or manage the PARs if permissions or access needs to change.
2. **Using a Dynamic Group**:
    - Ideal if your APEX application, compute instance, or other resources need continuous access to OCI Object Storage without exposing a URL publicly.
    - **More secure**: Access is tightly controlled via policies and is not dependent on exposing URLs like PARs.
    - **Automated**: Once the group and policies are set up, resources are granted access dynamically.

Conclusion

Since you are using a **bucket-level PAR**, a **Dynamic Group** is not required in your case. The PAR provides sufficient access for your APEX application to upload files to OCI Object Storage. However, if your use case evolves to where you need more granular or programmatic control without using PARs, **Dynamic Groups** would be a good option.

Appendix C: Pre-Authenticated Request (PAR) vs. Credentials

Pre-Authenticated Request (PAR) vs. Credentials

1. Pre-Authenticated Request (PAR)

A **PAR** is a URL that provides temporary access to a resource (such as a bucket or object in OCI Object Storage) without the need for credentials. It is often used for sharing or accessing resources publicly or semi-privately when you don't want to involve user authentication or policies.

Key Characteristics of PAR:

- **Bucket-Level PAR**: If you create a PAR for the entire bucket, anyone with the PAR URL can upload, download, or manage files in the bucket (based on the permissions set for the PAR). It simplifies access since no further authentication is required beyond knowing the URL.
- **Easy Sharing**: You can share a PAR publicly or within a specific group without needing them to authenticate.
- **Expiration**: PARs can be set to expire after a specific time, adding a layer of temporary access.
- **Limited Scope**: PARs are usually used for specific operations (like upload/download) and are attached to a specific resource, such as a bucket or an object.
- **No Authentication Needed**: When using a bucket-level PAR, you only need the PAR URL to access the bucket, so no credentials are needed in your APEX application.

When to Use a PAR:

- **Simpler access** for non-authenticated users or public applications.
- **Sharing files** with users who don't have OCI credentials.
- **Short-term access** where access can be revoked by expiring the PAR or deleting it.

2. OCI Credentials

OCI **credentials** refer to a combination of API keys and security policies that grant programmatic access to OCI resources. When you use **credentials**, you authenticate your access with a combination of an API key (for programmatic access) and policies that govern what actions the authenticated user or resource can perform.

Key Characteristics of OCI Credentials:

- **Strong Authentication**: OCI credentials (such as API keys, user OCIDs, or IAM roles) require authentication for every request, ensuring that only authorized users or resources can access the storage.
- **Granular Control**: You can specify exactly what actions a resource can perform (e.g., uploading files, listing buckets, etc.) through policies tied to your OCI credentials.
- **Programmatic Access**: Ideal for continuous, automated processes, such as when your APEX application, a server, or another OCI service needs to interact with OCI Object Storage without user involvement.

- **Security**: More secure in long-term usage as credentials are protected, and access can be tightly controlled by OCI Identity and Access Management (IAM) policies.

When to Use OCI Credentials:

- **For automated applications** that need continuous, programmatic access to resources without relying on a shared URL.
- **Higher security requirements**: Credentials and policies provide stronger security for enterprise-level applications where fine-grained control over access is required.
- **Granular control over access**: Policies tied to credentials allow fine-tuned permissions based on roles, actions, and resources.

Comparison: PAR vs. Credentials

Feature	Pre-Authenticated Request (PAR)	OCI Credentials
Access Method	URL-based (no authentication needed)	Programmatic access with API keys and user authentication
Granular Control	Limited to bucket or object-level access	Fine-grained control over actions and access policies
Usage Type	Public, shared access, or temporary access	Private, continuous, and automated access
Expiration	PARs can expire or be revoked manually	Credentials do not expire but can be revoked or rotated
Complexity	Easy to implement (just generate a URL)	Requires setting up API keys, policies, and authentication flows
Security	Less secure (anyone with the URL can access the resource)	More secure (access is controlled through policies and credentials)
Application Type	Ideal for public or semi-public sharing or temporary access	Ideal for long-term, automated applications or enterprise-level apps
Example Use Case	Sharing a file or allowing uploads from a website temporarily	Application needing continuous access to storage for data processing

---

Why You Don’t Need OCI Credentials with a Bucket-Level PAR

Since you’ve created a **bucket-level PAR**, the PAR URL already grants permission to access the bucket for the specified actions (uploading files, downloading files, etc.). Your **Oracle APEX application** can interact with the bucket using just the PAR URL.

In this case:

- You **do not need to configure OCI credentials** because the PAR already handles the access control.
- **No authentication** is required from the APEX application, as access is granted via the PAR URL itself.
- You only need to ensure that your APEX PL/SQL code correctly references the **PAR URL** for uploading or managing files.

When to Use OCI Credentials Instead of PAR

If your use case evolves, and you want:

- **More secure access**, especially if the APEX app is expected to interact with OCI services without exposing URLs publicly.
- **Granular control** over actions such as file management, directory listing, and bucket administration.
- **Long-term automated access** where credentials would allow seamless interactions with Object Storage without relying on expiring or manually-managed URLs.

In such scenarios, switching to OCI credentials and policies would be a better solution.

Conclusion

For your current setup, where you're using a **bucket-level PAR**, there is no need for OCI credentials because the PAR already grants access to the bucket. This simplifies the interaction between your Oracle APEX application and the OCI Object Storage bucket, making the **dynamic group** and **OCI credentials** unnecessary for now.

However, if your application requires more security or flexibility in the future, OCI credentials could be explored to replace the PAR-based access.

Appendix D: Step-by-Step Instructions for Creating an OLTP Database in Oracle OCI Free Tier

Prerequisites

- An active **Oracle Cloud Free Tier account**.
- Basic knowledge of navigating the OCI console.

Step 1: Log in to Oracle Cloud Console

1. Go to Oracle Cloud Console and log in using your credentials.

Step 2: Navigate to Autonomous Databases

1. From the **OCI Dashboard**, click the **Menu** icon (three horizontal lines) in the top-left corner.
2. Under **Databases**, click on **Autonomous Database**.

Step 3: Start Creating an Autonomous Database

1. Click on the **Create Autonomous Database** button in the top-right corner.

Step 4: Configure Autonomous Database Details

1. **Compartment**: Select your default compartment (it might be "root" unless you've created a custom compartment).
2. **Display Name**: Enter a meaningful name for your database (e.g., `FreeTierATPDB`).
3. **Database Name (DB Name)**: Choose a short, unique name (up to 14 characters) for your database (e.g., `ATPFree`).
4. **Workload Type**: Select **Transaction Processing**. This is important for OLTP use cases like real-time transactions, data processing, and quick queries.

Step 5: Select Infrastructure Type (Free Tier Defaults to Shared)

1. **Infrastructure Type**: This will default to **Shared Infrastructure**, which is required for the Free Tier.
    - The Free Tier only allows the use of **shared infrastructure** (you won't have to choose here since it's the only option in Free Tier).

Step 6: Choose Free Tier Database Configuration

1. **OCPU Count**: Set the OCPUs to **1 OCPU**, which is the maximum allowed in the Free Tier.
2. **Storage (TB)**: Set the storage to **20 GB**, the maximum free allocation (you can choose less if needed).
    - The Free Tier allows up to 20 GB of storage for Autonomous Databases.

Step 7: Set Database Credentials

1. **Admin Password**: Set a password for the `ADMIN` user. This is required for accessing and managing your database.
    - The password must meet the following requirements:
        - At least 12 characters.
        - At least one uppercase letter.
        - At least one lowercase letter.
        - At least one numeric character.
        - At least one special character.

Step 8: Configure Network Access

1. **Access Type**: Select **Secure Access from Anywhere**. This will make the database accessible over the public internet (with appropriate authentication).
2. **Virtual Cloud Network (VCN)**: Leave the VCN settings at their default for Free Tier. OCI automatically manages the network setup.

Step 9: Backup and Recovery (Automatic in Free Tier)

1. Backups are managed automatically in the Free Tier, so no configuration is required. Your database is backed up regularly by OCI.

Step 10: Advanced Options (Optional)

1. **Auto Scaling**: In the Free Tier, **Auto Scaling** is enabled automatically but limited to 1 OCPU.
2. **Data Safe**: Optionally, enable **Oracle Data Safe** for free if you want to add advanced security features like data masking, auditing, etc.
3. **Encryption**: Data is always encrypted in Oracle Autonomous Databases, so no further action is required here.

Step 11: License Type

1. **License Type**: Leave the license option set to **License Included**, as this is the only option available in the Free Tier.

Step 12: Review and Create

1. Review your configurations and ensure all settings are correct.
2. Click **Create Autonomous Database** to start provisioning the database.

Step 13: Monitor the Database Creation

1. The provisioning process may take a few minutes. You can track the progress on the **Autonomous Database** page.

2. Once the status shows **Available**, your database is ready for use.

Step 14: Connect to the Database

1. **Database Connection**: To connect to your OLTP database, you will need the **Database Wallet** file.
    - From the Autonomous Database details page, click **DB Connection**.
    - Click **Download Wallet**.
    - Set a wallet password (make sure to remember it) and download the `.zip` file.
2. **Configure SQL Developer or Other Tools**:
    - Extract the `.zip` wallet file.
    - Open Oracle SQL Developer (or another SQL tool), and configure the connection using the TNS (Transparent Network Substrate) strings provided in the wallet.
    - Use the `ADMIN` user credentials and the password you set during database creation.

Free Tier Limitations to Keep in Mind

- **Maximum 1 OCPU**: You cannot increase the number of OCPUs beyond 1 in the Free Tier.
- **Maximum 20 GB of storage**: The Free Tier gives you a maximum of 20 GB of storage for Autonomous Databases.
- **Auto Scaling Limited**: Auto scaling is limited to 1 OCPU in the Free Tier.
- **Shared Infrastructure Only**: The Free Tier only supports shared infrastructure for Autonomous Databases.

Appendix E: Why a Separate Compute Instance

A separate **OCI Compute Instance** is recommended instead of using the **Oracle Cloud Autonomous Database (ADB) code editor** for running and fine-tuning models like GPT-2 for several important reasons:

1. Resource Limitations of ADB (Cloud Tier):

- **Cloud Tier Code Editor**: The cloud tier's built-in code editor is designed for lightweight, database-centric tasks such as PL/SQL scripting, simple data manipulation, and Oracle-specific database functions. It is not optimized for running large machine learning models like GPT-2, which require significant memory, CPU, and GPU resources.
- **Compute Power**: Machine learning tasks, especially fine-tuning GPT-2, are resource-intensive. They require large amounts of memory, processing power, and sometimes even GPUs for efficient execution. The **Oracle Cloud Autonomous Database** does not offer the flexibility to configure high-performance computing resources (like GPUs) for running models efficiently. A dedicated **Compute Instance** allows you to scale up resources based on your needs.

2. Customization and Flexibility for Machine Learning Libraries:

- **Compute Instance**: A compute instance gives you full control over the environment. You can install any custom libraries, machine learning frameworks (such as TensorFlow, PyTorch, Hugging Face Transformers), and specific dependencies that are necessary for running advanced AI models.
- **Cloud Tier**: The ADB cloud tier environment is optimized for database operations and may not allow the installation of external machine learning libraries or frameworks. It is not designed to support the full ecosystem required for deep learning models (e.g., complex Python dependencies, TensorFlow, etc.).

3. Separation of Concerns:

- **Compute Instance**: By separating the machine learning model from the database environment, you achieve a **separation of concerns**. The compute instance handles the AI-related tasks (loading models, generating text, fine-tuning), while the **Oracle APEX and Database** focus on data management, front-end applications, and user interaction. This separation improves the scalability, reliability, and performance of both components.
- **Cloud Tier Code Editor**: Loading and running the model in the code editor would unnecessarily burden the database with tasks it isn't optimized to handle, leading to performance issues, slow response times, and possibly exceeding the resource limits of the ADB cloud tier.

4. Scalability and Flexibility:

- **Compute Instance**: With a compute instance, you can easily scale resources up or down based on demand. For instance, if the model requires more memory or processing power, you can adjust the number of OCPUs, memory, or even switch to a GPU-enabled instance. This flexibility is critical for fine-tuning large models or handling multiple users concurrently.
- **Cloud Tier**: The cloud tier is more rigid in terms of scalability. You cannot dynamically adjust computing resources, which is necessary for handling the computational load of GPT-2 model training or fine-tuning.

5. Efficient Resource Management and Cost Control:

- **Compute Instance**: You can optimize costs by choosing the right shape (CPU/memory) and only scaling up when necessary (e.g., during fine-tuning). You can stop and start the instance as needed, saving costs when it's not in use. Compute instances also allow you to pick shapes like **VM.Standard.E3.Flex** (customized CPU/memory configurations), which are cost-effective for running GPT-2 models.
- **Cloud Tier**: While the Oracle ADB cloud tier offers a free tier, its resource limits make it unsuitable for intensive machine learning tasks, and upgrading to a higher tier with more resources can become expensive. Additionally, overloading the cloud tier with machine learning tasks can affect database performance, leading to higher costs and inefficient resource usage.

6. GPUs and Machine Learning Acceleration:

- **Compute Instance**: For machine learning models like GPT-2, access to GPU instances can significantly speed up training and fine-tuning tasks. OCI allows you to provision **GPU-accelerated compute instances** (e.g., using NVIDIA GPUs), which are not available in the ADB cloud tier. GPUs are essential for complex model training and fine-tuning due to their ability to handle parallel processing efficiently.
- **Cloud Tier**: The cloud tier does not support GPUs or the high-performance computing capabilities necessary for deep learning workloads.

7. Network and Storage Integration:

- **Compute Instance**: When using a compute instance, you can configure network rules, storage options, and integrate easily with OCI Object Storage for retrieving or storing large datasets. The compute instance can handle heavy data processing tasks independently, without affecting the primary database's performance.
- **Cloud Tier**: The ADB environment is tightly coupled with the database's own performance. Running heavy workloads, such as retrieving large datasets or processing multiple documents, may affect the database performance and user experience within the APEX application.

Summary:

Using a **dedicated compute instance** for running and fine-tuning GPT-2 is crucial because it:

- Provides the required computational resources (CPU, memory, and GPU) for handling large machine learning models.
- Offers flexibility for installing custom machine learning frameworks and managing dependencies.
- Improves scalability and performance by separating AI processing from database management.
- Allows better resource management, which helps control costs while ensuring optimal performance.

In contrast, the **Oracle Cloud Autonomous Database code editor** is better suited for lightweight, database-centric operations and is not equipped to handle the computational demands of machine learning workloads like GPT-2.