```
!pip install  llama-index-embeddings-vertex \
  llama-index-llms-vertex \
  llama-index-vector_stores-vertexaivectorsearch \
  llama-index-llms-langchain \
  llama-index-llms-fireworks
!pip install google-cloud-aiplatform google-auth-httplib2 google-auth-oauthlib --upgrade
!pip install llama-index pinecone-client
```

```
Requirement already satisfied: jiter<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from openai>=1.14.0->llama-index-agent-openai<0.4.0,>=
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3.0.0,>=2.7.0->llama-index-core<0
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3.0.0,>=2.7.0->llama-index-core<0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31.0->llama-index-core<0.12.
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy>=1.4.49->SQLAlchemy[asyncio]>=1.4.49-
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from typing-inspect>=0.8.0->llama-index-core<0.
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json->llama-index-core<0.1
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->llama-index-legacy<0.10.0,>=0.9.48
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->llama-index-legacy<0.10.0,>=0.9.48->llama-in
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->llama-index-legacy<0.10.0,>=0.9.48->llama-
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio->httpx->llama-index-core<0.12.0,>=0.11.22->l
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.10/dist-packages (from marshmallow<4.0.0,>=3.18.0->dataclasses-json->lla
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->llama-index-legacy<0.10.
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from yarl<2.0,>=1.12.0->aiohttp<4.0.0,>=3.8.6->llama-
```

```
# Imports Core
import os
import json

# LangChain and Llama Index
from langchain.schema import Document

# Llama Index core components
from llama_index.core import (
    Document,
    PromptTemplate,
    Settings,
    SimpleDirectoryReader,
    StorageContext,
    SummaryIndex,
    VectorStoreIndex,
)

# Llama Index agent and query engine
from llama_index.core.agent import ReActAgent
from llama_index.core.base.base_query_engine import BaseQueryEngine

# Llama Index parsing and objects
from llama_index.core.node_parser import SentenceSplitter
from llama_index.core.objects import ObjectIndex

# Llama Index prompts
from llama_index.core.prompts import LangchainPromptTemplate
from llama_index.core.prompts.base import BasePromptTemplate

# Llama Index tools
from llama_index.core.tools import QueryEngineTool, ToolMetadata

# Llama Index embeddings and models using Vertex AI
from llama_index.embeddings.vertex import VertexTextEmbedding
from llama_index.llms.vertex import Vertex

# Llama Index vector store for Vertex AI Vector Search
```

```
# Llama Index Vector Store for Vertex AI Vector Search
from llama_index.vector_stores.vertexaivectorsearch import VertexAIVectorStore

# Google Cloud libraries
from google.cloud import aiplatform, storage
from google.auth import default
import vertexai


# Project Configuration
# Authenticate and initialize credentials
credentials, _ = default()

# Set project and region
PROJECT_ID = "theta-function-429605-j0"
LOCATION = "us-west1"
BUCKET_NAME = "news_articles-bucket"
VS_DIMENSIONS = 768
VS_INDEX_NAME = "llamaindex_doc_index"
VS_INDEX_ENDPOINT_NAME = "llamaindex_doc_endpoint"
VERTEX_TEXT_EMBEDDING="text-embedding-004"

# Initialize AI Platform and Vertex AI with project and location
aiplatform.init(project=PROJECT_ID, location=LOCATION, credentials=credentials)
vertexai.init(project=PROJECT_ID, location=LOCATION)

# Set the project configuration for gcloud CLI
!gcloud config set project {PROJECT_ID}

# Embedding Model
GEMINI_EMBEDDING_MODEL = VertexTextEmbedding(VERTEX_TEXT_EMBEDDING,credentials=credentials)
```

```
Updated property [core/project].
```

```
# Create Vector Index
def initialize_vector_index(index_name, dimensions):
    """
    Creates a Vector Search index in Vertex AI or retrieves an existing one if it already exists.

    Args:
        index_name (str): The name for the index.
        dimensions (int): The dimensionality of the embeddings for the index.

    Returns:
        aiplatform.MatchingEngineIndex: The created or retrieved index.
    """
    print(f"Initializing Vector Search index '{index_name}'...")

    existing_indices = [
        index.resource_name
```

```
        for index in aiplatform.MatchingEngineIndex.list(filter=f"display_name={index_name}")
    ]

    if not existing_indices:
        print(f"Creating new Vector Search index '{index_name}'...")
        vector_index = aiplatform.MatchingEngineIndex.create_tree_ah_index(
            display_name=index_name,
            dimensions=dimensions,
            distance_measure_type="COSINE_DISTANCE",
            shard_size="SHARD_SIZE_SMALL",
            index_update_method="STREAM_UPDATE",
            approximate_neighbors_count=5,
        )
        print(f"Vector Search index '{vector_index.display_name}' created with resource name {vector_index.resource_name}")
    else:
        vector_index = aiplatform.MatchingEngineIndex(index_name=existing_indices[0])
        print(f"Using existing Vector Search index '{vector_index.display_name}' with resource name {vector_index.resource_name}")

    return vector_index
vs_index = initialize_vector_index(VS_INDEX_NAME, VS_DIMENSIONS)
```

```
Initializing Vector Search index 'llamaindex_doc_index'...
Using existing Vector Search index 'llamaindex_doc_index' with resource name projects/184982369838/locations/us-west1/indexes/2998209879270752256
```

```
# Create Vector Search Endpoint
def initialize_vector_endpoint(endpoint_name):
    """
    Creates a Vector Search endpoint in Vertex AI or retrieves an existing one if it already exists.

    Args:
        endpoint_name (str): The name for the endpoint.

    Returns:
        aiplatform.MatchingEngineIndexEndpoint: The created or retrieved endpoint.
    """
    existing_endpoints = [
        endpoint.resource_name
        for endpoint in aiplatform.MatchingEngineIndexEndpoint.list(filter=f"display_name={endpoint_name}")
    ]

    if not existing_endpoints:
        print(f"Creating new Vector Search endpoint '{endpoint_name}'...")
        vector_endpoint = aiplatform.MatchingEngineIndexEndpoint.create(
            display_name=endpoint_name, public_endpoint_enabled=True
        )
        print(f"Vector Search endpoint '{vector_endpoint.display_name}' created with resource name {vector_endpoint.resource_name}")
    else:
        vector_endpoint = aiplatform.MatchingEngineIndexEndpoint(index_endpoint_name=existing_endpoints[0])
        print(f"Using existing Vector Search endpoint '{vector_endpoint.display_name}' with resource name {vector_endpoint.resource_name}")
```

```
    return vector_endpoint

vs_endpoint = initialize_vector_endpoint(VS_INDEX_ENDPOINT_NAME)
```

```
Using existing Vector Search endpoint 'llamaindex_doc_endpoint' with resource name projects/184982369838/locations/us-west1/indexEndpoints/605224
```

```
# Deploy Vector Search Endpoint
def deploy_vector_endpoint(vector_index, vector_endpoint, index_name):
    """
    Deploys a Vector Search index to the specified endpoint in Vertex AI.

    Args:
        vector_index (aiplatform.MatchingEngineIndex): The vector index to deploy.
        vector_endpoint (aiplatform.MatchingEngineIndexEndpoint): The endpoint to deploy the index to.
        index_name (str): The display name for the deployed index.

    Returns:
        aiplatform.MatchingEngineIndexEndpoint: The endpoint with the deployed index.
    """
    existing_deployments = [
        (deployed_index.index_endpoint, deployed_index.deployed_index_id)
        for deployed_index in vector_index.deployed_indexes
    ]

    if not existing_deployments:
        print(f"Deploying Vector Search index '{vector_index.display_name}' to endpoint '{vector_endpoint.display_name}'...")
        deployed_index = vector_endpoint.deploy_index(
            index=vector_index,
            deployed_index_id=index_name,
            display_name=index_name,
            machine_type="e2-standard-16",
            min_replica_count=1,
            max_replica_count=1,
        )
        print(f"Vector Search index '{vector_index.display_name}' deployed to endpoint '{deployed_index.display_name}'")
    else:
        deployed_index = aiplatform.MatchingEngineIndexEndpoint(index_endpoint_name=existing_deployments[0][0])
        print(f"Vector Search index '{vector_index.display_name}' is already deployed at endpoint '{deployed_index.display_name}'")

    return deployed_index
vs_deployed_index = deploy_vector_endpoint(vs_index, vs_endpoint, VS_INDEX_NAME)
```

```
Vector Search index 'llamaindex_doc_index' is already deployed at endpoint 'llamaindex_doc_endpoint'
```

```
# Initialize LLM and Storage Context
def setup_storage_context(
    vector_index,
    vector_endpoint,
    embed_model_name="text-embedding-004",
    llm_model_name="gemini-pro",
```

```
    project_id = PROJECT_ID,
    region = LOCATION,
    gcs_bucket=BUCKET_NAME,
    credentials=None
):
    """
    Initializes the storage context for Vertex AI's Vector Store, embedding model, and LLM.

    Args:
        vector_index: The Vertex AI Vector Search index.
        vector_endpoint: The Vertex AI Vector Search endpoint.
        embed_model_name: The name of the embedding model.
        llm_model_name: The name of the LLM model.
        project_id: The Google Cloud project ID.
        region: The Google Cloud region.
        gcs_bucket: The GCS bucket name for storing embeddings.
        credentials: Google Cloud credentials.

    Returns:
        StorageContext: Configured storage context for vector search and language models.
    """
    try:
        # Vector store
        vector_store = VertexAIVectorStore(
            project_id=project_id,
            region=region,
            index_id=vector_index.resource_name,
            endpoint_id=vector_endpoint.resource_name,
            gcs_bucket_name=gcs_bucket,
        )

        storage_context = StorageContext.from_defaults(vector_store=vector_store)

        embedding_model = GEMINI_EMBEDDING_MODEL
        language_model = Vertex(llm_model_name)

        Settings.embed_model = embedding_model
        Settings.llm = language_model

        return storage_context
    except Exception as e:
        print(f"Error initializing storage context: {e}")
        raise
storage_context = setup_storage_context(vs_index, vs_endpoint)
```

```
def list_and_download_document_blobs(bucket_name):
    """Lists all the JSON blobs in the specified folder and returns their content as a list of dictionaries."""
    # Data
    documents = []
```

```
    # Initialize a storage client
    storage_client = storage.Client()

    # Retrieve blobs in the specified bucket with the given prefix (folder)
    blobs = storage_client.list_blobs(bucket_name)

    # Iterate through the blobs and collect their JSON content
    for blob in blobs:
        #print(f"Processing blob: {blob.name}")
        if blob.name.endswith('.json'):  # Check if the blob is a JSON file
            # Download the blob content as a string and decode it
            json_content = blob.download_as_string().decode('utf-8')
            # Parse the JSON string into a Python dictionary
            try:
                json_data = json.loads(json_content)
                documents.append(Document(text=json.dumps(json_data)))
            except json.JSONDecodeError as e:
                print(f"Failed to decode JSON from {blob.name}: {e}")

    return documents

documents_data = list_and_download_document_blobs(BUCKET_NAME)
```

```
# Create Vector Store
vs_storage_context = VectorStoreIndex.from_documents(documents_data, storage_context=storage_context )

# Query Engine
query_engine = vs_storage_context.as_query_engine()

# Prompt Templage
PROMPT_TEMPLATE = query_engine.get_prompts()['response_synthesizer:text_qa_template'].default_template
```

```
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:Upserting datapoints MatchingEngineIndex index: projects/184982369838/location
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:MatchingEngineIndex index Upserted datapoints. Resource name: projects/184982
```

```
# Format Prompt
def format_prompt(context_text, query_text):
    """
    Formats the prompt with context and query.

    Args:
        context_text (str): The context information used to provide background in the prompt.
        query_text (str): The user's question or query to insert into the prompt.

    Returns:
        str: A formatted prompt string.
    """
    if PROMPT_TEMPLATE!="":
```

```
        return PROMPT_TEMPLATE.format(context_str=context_text, query_str=query_text)
    else:
        print("Error: Prompt template not available.")
        return None

# Quaery and Context
context_text = documents_data[1].text
query_text = "What is the stock Price of Apple?"

# Format the prompt using the cached template
formatted_prompt = format_prompt(context_text, query_text)

# Print the formatted prompt
if formatted_prompt:
    print(formatted_prompt)
```

```
Context information is below.
---------------------
{"ticker": "AAPL", "title": "Is Warren Buffett's $2.9 Billion Bet and Sales of Biggest Holding Apple a Warning for Wall Street?", "summary": "War
---------------------
Given the context information and not prior knowledge, answer the query.
Query: What is the stock Price of Apple?
Answer:
```

```
# Create Embedding for Input Prompt
def create_embeddings(prompt: str) -> list:
    """
    Generate embeddings for the provided prompt using Vertex AI Text Embedding model.

    Args:
        prompt (str): The input text for which embeddings need to be created.

    Returns:
        list: A list of embeddings for the input prompt.
    """
    # Create embeddings
    embeddings = GEMINI_EMBEDDING_MODEL.get_text_embedding(prompt)  # Pass as a list of prompts
    return embeddings # Return the first (and only) set of embeddings

embeddings = create_embeddings(formatted_prompt)
print("Generated Embeddings:", embeddings)
```

```
Generated Embeddings: [0.008891763165593147, 0.009792226366698742, 0.0015674149617552757, -0.02508361265063286, 0.03509104624390602, 0.066433601
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```