```
            Directory structure:
└── rag/
    ├── README.md
    ├── pyproject.toml
    ├── .env.example
    ├── deployment/
    │   ├── deploy.py
    │   ├── grant_permissions.sh
    │   └── run.py
    ├── eval/
    │   ├── test_eval.py
    │   └── data/
    │       ├── conversation.test.json
    │       └── test_config.json
    ├── rag/
    │   ├── __init__.py
    │   ├── agent.py
    │   ├── prompts.py
    │   └── shared_libraries/
    │       └── prepare_corpus_and_data.py
    └── tests/
        └── __init__.py
```

```
================================================
File: README.md
================================================
```

# Documentation Retrieval Agent

Overview

This agent is designed to answer questions related to documents you uploaded to Vertex AI RAG Engine. It utilizes Retrieval-Augmented Generation (RAG) with the Vertex AI RAG Engine to fetch relevant documentation snippets and code references, which are then synthesized by an LLM (Gemini) to provide informative answers with citations.

![RAG Architecture](RAG_architecture.png)

This diagram outlines the agent's workflow, designed to provide informed and context-aware responses. User queries are processed by agent development kit. The LLM determines if external knowledge (RAG corpus) is required. If so, the `VertexAiRagRetrieval` tool fetches relevant information from the configured Vertex RAG Engine corpus. The LLM then synthesizes this retrieved information with its internal knowledge to generate an accurate answer, including citations pointing back to the source documentation URLs.

Agent Details

Attribute	Details
Interaction Type	Conversational
Complexity	Intermediate
Agent Type	Single Agent
Components	Tools, RAG, Evaluation
Vertical	Horizontal

Agent Architecture

![RAG](RAG_workflow.png)

Key Features

*   **Retrieval-Augmented Generation (RAG):** Leverages [Vertex AI RAG Engine](https://cloud.google.com/vertex-ai/generative-ai/docs/rag-overview) to fetch relevant documentation.
*   **Citation Support:** Provides accurate citations for the retrieved content, formatted as URLs.
*   **Clear Instructions:** Adheres to strict guidelines for providing factual answers and proper citations.

Setup and Installation Instructions

Prerequisites

*   **Google Cloud Account:** You need a Google Cloud account.
*   **Python 3.9+:** Ensure you have Python 3.9 or a later version installed.
*   **Poetry:** Install Poetry by following the instructions on the official Poetry website: https://python-poetry.org/docs/
*   **Git:** Ensure you have git installed.

Project Setup with Poetry

1. **Clone the Repository:**

```bash
    git clone https://github.com/google/adk-samples.git
    cd adk-samples/python/agents/RAG
    ```

2. **Install Dependencies with Poetry:**

    ```bash
    poetry install
    ```

    This command reads the `pyproject.toml` file and installs all the necessary dependencies into a virtual environment managed by Poetry.

3. **Activate the Poetry Shell:**

    ```bash
    poetry env activate
    ```

    This activates the virtual environment, allowing you to run commands within the project's environment.

    Make sure the environment is active. If not, you can also activate it through

     ```bash
    source .venv/bin/activate
    ```

4. **Set up Environment Variables:**

Rename the file ".env.example" to ".env"

Follow the steps in the file to set up the environment variables.

5. **Setup Corpus:**

If you have an existing corpus in Vertex AI RAG Engine, please set corpus information in your .env file. For example: RAG_CORPUS='projects/123/locations/us-central1/ragCorpora/456'.

If you don't have a corpus setup yet, please follow "How to upload my file to my RAG corpus" section. The `prepare_corpus_and_data.py` script will automatically create a corpus (if needed) and update the `RAG_CORPUS` variable in your `.env` file with the resource name of the created or retrieved corpus.

How to upload my file to my RAG corpus

The `rag/shared_libraries/prepare_corpus_and_data.py` script helps you set up a RAG corpus and upload an initial document. By default, it downloads Alphabet's 2024 10-K PDF and uploads it to a new corpus.

1. **Authenticate with your Google Cloud account:**

```bash

gcloud auth application-default login

```

2. **Set up environment variables in your `.env` file:**

Ensure your `.env` file (copied from `.env.example`) has the following variables set:

```

GOOGLE_CLOUD_PROJECT=your-project-id

GOOGLE_CLOUD_LOCATION=your-location  # e.g., us-central1

```

3. **Configure and run the preparation script:**

*   **To use the default behavior (upload Alphabet's 10K PDF):**

    Simply run the script:

    ```bash
    python rag/shared_libraries/prepare_corpus_and_data.py
    ```

    This will create a corpus named `Alphabet_10K_2024_corpus` (if it doesn't exist) and upload the PDF `goog-10-k-2024.pdf` downloaded from the URL specified in the script.

*   **To upload a different PDF from a URL:**

    a. Open the `rag/shared_libraries/prepare_corpus_and_data.py` file.

    b. Modify the following variables at the top of the script:

       ```python
       # --- Please fill in your configurations ---
       # ... project and location are read from .env ...
       CORPUS_DISPLAY_NAME = "Your_Corpus_Name"  # Change as needed
       CORPUS_DESCRIPTION = "Description of your corpus" # Change as needed
       PDF_URL = "https://path/to/your/document.pdf"  # URL to YOUR PDF document
       PDF_FILENAME = "your_document.pdf"  # Name for the file in the corpus
       # --- Start of the script ---
       ```

    c. Run the script:

       ```bash
       python rag/shared_libraries/prepare_corpus_and_data.py
       ```

*   **To upload a local PDF file:**

    a. Open the `rag/shared_libraries/prepare_corpus_and_data.py` file.

b. Modify the `CORPUS_DISPLAY_NAME` and `CORPUS_DESCRIPTION` variables as needed (see above).

c. Modify the `main()` function at the bottom of the script to directly call `upload_pdf_to_corpus` with your local file details:

```python
def main():
  initialize_vertex_ai()
  corpus = create_or_get_corpus() # Uses CORPUS_DISPLAY_NAME & CORPUS_DESCRIPTION

  # Upload your local PDF to the corpus
  local_file_path = "/path/to/your/local/file.pdf" # Set the correct path
  display_name = "Your_File_Name.pdf" # Set the desired display name
  description = "Description of your file" # Set the description

  # Ensure the file exists before uploading
  if os.path.exists(local_file_path):
      upload_pdf_to_corpus(
          corpus_name=corpus.name,
          pdf_path=local_file_path,
          display_name=display_name,
          description=description
      )
  else:
      print(f"Error: Local file not found at {local_file_path}")

  # List all files in the corpus
  list_corpus_files(corpus_name=corpus.name)
```

d. Run the script:

```bash
python rag/shared_libraries/prepare_corpus_and_data.py
```

More details about managing data in Vertex RAG Engine can be found in the

[official documentation page](https://cloud.google.com/vertex-ai/generative-ai/docs/rag-quickstart).

Running the Agent

You can run the agent using the ADK command in your terminal.

from the root project directory:

1. Run agent in CLI:

    ```bash
    adk run rag
    ```

2. Run agent with ADK Web UI:

    ```bash
    adk web
    ```

    Select the RAG from the dropdown

Example Interaction

Here's a quick example of how a user might interact with the agent:

Example 1: Document Information Retrieval

User: What are the key business segments mentioned in Alphabet's 2024 10-K report?

Agent: According to Alphabet's 2024 10-K report, the key business segments are:

1. Google Services (including Google Search, YouTube, Google Maps, Play Store)

2. Google Cloud (offering cloud computing services, data analytics, and AI solutions)

3. Other Bets (including Waymo for autonomous driving technology)

[Source: goog-10-k-2024.pdf]

Evaluating the Agent

The evaluation can be run from the `RAG` directory using

the `pytest` module:

```

poetry run pytest eval

```

Evaluation Process

The evaluation framework consists of three key components:

1. **test_eval.py**: The main test script that orchestrates the evaluation process. It uses the `AgentEvaluator` from Google ADK to run the agent against a test dataset and assess its performance based on predefined criteria.

2. **conversation.test.json**: Contains a sequence of test cases structured as

a conversation. Each test case includes:

   - A user query (e.g., questions about Alphabet's 10-K report)

   - Expected tool usage (which tools the agent should call and with what parameters)

   - Reference answers (ideal responses the agent should provide)

3. **test_config.json**: Defines evaluation criteria and thresholds:

   - `tool_trajectory_avg_score`: Measures how well the agent uses the appropriate tools

   - `response_match_score`: Measures how closely the agent's responses match the reference answers

When you run the evaluation, the system:

1. Loads the test cases from conversation.test.json

2. Sends each query to the agent

3. Compares the agent's tool usage against expected tool usage

4. Compares the agent's responses against reference answers

5. Calculates scores based on the criteria in test_config.json

This evaluation helps ensure the agent correctly leverages the RAG capabilities to retrieve relevant information and generates accurate responses with proper citations.

Deploying the Agent

The Agent can be deployed to Vertex AI Agent Engine using the following

commands:

```

python deployment/deploy.py

```

After deploying the agent, you'll be able to read the following INFO log message:

```
Deployed agent to Vertex AI Agent Engine successfully, resource name:
projects/<PROJECT_NUMBER>/locations/us-central1/reasoningEngines/
<AGENT_ENGINE_ID>
```

Please note your Agent Engine resource name and update `.env` file accordingly as this is crucial for testing the remote agent.

You may also modify the deployment script for your use cases.

Testing the deployed agent

After deploying the agent, follow these steps to test it:

1. **Update Environment Variables:**

   - Open your `.env` file.

   - The `AGENT_ENGINE_ID` should have been automatically updated by the `deployment/deploy.py` script when you deployed the agent. Verify that it is set correctly:

     ```
     AGENT_ENGINE_ID=projects/<PROJECT_NUMBER>/locations/us-central1/
reasoningEngines/<AGENT_ENGINE_ID>
     ```

2. **Grant RAG Corpus Access Permissions:**

   - Ensure your `.env` file has the following variables set correctly:

     ```
     ```

```
GOOGLE_CLOUD_PROJECT=your-project-id

RAG_CORPUS=projects/<project-number>/locations/us-central1/ragCorpora/<corpus-id>
```

- Run the permissions script:

  ```bash
  chmod +x deployment/grant_permissions.sh

  ./deployment/grant_permissions.sh
  ```

  This script will:

  - Read the environment variables from your `.env` file
  - Create a custom role with RAG Corpus query permissions
  - Grant the necessary permissions to the AI Platform Reasoning Engine Service Agent

3. **Test the Remote Agent:**

   - Run the test script:

     ```bash
     python deployment/run.py
     ```

   This script will:

   - Connect to your deployed agent
   - Send a series of test queries
   - Display the agent's responses with proper formatting

The test script includes example queries about Alphabet's 10-K report. You can modify the queries in `deployment/run.py` to test different aspects of your deployed agent.

Customization

Customize Agent

You can customize system instruction for the agent and add more tools to suit your need, for example, google search.

Customize Vertex RAG Engine

You can read more about [official Vertex RAG Engine documentation](https://cloud.google.com/vertex-ai/generative-ai/docs/rag-quickstart) for more details on customizing corpora and data.

Plug-in other retrieval sources

You can also integrate your preferred retrieval sources to enhance the agent's capabilities. For instance, you can seamlessly replace or augment the existing `VertexAiRagRetrieval` tool with a tool that utilizes Vertex AI Search or any other retrieval mechanism. This flexibility allows you to tailor the agent to your specific data sources and retrieval requirements.

Disclaimer

This agent sample is provided for illustrative purposes only and is not intended for production use. It serves as a basic example of an agent and a foundational starting point for individuals or teams to develop their own agents.

This sample has not been rigorously tested, may contain bugs or limitations, and does not include features or optimizations typically required for a production environment (e.g., robust error handling, security measures, scalability, performance considerations, comprehensive logging, or advanced configuration options).

Users are solely responsible for any further development, testing, security hardening, and deployment of agents based on this sample. We recommend thorough review, testing, and the implementation of appropriate safeguards before using any derived agent in a live or critical system.

```
================================================
File: pyproject.toml
================================================
[project]
name = "rag"
version = "0.1.0"
description = "Answer questions related to Vertex AI documentation using Vertex AI RAG Engine"
authors = [
        { name = "Hangfei Lin", email = "hangfei@google.com" },
        { name = "Chouaieb Nemri", email = "nemri@google.com" },
]

license = "Apache License 2.0"
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.11"
pydantic-settings = "^2.8.1"
tabulate = "^0.9.0"
google-auth = "^2.36.0"
requests = "^2.32.3"
google-adk = "^1.0.0"
google-cloud-aiplatform = { extras = [
        "adk",
        "agent-engines",
], version = "^1.93.0" }
```

```
llama-index = "^0.12"


[tool.poetry.group.dev.dependencies]

pytest = "^8.3.5"

pytest-mock = "^3.14.0"

scikit-learn = "^1.6.1"

pytest-cov = "^6.0.0"

pytest-asyncio = "^0.26.0"

black = "^25.1.0"


[build-system]

requires = ["poetry-core>=2.0.0,<3.0.0"]

build-backend = "poetry.core.masonry.api"
```

```
================================================

File: .env.example

================================================

# Copy as .env file and fill your values below

# Run ./update_dotenv_example.sh to update .env-example from your .env file.


# Choose Model Backend: 0 -> ML Dev, 1 -> Vertex

GOOGLE_GENAI_USE_VERTEXAI=1


# ML Dev backend config

GOOGLE_API_KEY=YOUR_VALUE_HERE
```

```
# Vertex backend config

# Rag Engine only works with Vertex. So we should configure it to use Vertex:

GOOGLE_CLOUD_PROJECT=YOUR_VALUE_HERE

GOOGLE_CLOUD_LOCATION=YOUR_VALUE_HERE


# Existing corpus in Vertex RAG Engine to be used by RAG agent

# e.g. projects/123/locations/us-central1/ragCorpora/456

RAG_CORPUS=YOUR_VALUE_HERE


# Staging bucket name for ADK agent deployment to Vertex AI Agent Engine (Shall respect this format gs://your-bucket-name)

STAGING_BUCKET=YOUR_VALUE_HERE


# Agent Engine ID in the following format: projects/<PROJECT_NUMBER>/locations/us-central1/reasoningEngines/<AGENT_ENGINE_ID>

AGENT_ENGINE_ID=YOUR_VALUE_HERE




================================================

File: deployment/deploy.py

================================================

# Copyright 2025 Google LLC

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#     http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.


import vertexai

from vertexai import agent_engines

from vertexai.preview.reasoning_engines import AdkApp

from rag.agent import root_agent

import logging

import os

from dotenv import set_key


logging.basicConfig(level=logging.DEBUG)

logger = logging.getLogger(__name__)


GOOGLE_CLOUD_PROJECT = os.getenv("GOOGLE_CLOUD_PROJECT")

GOOGLE_CLOUD_LOCATION = os.getenv("GOOGLE_CLOUD_LOCATION")

STAGING_BUCKET = os.getenv("STAGING_BUCKET")

# Define the path to the .env file relative to this script

ENV_FILE_PATH = os.path.abspath(os.path.join(os.path.dirname(__file__), "..",
".env"))


vertexai.init(

    project=GOOGLE_CLOUD_PROJECT,

    location=GOOGLE_CLOUD_LOCATION,

    staging_bucket=STAGING_BUCKET,

)
```

```
# Function to update the .env file
def update_env_file(agent_engine_id, env_file_path):
    """Updates the .env file with the agent engine ID."""
    try:
        set_key(env_file_path, "AGENT_ENGINE_ID", agent_engine_id)
        print(f"Updated AGENT_ENGINE_ID in {env_file_path} to
{agent_engine_id}")
    except Exception as e:
        print(f"Error updating .env file: {e}")


logger.info("deploying app...")
app = AdkApp(
    agent=root_agent,
    enable_tracing=True,
)


logging.debug("deploying agent to agent engine:")


remote_app = agent_engines.create(
    app,
    requirements=[
        "google-cloud-aiplatform[adk,agent-engines]==1.88.0",
        "google-adk",
        "python-dotenv",
        "google-auth",
        "tqdm",
        "requests",
        "llama-index",
    ],
    extra_packages=[

        "./rag",

    ],

)



# log remote_app

logging.info(f"Deployed agent to Vertex AI Agent Engine successfully, resource
name: {remote_app.resource_name}")



# Update the .env file with the new Agent Engine ID

update_env_file(remote_app.resource_name, ENV_FILE_PATH)




================================================

File: deployment/grant_permissions.sh

================================================

#!/bin/bash

# Copyright 2025 Google LLC

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#     http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and
# limitations under the License.

# Script to grant RAG Corpus access permissions to AI Platform Reasoning Engine Service Agent

set -e

# Load environment variables from .env file
SCRIPT_DIR="$(dirname "$0")"
ENV_FILE="${SCRIPT_DIR}/../.env"
if [ -f "$ENV_FILE" ]; then
  source "$ENV_FILE"
else
  echo "Error: .env file not found at $ENV_FILE"
  exit 1
fi

# Get the project ID from environment variable
PROJECT_ID="$GOOGLE_CLOUD_PROJECT"
if [ -z "$PROJECT_ID" ]; then
  echo "No project ID found. Please set your project ID with 'gcloud config set project YOUR_PROJECT_ID'"
  exit 1
fi

# Get the project number
PROJECT_NUMBER=$(gcloud projects describe "$PROJECT_ID" --format="value(projectNumber)")
if [ -z "$PROJECT_NUMBER" ]; then
```

```
  echo "Failed to retrieve project number for project $PROJECT_ID"
  exit 1
fi

# Define the service account
SERVICE_ACCOUNT="service-${PROJECT_NUMBER}@gcp-sa-aiplatform-re.iam.gserviceaccount.com"

# Get RAG Corpus ID from the RAG_CORPUS environment variable
if [ -z "$RAG_CORPUS" ]; then
  echo "RAG_CORPUS environment variable is not set in the .env file"
  exit 1
fi

# Extract RAG_CORPUS_ID from the full RAG_CORPUS path
RAG_CORPUS_ID=$(echo $RAG_CORPUS | awk -F'/' '{print $NF}')

# Define the RAG Corpus resource
RAG_CORPUS="projects/${PROJECT_NUMBER}/locations/us-central1/ragCorpora/${RAG_CORPUS_ID}"

echo "Granting permissions to $SERVICE_ACCOUNT..."

# Ensure the AI Platform service identity exists
gcloud alpha services identity create --service=aiplatform.googleapis.com --project="$PROJECT_ID"

# Create a custom role with only the RAG Corpus query permission
ROLE_ID="ragCorpusQueryRole"
ROLE_TITLE="RAG Corpus Query Role"
```

```
ROLE_DESCRIPTION="Custom role with permission to query RAG Corpus"

# Check if the custom role already exists
echo "Checking if custom role $ROLE_ID exists..."
if gcloud iam roles describe "$ROLE_ID" --project="$PROJECT_ID" &>/dev/null; then
  echo "Custom role $ROLE_ID already exists."
else
  echo "Custom role $ROLE_ID does not exist. Creating it..."
  gcloud iam roles create "$ROLE_ID" \
    --project="$PROJECT_ID" \
    --title="$ROLE_TITLE" \
    --description="$ROLE_DESCRIPTION" \
    --permissions="aiplatform.ragCorpora.query"
  echo "Custom role $ROLE_ID created successfully."
fi

# Grant the custom role to the service account
echo "Granting custom role for RAG Corpus query permissions for $RAG_CORPUS..."
gcloud projects add-iam-policy-binding "$PROJECT_ID" \
  --member="serviceAccount:$SERVICE_ACCOUNT" \
  --role="projects/$PROJECT_ID/roles/$ROLE_ID" \

echo "Permissions granted successfully."
echo "Service account $SERVICE_ACCOUNT can now query the specific RAG Corpus: $RAG_CORPUS"
```

```
================================================

File: deployment/run.py

================================================

import os

import vertexai

from vertexai import agent_engines

from google.adk.sessions import VertexAiSessionService

from dotenv import load_dotenv

import json


import asyncio


def pretty_print_event(event):

    """Pretty prints an event with truncation for long content."""

    if "content" not in event:

        print(f"[{event.get('author', 'unknown')}]: {event}")

        return


    author = event.get("author", "unknown")

    parts = event["content"].get("parts", [])


    for part in parts:

        if "text" in part:

            text = part["text"]

            # Truncate long text to 200 characters

            if len(text) > 200:

                text = text[:197] + "..."

            print(f"[{author}]: {text}")
        elif "functionCall" in part:
            func_call = part["functionCall"]
            print(f"[{author}]: Function call: {func_call.get('name', 
'unknown')}")
            # Truncate args if too long
            args = json.dumps(func_call.get("args", {}))
            if len(args) > 100:
                args = args[:97] + "..."
            print(f"  Args: {args}")
        elif "functionResponse" in part:
            func_response = part["functionResponse"]
            print(f"[{author}]: Function response: {func_response.get('name', 
'unknown')}")
            # Truncate response if too long
            response = json.dumps(func_response.get("response", {}))
            if len(response) > 100:
                response = response[:97] + "..."
            print(f"  Response: {response}")


load_dotenv()


vertexai.init(
    project=os.getenv("GOOGLE_CLOUD_PROJECT"),
    location=os.getenv("GOOGLE_CLOUD_LOCATION"),
)


session_service = 
VertexAiSessionService(project=os.getenv("GOOGLE_CLOUD_PROJECT"),location=os.getenv("GOOGLE_CL
AGENT_ENGINE_ID = os.getenv("AGENT_ENGINE_ID")
```

```
session = asyncio.run(session_service.create_session(
    app_name=AGENT_ENGINE_ID,
    user_id="123",
))

agent_engine = agent_engines.get(AGENT_ENGINE_ID)

queries = [
    "Hi, how are you?",
    "According to the MD&A, how might the increasing proportion of revenues derived from non-advertising sources like Google Cloud and devices potentially impact Alphabet's overall operating margin, and why?",
    "The report mentions significant investments in AI. What specific connection is drawn between these AI investments and the company's expectations regarding future capital expenditures?",
    "Thanks, I got all the information I need. Goodbye!",
]

for query in queries:
    print(f"\n[user]: {query}")
    for event in agent_engine.stream_query(
        user_id="123",
        session_id=session.id,
        message=query,
    ):
        pretty_print_event(event)
```

==

File: eval/test_eval.py

```
================================================
# Copyright 2025 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import pathlib

import dotenv
import pytest
from google.adk.evaluation.agent_evaluator import AgentEvaluator

pytest_plugins = ("pytest_asyncio",)


@pytest.fixture(scope="session", autouse=True)
def load_env():
    dotenv.load_dotenv()
```

```
@pytest.mark.asyncio
async def test_eval_full_conversation():
    """Test the agent's basic ability on a few examples."""
    await AgentEvaluator.evaluate(
        agent_module="rag",
        eval_dataset_file_path_or_dir=str(
            pathlib.Path(__file__).parent / "data/conversation.test.json"
        ),
        num_runs=1,
    )



================================================
File: eval/data/conversation.test.json
================================================
[
  {
    "query": "Hi there, I have some questions about the Alphabet 10-K report.",
    "expected_tool_use": [],
    "reference": "Hello! I can help with that. What would you like to know based on the Alphabet 10-K for the fiscal year ended December 31, 2024?"
  },
  {
    "query": "According to the MD&A, how might the increasing proportion of revenues derived from non-advertising sources like Google Cloud and devices potentially impact Alphabet's overall operating margin, and why?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "How does the growth of non-advertising revenue affect Alphabet's margins?"
        }
      }
    ],
    "reference": "The report states that revenues from cloud, consumer subscriptions, platforms, and devices are increasing. It notes that the margins on these revenues vary significantly and are generally lower than advertising margins. Specifically, device sales adversely affect consolidated margins due to pricing pressure and higher cost of sales. Therefore, this trend may exert downward pressure on the overall operating margin. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7]"
  },
  {
    "query": "The report mentions significant investments in AI. What specific connection is drawn between these AI investments and the company's expectations regarding future capital expenditures?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Relationship between Alphabet's AI strategy and capital expenditure forecasts"
        }
      }
    ],
    "reference": "The report explicitly states an expectation that capital expenditures will increase, including expected spend and increased technical infrastructure investment, *in particular* to support AI products and services. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7 & Note About Forward-Looking Statements]"
  },
  {
    "query": "Beyond basic functionality issues, what specific legal, ethical, 
or societal challenges does Alphabet identify in its Risk Factors as 
potentially arising from the development and use of AI?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "What ethical and legal risks does Alphabet associate with 
artificial intelligence?"

        }

      }

    ],

    "reference": "The risks mentioned include harmful content, inaccuracies, 
discrimination, intellectual property infringement/misappropriation, violation 
of rights of publicity, defamation, data privacy, cybersecurity issues, and 
unintended consequences or uses potentially affecting human rights, privacy, 
employment, or other social concerns. [Citation: Based on Alphabet 10-K for 
FYE Dec 31, 2024, Item 1A]"

  },

  {

    "query": "The report details several ongoing antitrust investigations and 
lawsuits. Synthesizing the information provided, what common themes or 
business areas appear to be the primary focus of regulatory scrutiny?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "What areas of Alphabet's business face the most antitrust 
scrutiny?"

        }

      }

    ],

    "reference": "Common themes of regulatory scrutiny focus on Search and 
Search advertising distribution/practices, Advertising Technology (ad tech 
stack), the Android operating system distribution agreements, and Google Play 
store operations (billing, fees, business model). [Citation: Based on Alphabet
```

```
10-K for FYE Dec 31, 2024, Item 1A & Note 10]"
  },
  {
    "query": "How does Alphabet account for its investments in non-marketable equity securities, and what specific events trigger adjustments to their carrying value according to Note 1?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Alphabet accounting policy for non-marketable equity securities"
        }
      }
    ],
    "reference": "They are primarily accounted for under the measurement alternative (cost, less impairment, +/- adjustments). Adjustments to carrying value are triggered by observable price changes in orderly transactions for identical or similar investments of the same issuer, or upon impairment identification. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Note 1 & Note 3]"
  },
  {
    "query": "While Google Cloud's operating income turned positive and grew significantly in 2024, what factors mentioned in the MD&A partially offset the revenue growth and moderated the profitability increase?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Factors limiting Google Cloud profitability growth in 2024"
        }
      }
    ],
    "reference": "The increase in operating income was partially offset by increases in usage costs for technical infrastructure as well as increases in employee compensation expenses, largely driven by headcount growth. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7, Segment Profitability]"
  },
  {
    "query": "Explain the primary reasons for the significant increase in Alphabet's 'Other income (expense), net' from $1.4 billion in 2023 to $7.4 billion in 2024.",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Drivers of the change in Alphabet's Other Income (Expense), net in 2024"
        }
      }
    ],
    "reference": "The increase was primarily due to an increase in net gains on equity securities (mainly unrealized gains on non-marketable equity securities driven by fair value adjustments) and a decrease in net losses on foreign currency exchange compared to the prior year. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7, Other Income (Expense), Net & Note 7]"
  },
  {
    "query": "What key factors contributed to the increase in Alphabet's effective tax rate from 13.9% in 2023 to 16.4% in 2024?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Why did Alphabet's effective tax rate increase in 2024?"
        }
      }
    ],
    "reference": "The increase was primarily due to a 2023 one-time adjustment for IRS tax rule changes related to foreign tax credits and R&D capitalization (which lowered the 2023 rate), plus a decrease in the 2024 U.S. federal Foreign Derived Intangible Income deduction. This was partially offset by higher stock-based compensation tax benefits in 2024. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7, Provision for Income Taxes & Note 14]"
  },
  {
    "query": "Regarding the DOJ antitrust lawsuit focused on Search, what was the court's ruling in August 2024, and what are the subsequent steps outlined in the report?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Status and ruling in the DOJ Search antitrust case against Google"
        }
      }
    ],
    "reference": "The U.S. District Court for the District of Columbia ruled that Google violated antitrust laws related to Search and Search advertising. A separate proceeding is being held to determine remedies. Alphabet intends to appeal after the remedy decision. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 1A & Note 10]"
  },
  {
    "query": "What potential challenges does Alphabet acknowledge regarding achieving its ambitious 2030 sustainability goals (net-zero and 24/7 CFE)?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "What uncertainties or difficulties does Alphabet foresee
in meeting its climate goals?"
        }
      }
    ],
    "reference": "Challenges include the path not being easy or linear, long
lead times for large-scale infrastructure, significant uncertainty (e.g.,
future environmental impact of AI), dependency on new technology development
(by Alphabet and the energy sector), and the expectation that total GHG
emissions might rise before dropping toward the target. [Citation: Based on
Alphabet 10-K for FYE Dec 31, 2024, Item 1]"
  },
  {
    "query": "How might Alphabet's share repurchase program, as mentioned in
the risk factors, potentially influence the relative voting control held by
its founders, Larry Page and Sergey Brin?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Impact of Alphabet share buybacks on founder voting power"
        }
      }
    ],
    "reference": "The risk factor on concentration of ownership explicitly
states that share repurchases 'may also affect Larry and Sergey’s relative
voting power.' By repurchasing lower-vote Class A or non-voting Class C
shares, the proportional voting power of their high-vote Class B shares
increases relative to other classes. [Citation: Based on Alphabet 10-K for FYE
Dec 31, 2024, Item 1A]"
  },
  {
    "query": "What accounting estimates does Alphabet identify as 'critical'
due to the significant level of uncertainty and potential material impact on
financial results?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "What are identified as Alphabet's critical accounting
estimates?"
        }
      }
    ],
    "reference": "The critical accounting estimates identified are: Fair Value
Measurements of Non-Marketable Equity Securities; Property and Equipment
(useful lives assessment); Income Taxes (evaluation of uncertain tax
positions); and Loss Contingencies (likelihood and estimation of loss).
[Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7]"
  },
  {
    "query": "Describe Alphabet's strategy for using derivative financial
instruments, including the types of risks hedged and the accounting
designations applied.",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Describe Alphabet's derivative hedging strategy and
accounting"
        }
      }
    ],
    "reference": "Alphabet uses derivatives, primarily foreign currency
contracts, to manage foreign exchange risk. They designate contracts as Cash
Flow Hedges for forecasted revenue, Fair Value Hedges for certain foreign
currency marketable securities, and Net Investment Hedges for investments in
foreign subsidiaries. Non-designated derivatives are used for other exposures
like intercompany transactions and for investment purposes. [Citation: Based
on Alphabet 10-K for FYE Dec 31, 2024, Note 3 & Item 7A]"
  },
  {

    "query": "What constitutes the majority of Alphabet's $93.2 billion
revenue backlog as of year-end 2024, and what is the anticipated timeframe for
recognizing roughly half of this amount?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "Details on Alphabet's revenue backlog composition and
recognition period"

        }

      }

    ],

    "reference": "The revenue backlog primarily relates to Google Cloud
commitments for future services not yet recognized as revenue. Approximately
half of this backlog is expected to be recognized as revenue over the next 24
months. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Note 2]"

  },

  {

    "query": "Analyze the change in Google Services operating income from 2023
to 2024. What were the primary positive drivers, and what costs partially
offset the increase?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "Analysis of Google Services operating income change in
2024"

        }

      }

    ],

    "reference": "Google Services operating income increased by $25.4 billion.
This was primarily driven by an increase in revenues. Factors partially
offsetting this included increases in content acquisition costs and Traffic
```

```
Acquisition Costs (TAC). A reduction in employee compensation expenses also contributed positively to the income increase. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 7, Segment Profitability]"

  },

  {

    "query": "The report mentions 'Alphabet-level activities' in its segment reporting. What types of costs are typically included in this category and not allocated to the main segments?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "What costs are included in Alphabet-level activities?"

        }

      }

    ],

    "reference": "Costs included in Alphabet-level activities primarily consist of certain AI-focused shared R&D (like general AI model development), corporate initiatives (e.g., philanthropy), corporate shared costs (some finance, HR, legal functions, certain fines/settlements), charges related to employee severance and office space optimization (during 2023/2024), and hedging gains/losses related to revenue. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Note 15 & Item 7]"

  },

  {

    "query": "Contrast the year-over-year percentage change in 'Paid clicks' versus 'Cost-per-click' for Google Search & other in 2024. What does this suggest about monetization trends in that area?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "Comparison of paid click and cost-per-click growth for Google Search in 2024"

        }
      }

    ],

    "reference": "For Google Search & other in 2024 vs 2023, Paid clicks
increased by 5%, while Cost-per-click increased by 7%. This suggests that the
average revenue generated per click grew slightly faster than the volume of
clicks for these properties during the year. [Citation: Based on Alphabet 10-K
for FYE Dec 31, 2024, Item 7, Monetization Metrics]"

  },

  {

    "query": "What is the purpose of the 'measurement alternative' accounting
method used for certain non-marketable equity securities, and how does it
differ from marking-to-market based purely on estimated fair value each
period?",

    "expected_tool_use": [

      {

        "tool_name": "retrieve_rag_documentation",

        "tool_input": {

          "query": "Explain the 'measurement alternative' accounting for
equity securities"

        }

      }

    ],

    "reference": "The measurement alternative measures these securities at
cost less impairment, plus or minus adjustments resulting *only* from
observable price changes in orderly transactions for identical or similar
investments of the same issuer. This differs from marking-to-market based on
estimated fair value each period (like public stocks), as adjustments under
the measurement alternative only occur upon specific observable events or
impairment, reducing period-to-period volatility unless such events occur.
[Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Note 1 & Note 3]"

  },

  {

    "query": "According to the discussion on competition, what types of
companies or services represent competitive threats to Google's general-
purpose search engine beyond just other search engines?",

    "expected_tool_use": [

      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "What types of competitors does Google Search face besides other search engines?"
        }
      }
    ],
    "reference": "Competitive threats mentioned include vertical search engines and e-commerce providers (where users navigate directly), online advertising platforms and networks (including online shopping and streaming services), social networks (used for product/service referrals), and digital assistant providers. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Item 1]"
  },
  {
    "query": "How does Alphabet assess the recoverability of its long-lived assets like property & equipment and operating lease assets, and what happens if an asset is deemed impaired?",
    "expected_tool_use": [
      {
        "tool_name": "retrieve_rag_documentation",
        "tool_input": {
          "query": "Alphabet's process for testing long-lived assets for impairment"
        }
      }
    ],
    "reference": "Alphabet reviews long-lived assets for impairment when events indicate the carrying amount may not be recoverable. Recoverability is assessed by comparing the carrying amount to the *future undiscounted cash flows* the asset or asset group is expected to generate. If the carrying value is not recoverable, an impairment loss is recognized, measured as the amount by which the carrying value exceeds the asset's *fair value*. [Citation: Based on Alphabet 10-K for FYE Dec 31, 2024, Note 1]"
  },
  {
    "query": "That covers my questions for now. Thanks!",

    "expected_tool_use": [],

    "reference": "You're welcome! Glad I could help. Feel free to ask if
anything else comes up."

  }

]
```

```
================================================

File: eval/data/test_config.json

================================================

{

  "criteria": {

    "tool_trajectory_avg_score": 0.09,

    "response_match_score": 0.4

  }

}
```

```
================================================

File: rag/__init__.py

================================================

# Copyright 2025 Google LLC

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#
```

```
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from . import agent



================================================
File: rag/agent.py
================================================
# Copyright 2025 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os
```

```
from google.adk.agents import Agent

from google.adk.tools.retrieval.vertex_ai_rag_retrieval import
VertexAiRagRetrieval

from vertexai.preview import rag



from dotenv import load_dotenv

from .prompts import return_instructions_root



load_dotenv()



ask_vertex_retrieval = VertexAiRagRetrieval(

    name='retrieve_rag_documentation',

    description=(

        'Use this tool to retrieve documentation and reference materials for
the question from the RAG corpus,'

    ),

    rag_resources=[

        rag.RagResource(

            # please fill in your own rag corpus

            # here is a sample rag corpus for testing purpose

            # e.g. projects/123/locations/us-central1/ragCorpora/456

            rag_corpus=os.environ.get("RAG_CORPUS")

        )

    ],

    similarity_top_k=10,

    vector_distance_threshold=0.6,

)
```

```
root_agent = Agent(
    model='gemini-2.0-flash-001',
    name='ask_rag_agent',
    instruction=return_instructions_root(),
    tools=[
        ask_vertex_retrieval,
    ]
)



================================================
File: rag/prompts.py
================================================
# Copyright 2025 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Module for storing and retrieving agent instructions.
```

```
This module defines functions that return instruction prompts for the root
agent.

These instructions guide the agent's behavior, workflow, and tool usage.

"""




def return_instructions_root() -> str:



    instruction_prompt_v1 = """

        You are an AI assistant with access to specialized corpus of
documents.

        Your role is to provide accurate and concise answers to questions
based

        on documents that are retrievable using ask_vertex_retrieval. If you
believe

        the user is just chatting and having casual conversation, don't use
the retrieval tool.



        But if the user is asking a specific question about a knowledge they
expect you to have,

        you can use the retrieval tool to fetch the most relevant information.



        If you are not certain about the user intent, make sure to ask
clarifying questions

        before answering. Once you have the information you need, you can use
the retrieval tool

        If you cannot provide an answer, clearly explain why.



        Do not answer questions that are not related to the corpus.

        When crafting your answer, you may use the retrieval tool to fetch
details

        from the corpus. Make sure to cite the source of the information.
        Citation Format Instructions:

        When you provide an answer, you must also add one or more citations **at the end** of
        your answer. If your answer is derived from only one retrieved chunk,
        include exactly one citation. If your answer uses multiple chunks
        from different files, provide multiple citations. If two or more
        chunks came from the same file, cite that file only once.

        **How to cite:**
        - Use the retrieved chunk's `title` to reconstruct the reference.
        - Include the document title and section if available.
        - For web resources, include the full URL when available.

        Format the citations at the end of your answer under a heading like
        "Citations" or "References." For example:
        "Citations:
        1) RAG Guide: Implementation Best Practices
        2) Advanced Retrieval Techniques: Vector Search Methods"

        Do not reveal your internal chain-of-thought or how you used the chunks.
        Simply provide concise and factual answers, and then list the
        relevant citation(s) at the end. If you are not certain or the
        information is not available, clearly state that you do not have
        enough information.
        """
```

```
    instruction_prompt_v0 = """

        You are a Documentation Assistant. Your role is to provide accurate
and concise

        answers to questions based on documents that are retrievable using
ask_vertex_retrieval. If you believe

        the user is just discussing, don't use the retrieval tool. But if the
user is asking a question and you are

        uncertain about a query, ask clarifying questions; if you cannot

        provide an answer, clearly explain why.



        When crafting your answer,

        you may use the retrieval tool to fetch code references or additional

        details. Citation Format Instructions:



        When you provide an

        answer, you must also add one or more citations **at the end** of

        your answer. If your answer is derived from only one retrieved chunk,

        include exactly one citation. If your answer uses multiple chunks

        from different files, provide multiple citations. If two or more

        chunks came from the same file, cite that file only once.



        **How to

        cite:**

        - Use the retrieved chunk's `title` to reconstruct the

        reference.

        - Include the document title and section if available.

        - For web resources, include the full URL when available.



        Format the citations at the end of your answer under a heading like

        "Citations" or "References." For example:
```

```
        "Citations:
        1) RAG Guide: Implementation Best Practices
        2) Advanced Retrieval Techniques: Vector Search Methods"

        Do not
        reveal your internal chain-of-thought or how you used the chunks.
        Simply provide concise and factual answers, and then list the
        relevant citation(s) at the end. If you are not certain or the
        information is not available, clearly state that you do not have
        enough information.
        """

    return instruction_prompt_v1
```

==

File: rag/shared_libraries/prepare_corpus_and_data.py

==

```
# Copyright 2025 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.


from google.auth import default

import vertexai

from vertexai.preview import rag

import os

from dotenv import load_dotenv, set_key

import requests

import tempfile


# Load environment variables from .env file

load_dotenv()


# --- Please fill in your configurations ---

# Retrieve the PROJECT_ID from the environmental variables.

PROJECT_ID = os.getenv("GOOGLE_CLOUD_PROJECT")

if not PROJECT_ID:

    raise ValueError(

        "GOOGLE_CLOUD_PROJECT environment variable not set. Please set it in
your .env file."

    )

LOCATION = os.getenv("GOOGLE_CLOUD_LOCATION")

if not LOCATION:

    raise ValueError(

        "GOOGLE_CLOUD_LOCATION environment variable not set. Please set it in
your .env file."

    )
```

```
CORPUS_DISPLAY_NAME = "Alphabet_10K_2024_corpus"

CORPUS_DESCRIPTION = "Corpus containing Alphabet's 10-K 2024 document"

PDF_URL = "https://abc.xyz/assets/77/51/9841ad5c4fbe85b4440c47a4df8d/goog-10-k-2024.pdf"

PDF_FILENAME = "goog-10-k-2024.pdf"

ENV_FILE_PATH = os.path.abspath(os.path.join(os.path.dirname(__file__), "..", "..", ".env"))



# --- Start of the script ---

def initialize_vertex_ai():

  credentials, _ = default()

  vertexai.init(

      project=PROJECT_ID, location=LOCATION, credentials=credentials

  )



def create_or_get_corpus():

  """Creates a new corpus or retrieves an existing one."""

  embedding_model_config = rag.EmbeddingModelConfig(

      publisher_model="publishers/google/models/text-embedding-004"

  )

  existing_corpora = rag.list_corpora()

  corpus = None

  for existing_corpus in existing_corpora:

    if existing_corpus.display_name == CORPUS_DISPLAY_NAME:

      corpus = existing_corpus

      print(f"Found existing corpus with display name '{CORPUS_DISPLAY_NAME}'")

      break
  if corpus is None:
    corpus = rag.create_corpus(
        display_name=CORPUS_DISPLAY_NAME,
        description=CORPUS_DESCRIPTION,
        embedding_model_config=embedding_model_config,
    )
    print(f"Created new corpus with display name '{CORPUS_DISPLAY_NAME}'")
  return corpus


def download_pdf_from_url(url, output_path):
  """Downloads a PDF file from the specified URL."""
  print(f"Downloading PDF from {url}...")
  response = requests.get(url, stream=True)
  response.raise_for_status()  # Raise an exception for HTTP errors

  with open(output_path, 'wb') as f:
    for chunk in response.iter_content(chunk_size=8192):
      f.write(chunk)

  print(f"PDF downloaded successfully to {output_path}")
  return output_path


def upload_pdf_to_corpus(corpus_name, pdf_path, display_name, description):
  """Uploads a PDF file to the specified corpus."""
  print(f"Uploading {display_name} to corpus...")
  try:
    rag_file = rag.upload_file(
        corpus_name=corpus_name,
        path=pdf_path,
        display_name=display_name,
        description=description,
    )
    print(f"Successfully uploaded {display_name} to corpus")
    return rag_file
  except Exception as e:
    print(f"Error uploading file {display_name}: {e}")
    return None

def update_env_file(corpus_name, env_file_path):
    """Updates the .env file with the corpus name."""
    try:
        set_key(env_file_path, "RAG_CORPUS", corpus_name)
        print(f"Updated RAG_CORPUS in {env_file_path} to {corpus_name}")
    except Exception as e:
        print(f"Error updating .env file: {e}")

def list_corpus_files(corpus_name):
  """Lists files in the specified corpus."""
  files = list(rag.list_files(corpus_name=corpus_name))
  print(f"Total files in corpus: {len(files)}")
  for file in files:
    print(f"File: {file.display_name} - {file.name}")


def main():
  initialize_vertex_ai()
  corpus = create_or_get_corpus()

  # Update the .env file with the corpus name
  update_env_file(corpus.name, ENV_FILE_PATH)

  # Create a temporary directory to store the downloaded PDF
  with tempfile.TemporaryDirectory() as temp_dir:
    pdf_path = os.path.join(temp_dir, PDF_FILENAME)

    # Download the PDF from the URL
    download_pdf_from_url(PDF_URL, pdf_path)

    # Upload the PDF to the corpus
    upload_pdf_to_corpus(
        corpus_name=corpus.name,
        pdf_path=pdf_path,
        display_name=PDF_FILENAME,
        description="Alphabet's 10-K 2024 document"
    )

  # List all files in the corpus
  list_corpus_files(corpus_name=corpus.name)

if __name__ == "__main__":
  main()
```

```
================================================
File: tests/__init__.py
================================================
# Copyright 2025 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```