Quick tip: Review the prerequisites before you run the lab

End Lab 00:41:55

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Open Google Cloud Console

Username

student-00-29f5a342ae71

Password

vNF2B6Uvavqb

GCP Project ID

qwiklabs-gcp-04-f4e926e3

Measure Gen AI performance with the Generative AI Evaluation Service

Lab 1 hour No cost Intermediate

★★★★½

This lab may incorporate AI tools to support your learning.

Previous Next

Overview

In this lab, you'll learn about using Vertex AI's Generative AI Evaluation Service to measure a generative AI-specific metric on a generative AI application.

This API is designed for you to score your application's responses on a variety of metrics, either in comparison to ground-truth data or by using another model to score your app's output. In this way, you can measure your application's performance on multiple metrics, quantifying improvements made via prompt engineering, tuning a model, or other adjustments.

Objectives

In this lab you'll learn to:

- Understand pointwise vs. pairwise evaluation paradigms
- Understand different evaluation metrics available to you

Setup and requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

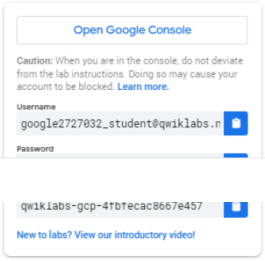
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

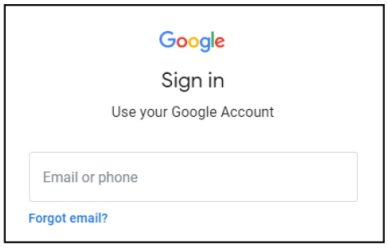
Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

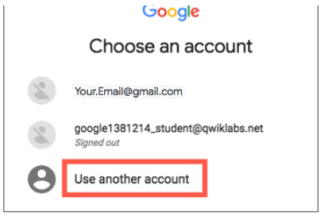


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a

After a few moments, the Cloud Console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.

Understanding evaluation paradigms

Evaluating generated text isn't simple. If a group of people were given two articles about how mountain ranges form and were asked to score the articles, some people might prefer version A and others version B. Nevertheless, when working with machine learning models, engineers value measuring performance and incremental improvements.

The Generative AI Evaluation Service evaluates Generative AI applications. You can say "applications" rather than "models" because the evaluations work in responses, which

information passed into the model as context.

The Generative AI Evaluation Service describes its offerings as belonging to one of two paradigms: pointwise and pairwise.

- Pointwise evaluations evaluate a generative AI application by scoring it on selected metrics. Improvements to the model or prompt should result in better scores on

these metrics.

- Pairwise evaluations don't offer objective scores, but instead choose a winner from the outputs of two generative AI applications. By choosing a preferred output for many different prompts, we can see which application performs better on that range of tasks.

In addition to understanding the paradigms above, it's important to understand two different categories of available metrics:

- Computation-based metrics compare generated output to a ground-truth reference that is considered the ideal response.
- Model-based metrics don't compare outputs to a ground truth, but instead use a

The Gen AI Evaluation Service API allows you to use a mix of computation-based and model-based metrics. You can use these metrics to guide your prompt engineering improvements by iterating rapidly and evaluating on a small batch of prompts (and their ideal responses, if using computation-based metrics).

In this lab, you'll focus on using the Gen AI Evaluation Service API to measure improvements to an application's prompt.

Task 1. Initialize Vertex AI in a Colab Enterprise notebook

1. Navigate to **Colab Enterprise** by searching for it using the search bar located at the top of the console.
2. If prompted, enable required APIs.
3. Select the `us-central1` region from the **Region** drop-down menu, if it is not already selected, and click the **+** button to create a new notebook.
4. Rename the notebook to `cymbal-notebook-eval`.
5. Paste the following code into the top cell and run it with **Shift + Return**. If you don't already have an active notebook runtime, running a cell in a Colab Enterprise notebook will trigger it to create one for you and connect the notebook to it. When a runtime is allocated for the first time, you may be presented with a pop-up window to authorize the environment to act as your Qwiklabs student account.

```
%pip install --upgrade --quiet google-cloud-aiplatform google-cloud-aiplatform[evaluation]
```

6. After the cell completes running, indicated by a checkmark to the left of the cell, the packages should be installed. To use them, we'll restart the runtime. Click on the downward-pointing caret (an icon like an arrow) in the upper right of the notebook:

7. Clicking the caret should have revealed a set of menus above the notebook (File, Edit, View, etc.). Select **Runtime** > **Restart Session** > **select Yes**. The runtime will restart, indicated by clearing the green checkmark and the cell run order integer next to the cell you ran above.
8. Click **+ Code** to create a new code cell and paste in the import & configuration code below. Press **Shift + Return** to run the cell.

```
import datetime
import nest_asyncio
import pandas as pd
from IPython.display import display, Markdown, HTML

import vertexai
from vertexai.generative_models import GenerativeModel

pd.set_option('display.max_colwidth', None)
```

9. Paste the following into a new code block & run it to initialize Vertex AI.

```
PROJECT_ID = "qwiklabs-gcp-04-f4e926e31c52"
LOCATION = "us-central1"
import vertexai
vertexai.init(project=PROJECT_ID, location=LOCATION)
```

Click **Check my progress** to verify the objectives.

Initialize Vertex AI in a Colab Enterprise notebook

Check my progress

Assessment Completed!

Task 2. Explore example data and use it to generate content

1. Run this line in a new code cell (by pasting the code and hitting **Shift + Return**) to download some example data.

```
!gcloud storage cp gs://partner-genai-bucket/genai065/apartment_table.csv .
```

2. Load it into a Pandas DataFrame, and view the first few rows:

```
apartment_df = pd.read_csv("apartment_table.csv")
apartment_df.head()
```

You should see a table like this, presenting data about apartments in New York City, including how many bedrooms they have, how large they are (measured in square feet, identified in this dataset as "Sqft"), whether the building has an elevator, and other attributes of an apartment.

	Address	Unit	Sqft	Bedrooms	Elevator	Washer & Dryer in Unit	Pets Allowed	Notable features
0	123 West 14th Street, New York, NY 10014	2E	550	2	yes	no	yes	doorman, pool in the building, shared roof dec...
	NY 10003							city views
3	1011 5th Avenue, New York, NY 10028	30	1024	2	no	no	yes	great view of Central Park, high ceilings
4	2222 Park Avenue, New York, NY 10017	4F	1234	1	no	yes	no	right next to soccer fields at the park

Notice in particular the "Notable features" column, in which **unstructured notes** from a real estate agent have been captured to describe other selling points of each apartment. You will have a model generate apartment listing descriptions based on these structured and unstructured fields, and then you will measure how well the model sticks to the information in these fields when generating listing descriptions.

3. Restructure the data into a list of records with keys identified for each record, and view the first record.

```
apartment_records = apartment_df.to_dict(orient='records')
apartment_records[0]
```

The above cell should produce a record like this:

```
{'Address': '123 West 14th Street, New York, NY 10014',
 'Bedrooms': 2,
 'Elevator': 'yes',
 'Washer & Dryer in Unit': 'no',
 'Pets Allowed': 'yes',
 'Notable features': 'doorman, pool in the building, shared roof deck with grills'}
```

4. Now it's time to generate some text. Instantiate a generative model, define a prompt with some instructions for it, and generate content based on the example data:

```
model = GenerativeModel(
  "gemini-pro",
  generation_config={
      "temperature": 0,
      "top_p": 0.4,
  },
)

prompt = "Write a one paragraph apartment listing to promote this

# View the response using Markdown to format it nicely for
```

```
notebook viewing
Markdown(model.generate_content(prompt +
str(apartment_records[0])).text)
```

The model will have generated some text like what you see below. You can compare the content to the record dictionary you saw above to see that features like the number of bedrooms and the square footage are included. Multiple details from the unstructured notes (the doorman, the pool, the grills on the roof deck) are also included.

```
Live the high life in this stunning 2-bedroom apartment at 123 West 14th
Street! This spacious 550 sq ft unit boasts an abundance of natural
light and modern finishes. Enjoy the convenience of an elevator building
and the luxury of a doorman. Take a dip in the building's sparkling pool
or soak up the sun on the shared roof deck with grills. This pet-
friendly apartment is your oasis in the heart of the city. Don't miss
out on this incredible opportunity!
```

You might need to take caution, however, as the description may also include some information that we didn't provide. In the output above, the model suggests that there is

You'll set up an evaluation to attempt to quantify how **grounded**, or using only provided information, are the model's responses.

Click **Check my progress** to verify the objectives.

Load data and use it to generate content

Check my progress

Assessment Completed!

Task 3. Configure and trigger a model-based evaluation

Select the Pointwise text use case template for "Groundedness" to read the prompt.

2. Notice that the model will score a response with a 1 if it appears to be fully grounded, or a 0 otherwise.

   Also notice that the prompt contains a couple of placeholders in curly braces: `{prompt}` and `{response}`. You must provide at least the `prompt` data in your evaluation dataset, but you can choose whether to provide the `response` data yourself or let the Evaluation Task run the prompts through a model you provide to generate the responses before evaluating them.

   Additionally, pairwise metrics (which compare two models) require the response of the other model you are comparing against, which must be provided in a field named `baseline_model_response`. For chat evaluations, the chat history will also need to be provided as `history`.

3. To create an evaluation dataset, create the `prompt` for each example, which will consist of the prompt instructions you defined earlier and the context data for each apartment.

```
# model, usually specific to a given query or example,
# that it needs to fulfill your instructions.
# In this case, the context is each apartment record.
contexts = [str(record) for record in apartment_records]
# The full prompt combines the prompt instructions you
# created earlier with the context for each apartment.
full_prompts = [prompt + str(record) for record in
apartment_records]

print(full_prompts[0])
```

4. Instead of generating the responses, yourself, create an evaluation dataset of just the prompts, and the evaluation service will generate responses for you as part of the evaluation task.

Note: Because of Qwiklabs quota limitations, you will limit your evaluation dataset to 5 examples. But the best practice recommendation would be to include around 100 examples covering the types of inputs your model might see.

```
})
```

5. Now you'll explore some of the classes available for evaluation. Run the following imports and print the list of available MetricPromptTemplateExamples:

```
from vertexai.evaluation import (
    MetricPromptTemplateExamples,
    EvalTask,
    PairwiseMetric,
    PairwiseMetricPromptTemplate,
    PointwiseMetric,
    PointwiseMetricPromptTemplate,
)

MetricPromptTemplateExamples.list_example_metric_names()
```

You'll see an output like:

```
['coherence',
 'groundedness',
 'instruction_following',
 'verbosity',
 'text_quality',
  ...]
```

6. Instead of using the documentation to review the prompt as you did earlier, you can view the criteria and ratings for each metric within your code like so:

```
print(MetricPromptTemplateExamples.get_prompt_template('groundednes
```

7. Instantiate an EvalTask by associating your dataset & selected metric. You can also provide an experiment name to track your evaluations in Vertex AI Experiments.

```
eval_task = EvalTask(
  dataset=eval_dataset,
  metrics=[MetricPromptTemplateExamples.Pointwise.GROUNDEDNESS],
```

8. Run the `evaluate()` method on the task by giving it a unique run name. By passing the model you would like to evaluate, the EvalTask can generate the responses needed to complete the evaluation dataset.

```
run_ts = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
eval_result = eval_task.evaluate(
  model=model,
  experiment_run_name=f"apt-gen-{run_ts}"
  )

# You might want to keep track of your results in a list
# which you will use to plot your results later on
eval_results_to_compare = []
eval_results_to_compare.append(eval_result)
```

9. The evaluation service's results consist of three parts: a complete table of results (`metrics_table`), some summary metrics with averages and standard deviation of the metrics (`summary_metrics`), and some `metadata` related to the run. First look at the summary metrics:

```
eval_result.summary_metrics
```

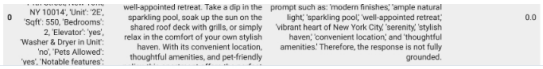
Viewing the outputs, you'll see that the groundedness was likely not perfect. From the Groundedness prompt descriptions we saw above, we know the model scores a 1 for grounded or a 0 for grounded, so a perfect result would have returned a mean of 1.0 and a std (or standard deviation) of 0 suggesting no deviation from that mean.

10. Next, view the complete results table:

```
eval_result.metrics_table
```

```
eval_result.metrics_table
```

	prompt	response	groundedness/explanation	groundedness/score
	Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {Address: '123 West 14th Street, New York	**Step into luxury living at 123 West 14th Street, Apartment 2E!** This stunning 550-square-foot 2-bedroom oasis boasts modern finishes, ample natural light, and a doorman for added security. Immerse yourself in the vibrant heart of New York City while enjoying the serenity of your	Although the response appropriately incorporates all details from the input, it introduces external details not found in the	



0	NY 10014', 'Unit': '2E', 'Sqft': 550, 'Bedrooms': 2, 'Elevator': 'yes', 'Washer & Dryer in Unit': 'no', 'Pets Allowed': 'yes', 'Notable features':	well-appointed retreat. Take a dip in the sparkling pool, soak up the sun on the shared roof deck with grills, or simply relax in the comfort of your own stylish haven. With its convenient location, thoughtful amenities, and pet-friendly	prompt such as: 'modern finishes,' 'ample natural light,' 'sparkling pool,' 'well-appointed retreat,' 'vibrant heart of New York City,' 'serenity,' 'stylish haven,' 'convenient location,' and 'thoughtful amenities.' Therefore, the response is not fully grounded.	0.0

You should see a table like this containing your results. Look for examples where the `groundedness/score` is 0.0, meaning the results were not grounded. Read the `response` and then the `groundedness/explanation` to see why the model rated the response as not grounded. You may see something like:

```
Although the response appropriately incorporates all details from the
input, it introduces external details not found in the prompt such as:
'modern finishes,' 'ample natural light,' 'sparkling pool,' 'well-
appointed retreat,' 'vibrant heart of New York City,' 'serenity,'
'stylish haven,' 'convenient location,' and 'thoughtful amenities.'
Therefore, the response is not fully grounded.
```

Click **Check my progress** to verify the objectives.

Configure & trigger an initial evaluation.

Check my progress

Assessment Completed!

Task 4. Improve the prompt to see scores improve

In this task, you'll make a tweak to address the hallucination concerns and improve your application's groundedness score.

1. Improve the prompt to improve your groundedness score. With the new prompt, also create a new evaluation dataset.

```
# This is a minor update, but should stop the model from
# inventing as many details about each apartment.
updated_prompt = "Write a one paragraph apartment listing
highlighting the best known features of this apartment. Use only
the details included in the following information: "
```

```
updated_eval_dataset = pd.DataFrame(
  {
      "prompt": updated_full_prompts[0:5]
  }
)
```

2. Create a new EvalTask and run its `evaluate()` method to generate evaluations. Preview them in a table.

```
updated_eval_task = EvalTask(
  dataset=updated_eval_dataset,
  metrics=[MetricPromptTemplateExamples.Pointwise.GROUNDEDNESS],
  experiment="apartment-listing-generation",
)

run_ts = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
updated_result = updated_eval_task.evaluate(
  model=model,
  experiment_run_name=f"apt-gen-{run_ts}"
```

```
# Append the new result to your results
eval_results_to_compare.append(updated_result)

# Preview the summary
print(updated_result.summary_metrics)
```

Your `groundedness/mean` score should now have increased.

3. Use the provided helper function to visualize your evaluation runs against each other. See whether your application improved on the metrics:

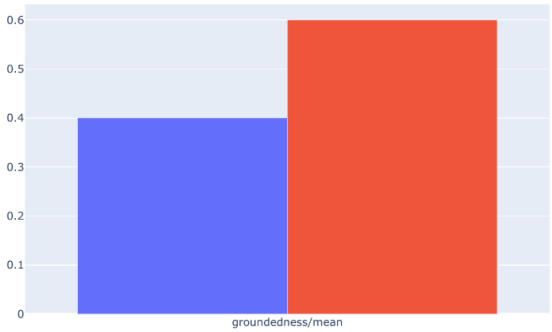
```
import plotly.graph_objects as go
def plot_bar_plot(eval_results, metrics=None):
  fig = go.Figure()
  data = []
```

```
    for eval_result in eval_results:
        summary_metrics = eval_result.summary_metrics
        if metrics:
            summary_metrics = {
                k: summary_metrics[k]
                for k, v in summary_metrics.items()
                }


        data.append(
            go.Bar(
                x=list(summary_metrics.keys()),
                y=list(summary_metrics.values()),
                name=eval_result.metadata["experiment_run"]
            )
        )
    fig = go.Figure(data=data)


    # Change the bar mode
    fig.update_layout(barmode="group")
    fig.show()


plot_bar_plot(eval_results_to_compare, metrics=
["groundedness/mean"])
```



5. Within the **Google Cloud Console**, navigate to **Vertex AI** and then Experiments within the left navigation bar. If you click on your experiment name and select a run name, you can select the **Metrics** and **Parameters** tabs to see that the Generative AI Evaluation Service has documented your runs.

Improve the prompt to see scores improve

Check my progress

Assessment Completed!

Congratulations!

You've successfully structured an evaluation dataset for the Generative AI Evaluation Service according to a particular metric and used the service to quickly evaluate how your Gen AI application performs across multiple prompt examples.

As a next step, you may want to explore adjusting a prompt template to your use case.

Manual Last September 4, 2024

Lab Last Tested September 4, 2024