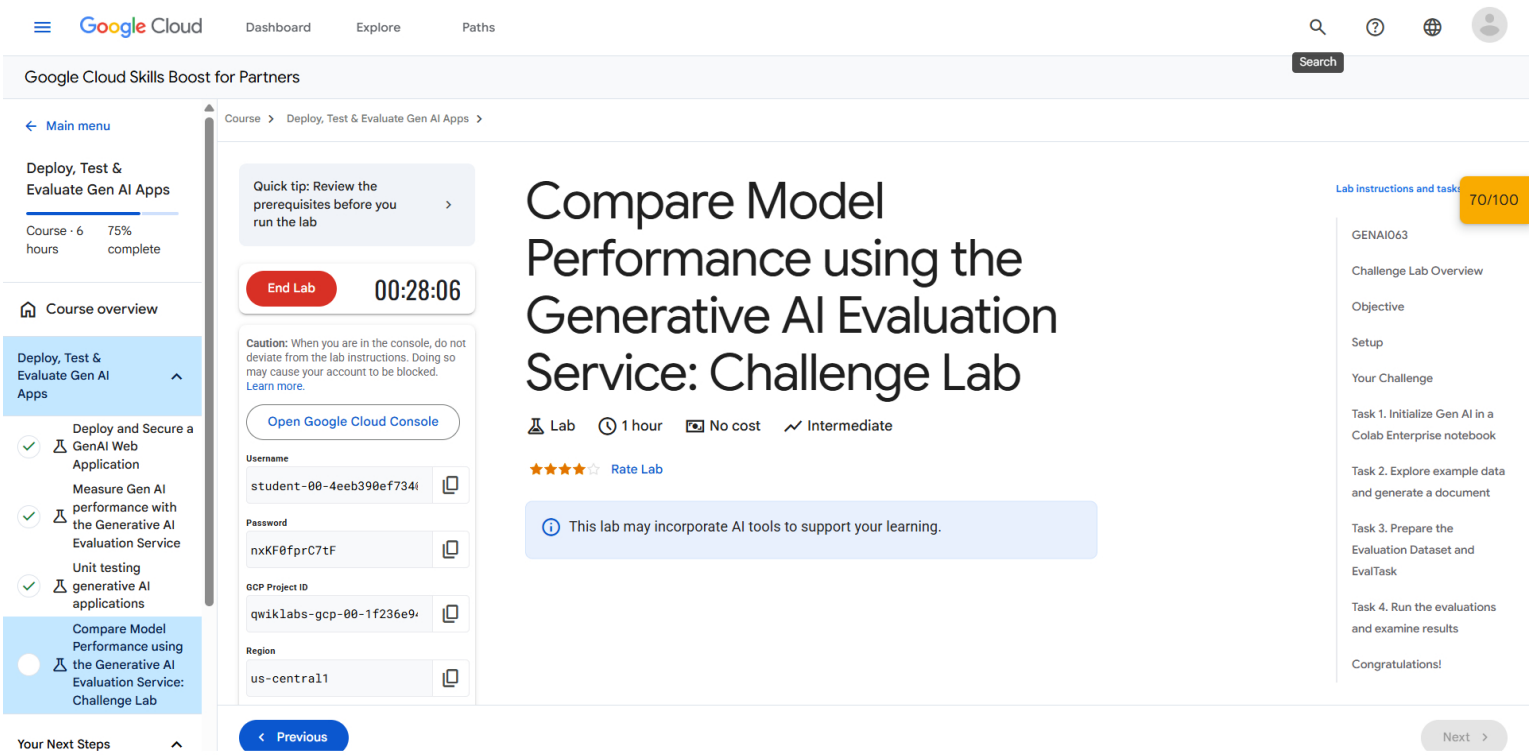


Challenge Lab Overview

This lab will challenge you to perform actions and automation across products. Instead of following step-by-step instructions, you are given a common business scenario and a set of tasks - you figure out how to complete them on your own! An automated scoring system (shown on this page) provides feedback on whether you have completed your tasks correctly.

When you take a Challenge Lab, you will not be taught Google Cloud concepts. You will need to use your skills to assess how to build the solution to the challenge presented. This lab is only recommended for students who have those skills. Are you up for the challenge?

Objective

This lab challenges you to conduct a model-based, pairwise evaluation on two models tasked with completing the same tasks. You will use the Generative AI Evaluation Service to complete this evaluation.

Setup

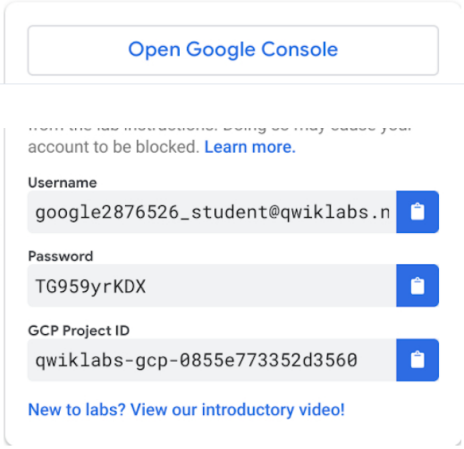
Qwiklabs setup

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, 02:00:00 and make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

3. When ready, click START LAB.

4. Note your lab credentials. You will use them to sign in to the Google Cloud Console.



5. Click **Open Google Console**.

6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

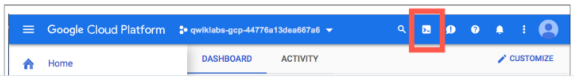
7. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

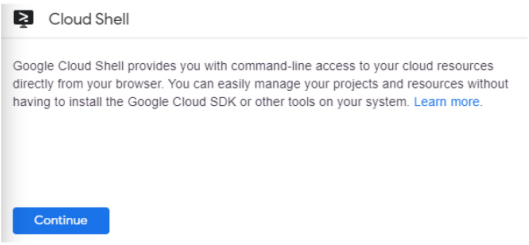
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

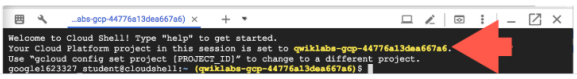
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
```

(Output)

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

```
gcloud config list project
```

(Output)

```
[core]
project = <project_ID>
```

(Example output)

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the gcloud command-line tool overview.

Your Challenge

You have been contracted by a movie production studio that wants to prepare for a series of low-budget short films. They've asked you to develop a generative AI tool to help them. They've provided you:

- Some unstructured notes on different phases of production
- A rate card which describes hourly rates for different crew positions on the films

You know that Gemini Flash is a faster and lower-cost alternative to Gemini Pro, so you'd like to quantify its performance to see if it would be an adequate alternative to Gemini Pro on these complex tasks.

Task 1. Initialize Gen AI in a Colab Enterprise notebook

In this task, you will be setting up a `Colab notebook` and initializing Gen AI to connect the notebook and generate creative text content.

1. In the **Google Cloud Console**, navigate to **Vertex AI > Colab Enterprise**.
2. If prompted, enable the required APIs.
3. Click on **+** to create a new notebook.

Note: While GCP Colab Enterprise Notebooks might default to the **us-central1** region, it's crucial to create your notebook in the same region where the lab environment is provisioned. You can find the lab's region on the left-hand side of the lab interface.

4. Rename the notebook to `cymbal-indie-film-planning`.
5. Paste the following code into the top cell and run it with **Shift + Return**.

Note: 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.

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** in the upper right of the notebook.

7. Clicking on the caret should reveal a set of menus above the notebook. Select **Runtime > Restart Session**. When asked to confirm, 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 add a new code and paste the following code below. Press **Shift + Return** to run the cell.

```
import pandas as pd
```

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

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

9. In a new code block, initialize Gen AI with `vertexai.init()`. Use the `us-central1` location and run the cell.

10. Save the notebook.

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

Create a notebook and allocate a runtime

Check my progress

Assessment Completed!

Task 2. Explore example data and generate a document

In this task, you will set up some sample data for a film production including crew rates, shooting schedules and then define questions for a large language model to answer.

1. Run the following code in a new cell to instantiate some example data. The calls to `cleandoc()` helps remove the indents and extra lines used for making the multi-line string readable in the code.

```
hourly_rates = cleandoc("""
  Screenwriter: $40
  Actor: $25
  Director: $30
  Camera Operator: $35
  Sound Engineer: $20
  Editor: $30
  """)

planning_notes = cleandoc("""
 Phases of Production:
   Writing:
   The Screenwriter will write the script.
   They need 72 hours to do so.


   Pre-Production:
   The Director needs time to analyze the script.
   They will work on it for 36 hours.
   The Camera Operator will join the director for 24 hours of
planning.
```

```
   The first three days of filming will require the director, 4
```

```
actors, the camera operator, and the sound engineer


   Production Phase 2
   The next three days of filming will require the director, 8
actors, the camera operator, and the sound engineer


   Post-Production
   The editor will take 64 hours to edit the film.
   The director will work with the editor for 24 hours during
this phase.
""")
```

2. Run the following code to define the content we would like the model to help us with.

```
tasks = [
    """What is the cost of each phase of production?


    """How many days will each phase require? Assume an
    8 hour work day. If multiple people are working in parallel,
    do not add those times together, but only use the longest
time.
    Also include a count of the total number of days of the
entire
    project.""",

    """Prepare a text schedule for all phases of the film
starting
    on Feb 3, 2025. The whole crew should be off Saturdays
    and Sundays."""
]
```

3. Next, define a prompt template.

```
prompt_template = cleandoc("""
  <instructions>
  Prepare a document to fulfill the task based on the context
provided.
  </instructions>


  </task>
<context>
  {context}
  </context>
  """)
```

4. You will compare how the lower-cost **Gemini Flash** compares against **Gemini Pro** on these instruction tasks to determine which you should use for this project. Instantiate a model variable `llm_pro` to contain a generative model using `gemini-2.5-pro-preview-05-06` and a model variable `llm_flash` to contain a generative model using `gemini-2.0-flash-001`.

5. Add a generation configuration to each model to set the temperature to 0.

6. Combine `hourly_rates` and `planning_notes` (with a pair of line breaks as a separator) to form a `context` chunk.

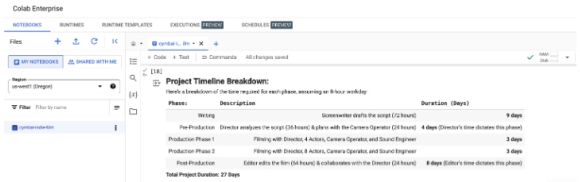
```
context = hourly_rates + "\n\n" + planning_notes
```

`Markdown()` class imported from `IPython.display` to wrap the response text to render Gemini's responses, which are often formatted as Markdown strings.

8. Note key differences between the responses generated by the models. Try double-checking some of the math against the ground truth if you notice differences between the model's outputs.

9. Save the notebook.

Output:



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

Explore example data and generate a document

Check my progress

Assessment Completed!

Task 3. Prepare the Evaluation Dataset and EvalTask

In this task, you will set up the data and scoring method to evaluate the models.

1. You will evaluate the models' responses against each other by using Pairwise question answering quality. Note the user input fields in curly braces in this prompt, which are required to evaluate this metrics. You will use the Gemini Pro responses as your **baseline model response** and your Gemini Flash responses

2. Prepare a Pandas DataFrame with the fields needed for evaluation.

3. Create an `EvalTask()`, passing in the dataset, identifying `MetricPromptTemplateExamples.Pairwise.QUESTION_ANSWERING_QUALITY` as the metric you would like to be calculated, and defining an experiment name of `indie-film-planning`.

4. Save the notebook.

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

Prepare the Evaluation Dataset and EvalTask

Check my progress

Assessment Completed!

Task 4. Run the evaluations and examine results

In this task, you will ask a model to choose a preferred response for each task from the two large language models `llm_pro` and `llm_flash`.

1. Run the evaluation of the `EvalTask` you configured above.

Note: This may take 2 minutes to run.

2. Print the `summary_table` of the evaluation results.

3. Display the evaluation response's `metrics_table`. Do you see a clear preference for either model, according to the evaluation service?

4. To simplify reading the results, display the column from the `metrics_table` that includes the evaluation service's preferred response for this example.

   service's preferences.

Which model do you think would be the more appropriate to use for this project?

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

Run the evaluations and store the results

Check my progress

Please run the evaluations with the llm_pro, llm_flash models and store the results.

Congratulations!

You have successfully conducted a model-based, pairwise evaluation of two models using the Generative AI Evaluation Service.

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

Manual Last Updated May 08, 2025

Lab Last Tested May 08, 2025