Quick tip: Review the prerequisites before you run the lab

Start Lab 01:00:00

Common Generative AI Use Cases

Lab 1 hour No cost Intermediate

★★★★½

This lab may incorporate AI tools to support your learning.

Overview

In this lab, you will implement several common generative AI use cases.

In addition to being the most common types of generative AI applications, these techniques serve as the building blocks of many more complex applications.

Previous

Thanks for reviewing this lab.

Next

Objective

You will learn to use text prompting for the following tasks:

- Ideation
- Text Classification
- Text Summarization
- Text Extraction
- Question Answering

Setup and requirements

Before you click the Start Lab button

Thanks for reviewing this lab.

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

To complete this lab, you need:

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

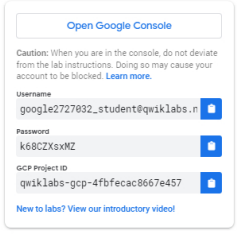
Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

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

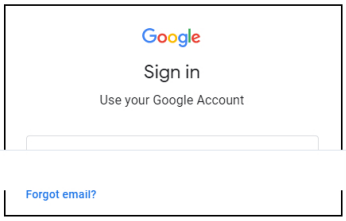
How to start your lab and sign in to the Google Cloud Console

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

Thanks for reviewing this lab.

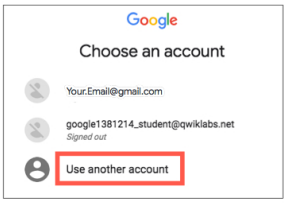


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



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).

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

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.

Task 1. Initialize Vertex AI in a Colab Enterprise notebook

In this task, you will be setting up a `Colab notebook` and initializing Vertex 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 **My notebooks** and select the region as `Lab Region` from dropdown and then click on **+ New Notebook** button.

4. Rename the notebook to `gen-ai-use-cases.ipynb`.

5. Paste the following code into the top cell and run it with **Shift + Return**.

```
%pip install --upgrade --quiet google-cloud-aiplatform
```

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.

Import packages

8. Click **+ Code** to create a new code cell and paste in the import code below. Press **Shift + Return** to run the cell.

```
from inspect import cleandoc
from IPython.display import display, Markdown

import vertexai
from vertexai.generative_models import GenerativeModel,
GenerationConfig
```


Initialize Vertex AI

9. Create a new code cell and paste the following into it.

10. Run the cell to initialize Vertex AI.

```
PROJECT = !gcloud config get-value project
PROJECT_ID = PROJECT[0]
REGION = "Lab Default Region"
vertexai.init(project=PROJECT_ID, location=REGION)
```

Task 2. Load a generative model

1. Create a new code cell and run the following to load Gemini Pro.

Task 3. Ideation

Within the context of using a generative AI model, "ideation" means using the model to help you brainstorm new content: article titles, sections to include in a document, interview questions, etc.

1. This is a case where you may want the model to be more creative or surprising in the content it generates. To enable that, create a GenerationConfig object by copying the code below to a new code cell and running it:

```
creative_gen_config = GenerationConfig(temperature=1,
top_p=0.8)
```

2. Create a new code cell and run the following code in it:


```
block?"

response = model.generate_content(prompt,

generation_config=creative_gen_config)
print(response.text)
```

Output:

```
    ## Strategies for Overcoming Writer's Block

    Writer's block is a common challenge faced by writers of all
levels. It can be frustrating and demotivating, but there are many
strategies you can use to overcome it. Here are a few tips:

    **1. Identify the Cause:**

    The first step is to identify what's causing your writer's
block. Are you feeling overwhelmed by the project? Do you lack
inspiration or direction? Are you struggling with perfectionism?
Once you understand the root of the problem, you can start to
address it.

    **2. Change Your Environment:**
```



```
creativity. Try writing in a different location, such as a coffee
shop, library, or park. You could also try writing at a different
time of day or using a different writing tool, like a pen and
paper instead of a computer.

    **3. Break Down the Task:**

    Large writing projects can feel daunting, which can lead to
writer's block. Break down your project into smaller, more
manageable tasks. This will make it feel less overwhelming and
help you get started.

    **4. Freewriting:**

    Freewriting is a great way to get your creative juices
flowing. Set a timer for 5-10 minutes and write whatever comes to
mind, without stopping or editing. This can help you overcome
perfectionism and generate new ideas.
```

```
    **5. Read:**

    Reading the work of other writers can be a great source of
inspiration. Pay attention to how other writers approach their
craft and try to incorporate new techniques into your own writing.

    **6. Talk to Someone:**
```

Thanks for reviewing this lab.

```
writer's block can be helpful. They may be able to offer some
insights or suggestions that you haven't thought of.

    **7. Take a Break:**

    Sometimes the best way to overcome writer's block is to simply
take a break. Go for a walk, watch a movie, or do something else
that you enjoy. When you come back to your writing, you may find
that you're feeling refreshed and ready to go.

    **8. Reward Yourself:**

    Set small goals for yourself and reward yourself for
completing them. This will help you stay motivated and make the
writing process more enjoyable.

    **9. Don't Give Up:**

    Writer's block is a temporary condition. Don't get discouraged
if you're struggling. Just keep trying different strategies and
eventually you'll find what works for you.

    Remember, everyone experiences writer's block from time to
time. The important thing is to keep trying and to find strategies
that help you overcome it.
```

```
    * **The Creative Penn:** https://www.thecreativepenn.com/how-
to-overcome-writers-block/
    * **MasterClass:** https://www.masterclass.com/articles/how-
to-overcome-writers-block
    * **Reedsy:** https://blog.reedsy.com/overcoming-writers-
block/

    I hope this information is helpful!
```

3. Run the following in another code cell for another example of Ideation:

```
prompt = "Provide ten interview questions for the role of
prompt engineer."

response = model.generate_content(prompt,

generation_config=creative_gen_config)
print(response.text)
```

Output:

```
    ## Ten interview questions for a prompt engineer role:

**

    This question assesses the candidate's basic understanding of
prompts and their awareness of the various types of prompts used
in NLP tasks.

    **2. Can you explain the role of a prompt engineer in the
development of large language models (LLMs)?**

    This probes the candidate's understanding of the prompt
engineering process and its importance in building effective LLMs.

    **3. How do you approach designing a prompt for a specific NLP
task?**

    This explores the candidate's thought process and methodology
for designing prompts. Ideally, the answer should involve steps
like understanding the task, analyzing the LLM capabilities, and
crafting the prompt accordingly.

    **4. What are some challenges you have faced in designing
effective prompts? How did you overcome them?**

    This assesses the candidate's experience with troubleshooting
and finding solutions to challenges in prompt engineering.
```

```
natural language examples?**

    This assesses the candidate's knowledge of specific techniques
used in prompt engineering.

    **6. How do you evaluate the effectiveness of a prompt? What
metrics do you use?**

    This evaluates the candidate's understanding of evaluating and
measuring the performance of prompts.

    **7. How do you stay updated on the latest research and
developments in prompt engineering?**

    This assesses the candidate's commitment to continuous
learning and staying updated in the fast-evolving field of prompt
engineering.
```

```
    **8. Can you share an example of a prompt you designed and how
it contributed to improved performance on a specific NLP task?**

    This allows the candidate to showcase their practical
experience and demonstrate their ability to apply their knowledge.

    **9. Describe a situation where you collaborated with other
engineers or researchers on a prompt engineering project. What was
```

```
    This assesses the candidate's ability to collaborate and
communicate effectively in a team environment.

    **10. What are your thoughts on the future of prompt
engineering and its potential impact on the field of NLP?**

    This allows the candidate to share their vision and
understanding of the future direction of prompt engineering and
its role in shaping the future of NLP.

    These ten questions can be used as a starting point for
interviewing a prompt engineer. You can adapt them and add further
questions based on your specific requirements and the candidate's
experience level.
```

Task 4. Text Classification

In modern enterprises, a lot of work is dedicated to putting the right information in the

generative AI to classify those blocks of text to determine the right team.

Classification can have many possible applications, including:

- Sentiment analysis
- Topic classification
- Spam detection
- Intent recognition
- Language identification
- Toxicity detection
- Emotion detection

1. For a more tightly-constrained summary, you may want to decrease the temperature and top_p parameters:

```
predictable_gen_config = GenerationConfig(temperature=0.1,
top_p=0.1)
```

2. Run the following code cell for an example of intent detection:

```
prompt = """
an order". \n
user input: Hi, can you please book a table for two at Juan
for May 1?
"""

response = model.generate_content(
    prompt, generation_config=predictable_gen_config
)
print(response.text)
```

Output:

```
## Intent: Making a reservation

The user wants to make a reservation at a restaurant called Juan
for two people on May 1st.
```

3. You can help guide correct classifications of your data by providing examples:

```
prompt = """
What is the topic for a given news headline? \n
- health \n
- sports \n
- technology \n\n

Text: Pixel 7 Pro Expert Hands On Review. \n
The answer is: technology \n

Text: Quit smoking? \n
The answer is: health \n

Text: Birdies or bogeys? Top 5 tips to hit under par \n
The answer is: sports \n

Text: Relief from local minimum-wage hike looking more
remote \n
The answer is: business \n

Text: You won't guess who just arrived in Bari, Italy for
```

```
the movie premiere. \n
The answer is:
"""

response = model.generate_content(
    prompt, generation_config=predictable_gen_config
)
print(response.text)
```

Output:

```
entertainment
```

4. One great thing about defining a task as a classification task is that you can evaluate classification results very easily and meaningfully. Run this cell to import `pandas` as well as a `classification_report` metric from `sklearn`. You will also define some ground truth data for evaluation.

```
import pandas as pd
from sklearn.metrics import classification_report

review_data_df = pd.DataFrame(
    {
        "review": [
            "i love this product. it does have everything i
am looking for!",
            "all i can say is that you will be happy after
buying this product",
            "its way too expensive and not worth the
price",
            "i am feeling okay. its neither good nor too
bad.",
        ],
        "sentiment_groundtruth": ["positive", "positive",
"negative", "neutral"],
    }
)

review_data_df
```

Output:

	review	sentiment_groundtruth
0	i love this product. it does have everything i...	positive
1	all i can say is that you will be happy after ...	positive
2	its way too expensive and not worth the price	negative
3	i am feeling okay. its neither good nor too bad.	neutral

5. Now that you have the data with reviews and sentiments as ground truth labels, you can call the text generation model on each review row using the `apply` function. The `get_sentiment` function will be called to predict the sentiment for each row's `review` column, and the results will be stored in the `sentiment_prediction` column.

```
    prompt = f"""Classify the sentiment of the following
review as "positive", "neutral" or "negative". Return only
the classification.
                review: {row}
                """
    response = model.generate_content(
        contents=prompt,
generation_config=predictable_gen_config
    ).text
    return response


review_data_df["sentiment_prediction"] =
review_data_df["review"].apply(get_sentiment)
review_data_df
```

Output:

	review	sentiment_groundtruth	sentiment_prediction
0	i love this product. it does have everything i...	positive	positive
1	all i can say is that you will be happy after ...	positive	positive
3	i am feeling okay. its neither good nor too bad.	neutral	neutral

6. Now you can call the `classification_report` function from `sklearn` to return classification metrics based on comparing the `sentiment_groundtruth` and predicted `sentiment_prediction` fields:

```
report = classification_report(
    review_data_df["sentiment_groundtruth"],
review_data_df["sentiment_prediction"]
)
Markdown(report)
```

Output:

```
              precision    recall  f1-score   support

    negative       1.00      1.00      1.00         1
     neutral       1.00      1.00      1.00         1
    positive       1.00      1.00      1.00         2

weighted avg       1.00      1.00      1.00         4
```

7. If you need a refresher on interpreting this report, you can review Classification metrics in the Google ML Crash Course.

Task 5. Text Summarization

Sometimes you want to read the entirety of a text, and sometimes you just want the gist of it. Generative AI models can help create summaries of longer texts.

1. You may want to continue to use the more tightly-constrained temperature and top_p parameters we used above:

```
predictable_gen_config = GenerationConfig(temperature=0.1,
top_p=0.1)
```

the length of the summary desired. Run the following code block in a new code cell to generate a short summary.

```
prompt = """
Provide a very short summary, no more than three sentences,
for the following article:

Our quantum computers work by manipulating qubits in an
orchestrated fashion that we call quantum algorithms.
The challenge is that qubits are so sensitive that even
stray light can cause calculation errors — and the problem
worsens as quantum computers grow.
This has significant consequences, since the best quantum
algorithms that we know for running useful applications
require the error rates of our qubits to be far lower than
we have today.
To bridge this gap, we will need quantum error correction.
Quantum error correction protects information by encoding
it across multiple physical qubits to form a "logical
qubit," and is believed to be the only way to produce a
large-scale quantum computer with error rates low enough
for useful calculations.
Instead of computing on the individual qubits themselves,
we will then compute on logical qubits. By encoding larger
enable useful quantum algorithms.

Summary:

"""

response = model.generate_content(
    prompt, generation_config=predictable_gen_config
)
print(response.text)
```

Output:

```
## Summary:

Quantum computers are sensitive to errors, which limits their
usefulness. Quantum error correction aims to solve this by
encoding information across multiple qubits, creating a more
stable "logical qubit". This could enable large-scale quantum
computers to perform complex calculations with low error rates.
```

3. You might prefer your summary to be delivered as bullet points, displayed using Markdown to interpret the text formatting that Gemini returns:

```
prompt = """
Provide a very short summary in four bullet points for the
following article:

Our quantum computers work by manipulating qubits in an
orchestrated fashion that we call quantum algorithms.
The challenge is that qubits are so sensitive that even
stray light can cause calculation errors — and the problem
worsens as quantum computers grow.
This has significant consequences, since the best quantum
algorithms that we know for running useful applications
require the error rates of our qubits to be far lower than
we have today.
To bridge this gap, we will need quantum error correction.
Quantum error correction protects information by encoding
it across multiple physical qubits to form a "logical
qubit," and is believed to be the only way to produce a
large-scale quantum computer with error rates low enough
for useful calculations.
```

```
Instead of computing on the individual qubits themselves,
we will then compute on logical qubits. By encoding larger
numbers of physical qubits on our quantum processor into
one logical qubit, we hope to reduce the error rates to
enable useful quantum algorithms.

Bullet points:
```

```
response = model.generate_content(
    contents=prompt,
generation_config=predictable_gen_config
)
Markdown(response.text)
```

Output:

```
* Quantum computers use qubits, which are sensitive to errors.
* Current error rates are too high for useful applications.
* Quantum error correction is needed to reduce error rates.
* This involves encoding information across multiple qubits to
form a "logical qubit"
```

4. Summarization can also be done on other forms of documents, like an email thread or the transcript of a conversation:

```
prompt = """
Please generate a summary of the following conversation and
at the end summarize the to-do's for the support Agent:

Support Agent: Hi, Larry. How would you like to see this
resolved?

Customer: That's alright. I want to return the item and get
a refund, please.

Support Agent: Of course. I can process the refund for you
now. Can I have your order number, please?

Customer: It's [ORDER NUMBER].

Support Agent: Thank you. I've processed the refund, and
you will receive your money back within 14 days.

Customer: Thank you very much.

Support Agent: You're welcome, Larry. Have a good day!

Summary:
"""

response = model.generate_content(
    contents=prompt,
generation_config=predictable_gen_config
)
Markdown(response.text)
```

Output:

```
## Conversation Summary:

* **Customer:** Larry received the wrong item.
* **Desired Resolution:** Larry wants to return the item and
receive a refund.
* **Resolution:** The support agent processed the refund for the
order.
* **Refund Timeline:** Larry will receive the refund within 14
days.

## To-Do's for Support Agent:

* **Confirm:** Verify the refund has been processed correctly.
* **Inform:** Let the customer know when they can expect the
refund.
* **Close:** Close the support ticket.
```

Task 6. Text Extraction

Text Extraction is the process of pulling structured fields from unstructured text. By unstructured text, we mean text that lacks a computer-readable structure like CSV, JSON, or YAML, even if a human can identify some structure (like in the ingredient list of a recipe).

Some specific types of extraction include:

- **Named entity recognition (NER)**: Extract named entities from text, including people, places, organizations, and dates.
- **Relation extraction**: Extract the relationships between entities in text, such as family relationships between people.
- **Event extraction**: Extract events from text, such as project milestones and product launches.
- **Question answering**: Extract information from text to answer a question.

1. Run the following code in a code cell to see an example of Extraction.

```
order = "We need eight grilled cheese sandwiches. Two with
swiss cheese, three with muenster, three with cheddar."
```

```
keys for the following fields:
    - item_name
    - cheese_selection

    Order:
    {order_field}
""".format(order_field=order)

response = model.generate_content(
    contents=prompt,
generation_config=predictable_gen_config
)
print(response.text)
```

Output:

```
[
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Swiss"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Swiss"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Muenster"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Muenster"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Muenster"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Cheddar"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Cheddar"
  },
  {
    "item_name": "Grilled Cheese Sandwich",
    "cheese_selection": "Cheddar"
  }
]
```

according to their descriptions in order to allow you to call a function. While this is outside of the scope of this lab, you may want to investigate using these capabilities for Extraction, even if your end goal isn't to call a function.

Task 7. Question Answering

Answering questions is one of the most common and most impressive uses of generative AI. The information the model returns could come from patterns in data the model was trained on or from additional information you provide the model as context.

1. Run this code for an example of answering an **Open Domain** question, meaning its answer is publically available on the internet and does not require very recent or up-to-date knowledge. An answer that meets those qualifications may have been included in the model's training data:

```
prompt = """Q: Who was President of the United States in
            A:
        """
response = model.generate_content(
    contents=prompt,
generation_config=predictable_gen_config
)
print(response.text)
```

Output:

```
Dwight D. Eisenhower was President of the United States in 1955.
He belonged to the Republican Party.
```

2. Run this code for an example of answering an **closed domain** question, meaning its answer may be private to your organization and therefore you need to provide that information to the model within the prompt's "context":

```
context = """
Storage and content policy \n
How durable is my data in Cloud Storage? \n
Cloud Storage is designed for 99.999999999% (11 9's) annual
durability, which is appropriate for even primary storage
is achieved through erasure coding that stores data pieces
redundantly
across multiple devices located in multiple availability
zones.
Objects written to Cloud Storage must be redundantly stored
in at least two different availability zones before the
write is acknowledged as successful. Checksums are stored
and regularly revalidated to proactively verify that the
data
integrity of all data at rest as well as to detect
corruption of data in transit. If required, corrections are
automatically
made using redundant data. Customers can optionally enable
object versioning to add protection against accidental
deletion.
"""

question = "How is high availability achieved?"

prompt = f"""Answer the question given in the contex below:
Context: {context}?\n
Question: {question} \n
Answer:
"""

print("[Prompt]")
print(prompt)

print("[Response]")
response = model.generate_content(contents=prompt,
generation_config=predictable_gen_config)
print(response.text)
```

Output:

```
[Prompt]
Answer the question given in the contex below:
Context:
Storage and content policy

How durable is my data in Cloud Storage?

Cloud Storage is designed for 99.999999999% (11 9's) annual
durability, which is appropriate for even primary storage and
business-critical applications. This high durability level is
achieved through erasure coding that stores data pieces
redundantly
across multiple devices located in multiple availability
zones.
Objects written to Cloud Storage must be redundantly stored in
at least two different availability zones before the
write is acknowledged as successful. Checksums are stored and
regularly revalidated to proactively verify that the data
integrity of all data at rest as well as to detect corruption
of data in transit. If required, corrections are automatically
made using redundant data. Customers can optionally enable
object versioning to add protection against accidental deletion.
?

Question: How is high availability achieved?

Answer:

[Response]
High availability is achieved through erasure coding that
stores data pieces redundantly across multiple devices located in
multiple availability zones.
```

Note: Evaluating the correctness of question-answering systems can be challenging, but Vertex AI provides a Generative AI evaluation service with some metrics tailor-made for this use case.

Congratulations!

Congratulations! You have successfully taken a survey at several of the most common text-based generative AI use cases. Many complicated use cases build on these basic patterns, but these fundamental tasks will be utilized at all levels of generative AI practitioners.

Next Steps

- Study Gemini Function Calling with Gemini.
- Learn more about the Generative AI evaluation service.

Manual Last Updated April 11, 2025

Lab Last Tested April 11, 2025