Welcome to Colab!

Explore the Gemini API

The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code, and audio.

How to get started?

- Go to Google AI Studio and log in with your Google account.
- Create an API key.
- Use a quickstart for Python, or call the REST API using curl.

Discover Gemini's advanced capabilities

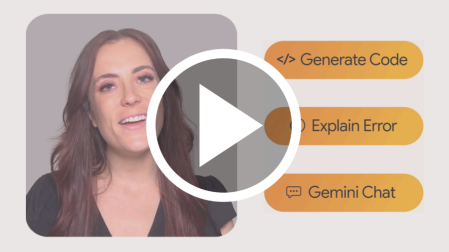
- Play with Gemini multimodal outputs, mixing text and images in an iterative way.
- Discover the multimodal Live API (demo here).
- Learn how to analyze images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
- Unlock the power of Gemini thinking model, capable of solving complex task with its inner thoughts.

Explore complex use cases

- Use Gemini grounding capabilities to create a report on a company based on what the model can find on internet.
- Extract invoices and form data from PDF in a structured way.
- Create illustrations based on a whole book using Gemini large context window and Imagen.

To learn more, check out the Gemini cookbook or visit the Gemini API documentation.

Colab now has AI features powered by Gemini. The video below provides information on how to use these features, whether you're new to Python, or a seasoned veteran.



What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab or Colab Features You May Have Missed to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

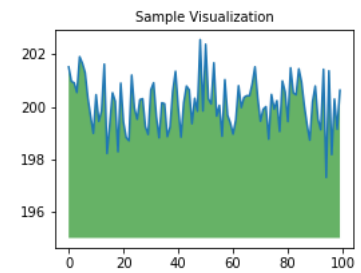
You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```



Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

For example, if you find yourself waiting for **pandas** code to finish running and want to go faster, you can switch to a GPU Runtime and use libraries like RAPIDS cuDF that provide zero-code-change acceleration.

To learn more about accelerating pandas on Colab, see the 10 minute guide or US stock market data analysis demo.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colab
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning

These are a few of the notebooks related to Machine Learning, including Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Intro to RAPIDS cuDF to accelerate pandas
- Getting Started with cuML's accelerator mode
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TPUs in Colab

Featured examples

- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
# STEP 1: Load the text dataset
import tensorflow as tf
import numpy as np
import os

# Download and read the text data
path = tf.keras.utils.get_file("little_prince.txt", "https://www.gutenberg.org/files/1417/1417-0.txt")
text = open(path, encoding='utf-8').read().lower()
```

```
text = text[:100000]  # ▼ Use only the first 100,000 characters for faster training
print(f"Reduced text length: {len(text)} characters")

# STEP 2: Preprocess the data
vocab = sorted(set(text))
char2idx = {char: idx for idx, char in enumerate(vocab)}
idx2char = np.array(vocab)
text_as_int = np.array([char2idx[c] for c in text])

# Create training examples and targets
seq_length = 100
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

def split_input_target(chunk):
    return chunk[:-1], chunk[1:]

dataset = sequences.map(split_input_target)

# STEP 3: Build the LSTM model (optimized)
BATCH_SIZE = 32  # ▼ Reduced
BUFFER_SIZE = 10000
EMBEDDING_DIM = 128  # ▼ Reduced
RNN_UNITS = 256      # ▼ Reduced
VOCAB_SIZE = len(vocab)

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)

def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    return tf.keras.Sequential([
        tf.keras.layers.Input(shape=(None,), batch_size=batch_size),
        tf.keras.layers.Embedding(vocab_size, embedding_dim),
        tf.keras.layers.LSTM(rnn_units, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
        tf.keras.layers.Dense(vocab_size)
    ])

model = build_model(VOCAB_SIZE, EMBEDDING_DIM, RNN_UNITS, BATCH_SIZE)

# STEP 4: Compile and train the model
def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

model.compile(optimizer='adam', loss=loss)

# Checkpoint setup
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}.weights.h5")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True
)

EPOCHS = 1  # ▼ Reduced to 1 epoch for quick testing
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])

# STEP 5: Text Generation with Temperature Scaling
def generate_text(model, start_string, temperature=1.0):
    model = build_model(VOCAB_SIZE, EMBEDDING_DIM, RNN_UNITS, batch_size=1)
    model.load_weights(os.path.join(checkpoint_dir, "ckpt_1.weights.h5"))
    model.build(tf.TensorShape([1, None]))

    input_eval = [char2idx[s] for s in start_string.lower()]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []

    for i in range(500):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)

        predictions = predictions / temperature
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()

        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(idx2char[predicted_id])
```

```
    return start_string + ''.join(text_generated)

# Generate sample outputs
print("\n--- Generated Text with Temperature = 0.5 ---\n")
print(generate_text(model, start_string="Once upon a time ", temperature=0.5))

print("\n--- Generated Text with Temperature = 1.0 ---\n")
print(generate_text(model, start_string="Once upon a time ", temperature=1.0))
```

```
Downloading data from https://www.gutenberg.org/files/1417/1417-0.txt
692241/692241 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Reduced text length: 100000 characters
30/30 ━━━━━━━━━━━━━━━━━━━━ 18s 392ms/step - loss: 3.4927

--- Generated Text with Temperature = 0.5 ---

Once upon a time  s  ede ae eet ati, hisd naa sos  rl ap r  v o .rhle a n t   e oaleioat hut hene nih  no oftanr nr r too   nd  a teaihe
rreof  coia e do  t  to anloit a t e e st ae aeid  s,aueace t t e oa tad lel asl otn  ti t esboe ia  t hs an s e, a wr eh i t  lca     a
 s rilth se tte t  ano tuse ue  t ons intit we in fs laor eer te en taaoho  ra    s e r he n rayoneo to lit o  hoa sse ttlo e   taateaea

--- Generated Text with Temperature = 1.0 ---

Once upon a time sn iy*heernnednaoatnseit litho na(ilhrisuo to fhv ao  sctuoohe prythmetbia tleme thee ionsr ht ksi,uddtbtefc ythpdt a,,
rtbaffotetrs efgi o ei
s smrewcsvmi
ltgisepwaamui tntci, ,vnongoc e h dnno ccohrn f e a dashte o ennelyac u smmffn a  ef v tos d moonyl sidie-nhifdfnfhodcrr wttli
emhibsbshhrofe _suonr, ce  ntad rgsiocrso pociruhrnhesgncoaorsh oef gdner oio   s nw s a  oe rh a sgy wcdddseo
 lrs tw nu o)a,fo ese ufuuc es,ihwkp  t,nhttd lhnaauitnentnn ow ,dsfmtseps
drpala wrwbii
```

```
import nltk
# Download all essential resources safely
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download required NLTK data (only once)
nltk.download('punkt')
nltk.download('stopwords')

def preprocess_nlp(sentence):
    # Step 1: Tokenization
    tokens = word_tokenize(sentence)
    print("1. Original Tokens:")
    print(tokens)

    # Step 2: Remove Stopwords
    stop_words = set(stopwords.words('english'))
    tokens_without_stopwords = [word for word in tokens if word.lower() not in stop_words]
    print("\n2. Tokens Without Stopwords:")
    print(tokens_without_stopwords)

    # Step 3: Stemming
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in tokens_without_stopwords]
    print("\n3. Stemmed Words:")
    print(stemmed_tokens)

# Sample sentence
sentence = "NLP techniques are used in virtual assistants like Alexa and Siri."
preprocess_nlp(sentence)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
1. Original Tokens:
['NLP', 'techniques', 'are', 'used', 'in', 'virtual', 'assistants', 'like', 'Alexa', 'and', 'Siri', '.']

2. Tokens Without Stopwords:
['NLP', 'techniques', 'used', 'virtual', 'assistants', 'like', 'Alexa', 'Siri', '.']

3. Stemmed Words:
```

```
['nlp', 'techniqu', 'use', 'virtual', 'assist', 'like', 'alexa', 'siri', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
import spacy

# Load English language model
nlp = spacy.load("en_core_web_sm")

# Input sentence
sentence = "Barack Obama served as the 44th President of the United States and won the Nobel Peace Prize in 2009."

# Apply NLP pipeline
doc = nlp(sentence)

# Extract and print entities
for ent in doc.ents:
    print(f"Text: {ent.text}, Label: {ent.label_}, Start: {ent.start_char}, End: {ent.end_char}")
```

```
Text: Barack Obama, Label: PERSON, Start: 0, End: 12
Text: 44th, Label: ORDINAL, Start: 27, End: 31
Text: the United States, Label: GPE, Start: 45, End: 62
Text: the Nobel Peace Prize, Label: WORK_OF_ART, Start: 71, End: 92
Text: 2009, Label: DATE, Start: 96, End: 100
```

```
import numpy as np

def scaled_dot_product_attention(Q, K, V):
    # Step 1: Compute dot product of Q and K^T
    matmul_qk = np.dot(Q, K.T)

    # Step 2: Scale by sqrt(d), where d is the key dimension
    d_k = K.shape[-1]
    scaled_scores = matmul_qk / np.sqrt(d_k)

    # Step 3: Apply softmax to get attention weights
    def softmax(x):
        e_x = np.exp(x - np.max(x, axis=-1, keepdims=True))
        return e_x / e_x.sum(axis=-1, keepdims=True)

    attention_weights = softmax(scaled_scores)

    # Step 4: Multiply attention weights by V
    output = np.dot(attention_weights, V)

    return attention_weights, output

# Test inputs
Q = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])
K = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])
V = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

# Run the function
attention_weights, output = scaled_dot_product_attention(Q, K, V)

# Print results
print("1. Attention Weights:\n", attention_weights)
print("\n2. Final Output:\n", output)
```

```
1. Attention Weights:
 [[0.73105858 0.26894142]
 [0.26894142 0.73105858]]

2. Final Output:
 [[2.07576569 3.07576569 4.07576569 5.07576569]
 [3.92423431 4.92423431 5.92423431 6.92423431]]
```

```
# Sentiment Analysis using HuggingFace Transformers

from transformers import pipeline
```

```
# Load a pre-trained sentiment analysis pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# Input sentence
sentence = "Despite the high price, the performance of the new MacBook is outstanding."

# Analyze sentiment
result = sentiment_pipeline(sentence)[0]

# Display results
print(f"Sentiment: {result['label']}")
print(f"Confidence Score: {round(result['score'], 4)}")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%  629/629 [00:00<00:00, 10.8kB/s]
model.safetensors: 100%  268M/268M [00:08<00:00, 33.4MB/s]
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 889B/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 2.00MB/s]
Device set to use cpu
Sentiment: POSITIVE
Confidence Score: 0.9998
```