Using OpenAI LLMs of GPT 3.5 turbo

```
!pip install openai
```

```
Requirement already satisfied: openai in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.1
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-pa
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.1
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.10/d
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.1
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/py
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/pyth
```

```
import openai

openai.api_key = "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"

def chat_with_gpt(prompt):
  response = openai.ChatCompletion.create(
      model="gpt-3.5-turbo",
      messages=[{"role": "user", "content": prompt}]
  )
  return response.choices[0].message.content.strip()

  if _name_ == "_main_":
    while True:
      user_input = input("You : ")
      if user_input.lower() in ["quite" , "exit" , "bye"]:
        break

      response = chat_with_gpt(user_input)
      print("Chatbot:", response)
```

HuggingFace Tranformer models: Using LangChain

```
!pip install langchain
```

```
  Downloading langchain-0.1.14-py3-none-any.whl (812 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 812.8/812.8 kB 6.6 MB/s eta 0
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/pyth
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/li
```

```
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain)
  Downloading dataclasses_json-0.6.4-py3-none-any.whl (28 kB)
Collecting jsonpatch<2.0,>=1.33 (from langchain)
  Downloading jsonpatch-1.33-py2.py3-none-any.whl (12 kB)
Collecting langchain-community<0.1,>=0.0.30 (from langchain)
  Downloading langchain_community-0.0.31-py3-none-any.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 11.8 MB/s eta 0:0
Collecting langchain-core<0.2.0,>=0.1.37 (from langchain)
  Downloading langchain_core-0.1.40-py3-none-any.whl (276 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 276.8/276.8 kB 14.9 MB/s eta
Collecting langchain-text-splitters<0.1,>=0.0.1 (from langchain)
  Downloading langchain_text_splitters-0.0.1-py3-none-any.whl (21 kB)
Collecting langsmith<0.2.0,>=0.1.17 (from langchain)
  Downloading langsmith-0.1.41-py3-none-any.whl (91 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 91.7/91.7 kB 12.4 MB/s eta 0
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/py
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.1
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/pytho
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/
Collecting marshmallow<4.0.0,>=3.18.0 (from dataclasses-json<0.7,>=0.5.7->
  Downloading marshmallow-3.21.1-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.4/49.4 kB 7.9 MB/s eta 0:0
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->lan
  Downloading typing_inspect-0.9.0-py3-none-any.whl (8.8 kB)
Collecting jsonpointer>=1.9 (from jsonpatch<2.0,>=1.33->langchain)
  Downloading jsonpointer-2.4-py2.py3-none-any.whl (7.8 kB)
Collecting packaging<24.0,>=23.2 (from langchain-core<0.2.0,>=0.1.37->langc
  Downloading packaging-23.2-py3-none-any.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.0/53.0 kB 8.6 MB/s eta 0:0
Collecting orjson<4.0.0,>=3.9.14 (from langsmith<0.2.0,>=0.1.17->langchain
  Downloading orjson-3.10.0-cp310-cp310-manylinux_2_17_x86_64.manylinux201
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.8/144.8 kB 11.2 MB/s eta
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/py
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/pyt
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.1
Collecting mypy-extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclass
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Installing collected packages: packaging, orjson, mypy-extensions, jsonpoin
  Attempting uninstall: packaging
    Found existing installation: packaging 24.0
    Uninstalling packaging-24.0:
      Successfully uninstalled packaging-24.0
Successfully installed dataclasses-json-0.6.4 jsonpatch-1.33 jsonpointer-2
```

```
from langchain import HuggingFaceHub
from langchain import PromptTemplate, LLMChain
repo_id = "tiiuae/falcon-7b-instruct"
huggingfacehub_api_token = "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"
llm = HuggingFaceHub(huggingfacehub_api_token=huggingfacehub_api_token
```

```
llm = HuggingFaceHub(huggingfacehub_api_token=huggingfacehub_api_token,
                     repo_id=repo_id,
                     model_kwargs={"temparature": 0.7 , "max_new_tokens": 500}


from re import template
template = """Question:{question}
             Answer: Let's give a detailed answer."""
prompt = PromptTemplate(template=template, input_variables=["question"])


chain = LLMChain(prompt=prompt, llm=llm)
```

```
out = chain.run("Give me a good receipe to make a cup cake.")
print(out)
```

```
Question:Give me a good receipe to make a cup cake.
                Answer: Let's give a detailed answer.

    Ingredients:
    - 1 1/2 cups all-purpose flour
    - 1/2 cup unsweetened cocoa powder
    - 1/2 teaspoon baking soda
    - 1/2 teaspoon baking powder
    - 1/2 teaspoon salt
    - 1/2 cup vegetable oil
    - 1/2 cup buttermilk
    - 1/2 cup granulated sugar
    - 1 large egg
    - 1 teaspoon vanilla extract
```

```
- 1/2 cup boiling water

Instructions:
1. Preheat oven to 350 degrees F. Grease a 12-cup muffin tin.
2. In a large bowl, whisk together the flour, cocoa powder, baking soda, ba
3. In a separate bowl, whisk together the oil, buttermilk, sugar, egg, and
4. Slowly add the wet ingredients to the dry ingredients, stirring until ev
5. Add the boiling water and mix until everything is evenly distributed.
6. Pour the batter into the prepared muffin tin.
7. Bake for 20-25 minutes, or until a toothpick inserted into the center of
8. Let cool for 5 minutes before removing from the pan.
9. Decorate with your favorite frosting and sprinkles.
10. Enjoy your delicious cup cakes!
```

```
from re import template
template = """Question:{question}
             Answer: Let's give a detailed answer with logical deductions in a
prompt = PromptTemplate(template=template, input_variables=["question"])
chain = LLMChain(prompt=prompt, llm=llm)
out = chain.run("Give me the number theoretical proof of Pythagoras Theorem.")
print(out)
```

```
Question:Give me the number theoretical proof of Pythagoras Theorem.
                Answer: Let's give a detailed answer with logical deductions
The number theoretical proof of Pythagoras Theorem is based on the concept

In the context of Pythagoras Theorem, we can prove that the hypotenuse of a

To prove this theorem, we can use the Pythagorean Theorem, which states tha

Let's consider a right triangle with legs 'a' and 'b' and hypotenuse 'c'. T

Let's assume that the length of leg 'a' is x and the length of leg 'b' is y

Now, let's consider the theorem in reverse. Let's assume that the length of

Then, we can calculate the length of leg 'a' as sqrt(x^2 + y^2) and the len
```

```
Then, we can calculate the length of leg 'a' as sqrt(x^2 + y^2) and the le

Now, let's consider the theorem in reverse. Let's assume that the length o

Now, we can calculate the length of leg 'a' as sqrt(x^2 + y^2) and the leng
```

```
!pip install huggingface_hub
!pip install transformers
!pip install langchain
!pip install chainLit
```

```
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.1
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/li
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/d
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
Requirement already satisfied: transformers in /usr/local/lib/python3.10/d
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in /usr/local/
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/di
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.1
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/py
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dis
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/li
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/d
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
Collecting langchain
  Downloading langchain-0.1.15-py3-none-any.whl (814 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 814.5/814.5 kB 5.3 MB/s eta 0
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/di
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/pyt
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/l
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain)
  Downloading dataclasses_json-0.6.4-py3-none-any.whl (28 kB)
Collecting jsonpatch<2.0,>=1.33 (from langchain)
  Downloading jsonpatch-1.33-py2.py3-none-any.whl (12 kB)
Collecting langchain-community<0.1,>=0.0.32 (from langchain)
  Downloading langchain_community-0.0.32-py3-none-any.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 30.3 MB/s eta 0:0
Collecting langchain-core<0.2.0,>=0.1.41 (from langchain)
  Downloading langchain_core-0.1.41-py3-none-any.whl (278 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 278.4/278.4 kB 19.8 MB/s eta
Collecting langchain-text-splitters<0.1,>=0.0.1 (from langchain)
  Downloading langchain_text_splitters-0.0.1-py3-none-any.whl (21 kB)
```

```
Collecting langsmith<0.2.0,>=0.1.17 (from langchain)
  Downloading langsmith-0.1.43-py3-none-any.whl (103 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.9/103.9 kB 10.4 MB/s eta
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/py
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.1
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python
```

```
import os
import chainlit as cl
from langchain import HuggingFaceHub, PromptTemplate, LLMChain


from getpass import getpass
huggingfacehub_api_token = getpass()
os.environ['huggingfacehub_api_token'] = huggingfacehub_api_token
```

```
··········
```

```
# Setting the Conversational Model

model_id = "gpt2-medium" #380M parameters
conv_model = HuggingFaceHub(huggingfacehub_api_token = os.environ['huggingface
                            repo_id=model_id,
                            model_kwargs={"temperature": 0.8 , "max_new_tokens"


template = """You are a helpul AI assistant that makes stories by completing t
              {query}
           """
prompt = PromptTemplate(template=template, input_variables=['query'])


conv_chain = LLMChain(llm=conv_model,
                      prompt=prompt,
                      verbose=True)
print(conv_chain.run("Once upon a time in India, there was a emperor Asoka who
```

```
> Entering new LLMChain chain...
Prompt after formatting:
You are a helpul AI assistant that makes stories by completing the query pr
              Once upon a time in India, there was a emperor Asoka who had


> Finished chain.
You are a helpul AI assistant that makes stories by completing the query pr
              Once upon a time in India, there was a emperor Asoka who had
                The emperor Asoka's empire was so large that it could suppo
               The capital was in charge of the management of all the citi
```

Using Google AI : Gemini 1.5 pro the next version of BARD

```
!pip install -q -U google-generativeai
```

```
158.8/158.8 kB 3.7 MB/s eta
```

```
import pathlib
import textwrap

import google.generativeai as genai

from IPython.display import display
from IPython.display import Markdown


def to_markdown(text):
  text = text.replace('•', '  *')
  return Markdown(textwrap.indent(text, '> ', predicate=lambda _: True))


# Used to securely store your API key
from google.colab import userdata


# Or use `os.getenv('GOOGLE_API_KEY')` to fetch an environment variable.
GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')

genai.configure(api_key=GOOGLE_API_KEY)
```

Now I am ready to call the Gemini API.

Use list_models to see the available Gemini models:

- gemini-1.5-pro: optimized for high intelligence tasks, the most powerful Gemini model
- gemini-1.5-flash: optimized for multi-modal use-cases where speed and cost are important

The genai package also supports the PaLM family of models, but only the Gemini models support the generic, multimodal capabilities of the generateContent method.

```
for m in genai.list_models():
  if 'generateContent' in m.supported_generation_methods:
    print(m.name)
```

```
models/gemini-1.0-pro
models/gemini-1.0-pro-001
models/gemini-1.0-pro-latest
models/gemini-1.0-pro-vision-latest
models/gemini-1.5-flash
models/gemini-1.5-flash-001
models/gemini-1.5-flash-latest
models/gemini-1.5-pro
models/gemini-1.5-pro-001
models/gemini-1.5-pro-latest
models/gemini-pro
models/gemini-pro-vision
```

Generate text from text inputs

For text-only prompts, use the gemini-pro model:

```
model = genai.GenerativeModel('gemini-1.5-flash')
```

The generate_content method can handle a wide variety of use cases, including multi-turn chat and multimodal input, depending on what the underlying model supports. The available models only support text and images as input, and text as output.

In the simplest case, one can pass a prompt string to the GenerativeModel.generate_content method:

```
%%time
response = model.generate_content("What is the meaning of life?")
```

```
CPU times: user 80.8 ms, sys: 11.1 ms, total: 91.9 ms
Wall time: 6.28 s
```

In simple cases, the response.text accessor is all I need. To display formatted Markdown

text, use the to_markdown function:

```
to_markdown(response.text)
```

The meaning of life is a question that has been pondered by philosophers, theologians, and everyday people for centuries. There is no single, universally accepted answer, and it's a question that each individual must ultimately answer for themselves.

Here are some perspectives:

Philosophical Views:

- **Nihilism:** There is no inherent meaning or purpose in life.
- **Existentialism:** Individuals are free to create their own meaning and purpose.
- **Absurdism:** Life is inherently meaningless, but we can still find joy and fulfillment in embracing this absurdity.
- **Hedonism:** The pursuit of pleasure and happiness is the ultimate goal of life.
- **Utilitarianism:** The greatest good for the greatest number of people should be the guiding principle.

Religious Views:

- **Theism:** Life has meaning and purpose as defined by a higher power, often through service to that power or to humanity.
- **Buddhism:** The ultimate goal of life is to achieve enlightenment and escape the cycle of suffering.

Personal Perspectives:

- **Love and connection:** Finding meaningful relationships with others can give life purpose.
- **Contribution:** Making a positive impact on the world can bring a sense of fulfillment.
- **Creativity and self-expression:** Expressing oneself through art, music, writing, or other creative pursuits can be deeply meaningful.
- **Personal growth and learning:** Continuously expanding one's knowledge and understanding can be a source of purpose.

If the API failed to return a result, use `GenerateContentResponse.prompt_feedback` to see if it was blocked due to safety concerns regarding the prompt.

```
response.prompt_feedback
```

Gemini can generate multiple possible responses for a single prompt.

These possible responses are called candidates, and one can review them to select the most suitable one as the response. View the response candidates with `GenerateContentResponse.candidates`:

```
response.candidates
```

[content {
  parts {
    text: "The meaning of life is a question that has been pondered by philosophers, theologians, and everyday people for centuries. There is no single, universally accepted answer, and it\'s a question that each individual must ultimately answer for themselves. \n\nHere are some perspectives:\n\n**Philosophical Views:**\n\n* **Nihilism:** There is no inherent meaning or purpose in life.\n* **Existentialism:** Individuals are free to create their own meaning and purpose.\n* **Absurdism:** Life is inherently meaningless, but we can still find joy and fulfillment in embracing this absurdity.\n* **Hedonism:** The pursuit of pleasure and happiness is the ultimate goal of life.\n* **Utilitarianism:** The greatest good for the greatest number of people should be the guiding principle.\n\n**Religious Views:**\n\n* **Theism:** Life has meaning and purpose as defined by a higher power, often through service to that power or to humanity.\n* **Buddhism:** The ultimate goal of life is to achieve enlightenment and escape the cycle of suffering.\n\n**Personal Perspectives:**\n\n* **Love and connection:** Finding meaningful relationships with others can give life purpose.\n* **Contribution:** Making a positive impact on the world can bring a sense of fulfillment.\n* **Creativity and self-expression:** Expressing oneself through art, music, writing, or other creative pursuits can be deeply meaningful.\n* **Personal growth and learning:** Continuously expanding one\'s knowledge and understanding can be a source of purpose.\n* **Experiencing life:** Embracing the joys and challenges of life, appreciating its beauty and fragility.\n\nUltimately, the meaning of life is what you make of it. There is no right or wrong answer, and the journey of discovering your own meaning is a personal and ongoing one. \n\nIt\'s important to note that I am an AI, and I cannot provide you with a definitive answer to this question. However, I can offer you different perspectives and encourage you to explore your own thoughts and feelings about the meaning of life. \n"
  }
  role: "model"
}
finish_reason: STOP
index: 0
safety_ratings {
  category: HARM_CATEGORY_SEXUALLY_EXPLICIT
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_HATE_SPEECH
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_HARASSMENT
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_DANGEROUS_CONTENT
  probability: NEGLIGIBLE
}
]

By default, the model returns a response after completing the entire generation process.

One can also stream the response as it is being generated, and the model will return chunks of the response as soon as they are generated. To stream responses, use `GenerativeModel.generate_content(..., stream=True)`.

```
%%time
response = model.generate_content("What is the meaning of life?", stream=True)
```

```
CPU times: user 84 ms, sys: 9.71 ms, total: 93.7 ms
Wall time: 7.44 s
```

```
for chunk in response:
  print(chunk.text)
  print("_"*80)
```

```
As
________________________________________________________________________________
 a large language model, I don't have personal beliefs or experiences, so
________________________________________________________________________________
 I can't tell you what the meaning of life is. However, I
________________________________________________________________________________
 can tell you that philosophers and theologians have been debating this qu

Here are some perspectives on the meaning
________________________________________________________________________________
 of life:
```

```
**Philosophical Perspectives:**

* **Nihilism:**  Life is inherently meaningless. There is no objective purp
*
_____________________________________________________________________________
 **Existentialism:**  Life has no inherent meaning, but we are free to crea
* **Absurdism:**  The world is inherently absurd, but we can find meaning b
_____________________________________________________________________________
 embracing the absurdity and living life to the fullest.
* **Utilitarianism:**  The meaning of life is to maximize happiness for the
* **Hedonism:**  The meaning of life is to pursue
_____________________________________________________________________________
 pleasure and avoid pain.

**Religious Perspectives:**

* **Monotheistic Religions:** The meaning of life is to worship God and liv
* **Buddhism:** The meaning of life is to achieve enlightenment and escape
* **Hinduism:** The meaning of
_____________________________________________________________________________
 life is to achieve liberation from the cycle of rebirth and attain union w

**Personal Perspectives:**

Ultimately, the meaning of life is a personal question that each individua
_____________________________________________________________________________
 society or making a difference in the world.

**What I can do for you:**

* I can share philosophical and religious perspectives on the meaning of li
* I can help you explore different ideas and concepts related to the meanin
* I can help you think critically about your own beliefs and
_____________________________________________________________________________
 values.

Remember, there is no right or wrong answer to the question of the meaning

_____________________________________________________________________________
```

```
response = model.generate_content("What is the meaning of life?", stream=True)
```

```
response.prompt_feedback
```

```
try:
  response.text
except Exception as e:
  print(f'{type(e).__name__}: {e}')
```

```
IncompleteIterationError: Please let the response complete iteration before
attributes (or call `response.resolve()`)
```

Generate text from image and text inputs

The `GenerativeModel.generate_content` API is designed to handle multimodal prompts and returns a text output. Let include an image:

```
!curl -o image.jpg https://t0.gstatic.com/licensed-image?q=tbn:ANd9GcQ_Kevbk21
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
                                 Dload  Upload   Total   Spent    Left  Sp
100  405k  100  405k    0     0  4674k      0 --:--:-- --:--:-- --:--:-- 4
```

```
import PIL.Image

img = PIL.Image.open('image.jpg')
img
```

Use the `gemini-1.5-flash` model and pass the image to the model with `generate_content`.

```
model = genai.GenerativeModel('gemini-1.5-flash')
```

```
response = model.generate_content(img)

to_markdown(response.text)
```

The image contains two containers of food. One container has chicken, broccoli, carrots, and rice. The other container has chicken, broccoli, and rice. Both containers have sesame seeds on top. There is a small bowl of sesame seeds to the left of the containers and a bottle of soy sauce in the background. There are two chopsticks in the bottom left corner. The background is a grey surface. The image shows a healthy and

To provide both text and images in a prompt, pass a list containing the strings and images:

```
response = model.generate_content(["Write a short, engaging blog post based on
response.resolve()

to_markdown(response.text)
```

From Chaos to Calm: My Meal Prep Journey

This picture? This is my happy place. A week's worth of delicious, healthy meals ready to go. It may seem simple, but it represents a huge change in my life.

> For years, I struggled with healthy eating. I'd be starving at work, grabbing whatever was convenient, and then feeling sluggish and guilty. Then I decided to try meal prepping. It started slowly, with just a few meals for the week. But gradually, I found myself planning out my meals, prepping ingredients, and cooking in bulk.
>
> And the results have been amazing! Not only am I eating healthier, but I'm saving time and money. This container holds a delicious Teriyaki Chicken Bowl, with fluffy rice, broccoli, bell peppers, and carrots - and it takes just minutes to heat up and enjoy. No more greasy takeout or last-minute grocery runs.
>
> Meal prepping is definitely an investment of time, but for me, it's been worth it. Now, I can focus on what matters most - living a healthier and happier life. And this photo? It's a reminder that good things come to those who prepare!

Chat conversations

Gemini enables you to have freeform conversations across multiple turns. The `ChatSession` class simplifies the process by managing the state of the conversation, so unlike with `generate_content`, I do not have to store the conversation history as a list. Initialize the chat:

```
model = genai.GenerativeModel('gemini-1.5-flash')
chat = model.start_chat(history=[])
chat
```

```
ChatSession(
    model=genai.GenerativeModel(
        model_name='models/gemini-1.5-flash',
        generation_config={},
        safety_settings={},
        tools=None,
        system_instruction=None,
    ),
    history=[]
)
```

The `ChatSession.send_message` method returns the same `GenerateContentResponse` type as `GenerativeModel.generate_content`. It also appends my message and the response to the chat history:

```
response = chat.send_message("In one long sentence, explain how a computer wor
to_markdown(response.text)
```

> A computer is like a super smart toy that you tell what to do by typing or clicking on the screen, and it uses special numbers and pictures to follow your instructions and do amazing things like show you pictures, play music, and even talk to people far away!

```
chat.history
```

```
[parts {
    text: "In one sentence, explain how a computer works to a young child."
  }
  role: "user",
  parts {
    text: "A computer is like a super smart toy that follows instructions
you give it, using numbers and pictures to do amazing things! \n"
  }
  role: "model",
  parts {
    text: "In one long sentence, explain how a computer works to a young
child."
  }
  role: "user",
  parts {
    text: "A computer is like a super smart toy that you give instructions
to using buttons and a screen, and it uses special numbers and pictures to
follow your instructions and do amazing things like show you videos, play
games, and even talk to people far away! \n"
  }
  role: "model",
  parts {
    text: "In one long sentence, explain how a computer works to a young
child."
  }
  role: "user",
  parts {
    text: "A computer is like a super smart toy that you tell what to do by
typing or clicking on the screen, and it uses special numbers and pictures
to follow your instructions and do amazing things like show you pictures,
play music, and even talk to people far away! \n"
  }
  role: "model"]
```

One can keep sending messages to continue the conversation. Use the `stream=True` argument to stream the chat:

```
response = chat.send_message("Okay, how about a more detailed explanation to a

for chunk in response:
  print(chunk.text)
  print("_"*80)
```

```
Think
_____
 of a computer as a super-powered brain that follows instructions written i
_____
 language called code.  This code is like a recipe, outlining step-by
_____
-step what the computer should do.  Instead of using ingredients, the compu
_____
.  This data is stored in the computer's memory, like a giant recipe book.

When you give the computer an instruction, it translates the
_____
 code into electrical signals. These signals are like tiny messages that te
_____
 the data in its memory according to the instructions.  It might perform ca

Finally, the computer translates the results back into a form we can unders
_____
 displayed on the screen, sounds played through speakers, or even actions t

So, essentially, a computer takes instructions written in code, processes d
_____
 the computer seem like a magical machine that can do almost anything!

_____
```

`genai.protos.Content` objects contain a list of `genai.protos.Part` objects that each contain either a text (string) or inline_data (`genai.protos.Blob`), where a blob contains binary data and a `mime_type`. The chat history is available as a list of `genai.protos.Content` objects in `ChatSession.history`:

```
for message in chat.history:
  display(to_markdown(f'**{message.role}**: {message.parts[0].text}'))
```

user: In one sentence, explain how a computer works to a young child

user: In one sentence, explain how a computer works to a young child.

model: A computer is like a super smart toy that follows instructions you give it, using numbers and pictures to do amazing things!

user: In one long sentence, explain how a computer works to a young child.

model: A computer is like a super smart toy that you give instructions to using buttons and a screen, and it uses special numbers and pictures to follow your instructions and do amazing things like show you videos, play games, and even talk to people far away!

user: In one long sentence, explain how a computer works to a young child.

model: A computer is like a super smart toy that you tell what to do by typing or clicking on the screen, and it uses special numbers and pictures to follow your instructions and do amazing things like show you pictures, play music, and even talk to people far away!

user: Okay, how about a more detailed explanation to a high schooler?

model: A computer fundamentally works by processing information, taking instructions in the form of code, converting them into electrical signals, manipulating those signals according to the instructions, and then outputting the results as text, images, sounds, or actions in the real world.

user: Okay, how about a more detailed explanation to a high schooler?

model: At its core, a computer works by processing information. It takes instructions written in a language it understands, called code, and converts them into electrical signals. These signals are then manipulated based on the instructions, performing calculations, moving data, and controlling various components. Finally, the results of these manipulations are translated back into something we can understand, like text displayed on a screen, sounds played through speakers, or even physical actions controlled by connected devices. Think of it like a super fast, complex calculator that can do a whole lot more than just add numbers!

user: Okay, how about a more detailed explanation to a high schooler?

model: Imagine a computer as a giant, super-fast calculator that can understand and follow instructions written in a special language called code. You give the computer these instructions, which are essentially a set of steps telling it what to do. The computer translates these instructions into electrical signals, which it uses to manipulate data stored in its memory. This data can be anything from text and images to numbers and even instructions for controlling other devices. The computer then processes this data according to the instructions, performing calculations, comparing information, and making decisions. Finally, it translates the results back into something we can understand, like displaying text on the screen, playing music through speakers, or even controlling the movement of a robotic arm. Essentially, a computer is a powerful tool that can process and manipulate information in incredibly complex ways, allowing it to perform a wide range of tasks based on the instructions it receives.

user: Okay, how about a more detailed explanation to a high schooler?

model: Think of a computer as a super-powered brain that follows instructions written in a special language called code. This code is like a recipe, outlining step-by-step what the computer should do. Instead of using ingredients, the computer works with data, which can be anything from numbers and letters to images and sounds. This data is stored in the computer's memory, like a giant recipe book.

When you give the computer an instruction, it translates the code into electrical signals. These signals are like tiny messages that tell different parts of the computer what to do. The computer's central processing unit (CPU), which is like the brain of the computer, receives these signals and manipulates the data in its memory according to the

Count tokens

Large language models have a context window, and the context length is often measured in terms of the number of tokens. With the Gemini API, one can determine the number of tokens per any `genai.protos.Content` object. In the simplest case, you can pass a query string to the `GenerativeModel.count_tokens` method as follows:

```
model.count_tokens("What is the meaning of life?")
```

```
total_tokens: 7
```

```
model.count_tokens(chat.history)
```

```
total_tokens: 867
```

Use Embeddings

Embedding is a technique used to represent information as a list of floating point numbers in an array.

- With Gemini, one can represent text (words, sentences, and blocks of text) in a vectorized form, making it easier to compare and contrast embeddings.
- For example, two texts that share a similar subject matter or sentiment should have similar embeddings, which can be identified through mathematical comparison techniques such as cosine similarity.

Use the `embed_content` method to generate embeddings. The method handles embedding for the following tasks (`task_type`):

Task Type	Description
RETRIEVAL_QUERY	Specifies the given text is a query in a search/retrieval setting.
RETRIEVAL_DOCUMENT	Specifies the given text is a document in a search/retrieval setting. Using this task type requires a
SEMANTIC_SIMILARITY	Specifies the given text will be used for Semantic Textual Similarity (STS).

CLASSIFICATION	Specifies that the embeddings will be used for classification.
CLUSTERING	Specifies that the embeddings will be used for clustering.

The following generates an embedding for a single string for document retrieval:

```
result = genai.embed_content(
    model="models/text-embedding-004",
    content="What is the meaning of life?",
    task_type="retrieval_document",
    title="Embedding of single string")

# 1 input > 1 vector output
print(str(result['embedding'])[:50], '... TRIMMED]')
```

```
[-0.028545432, 0.044588123, -0.03419736, -0.004266 ... TRIMMED]
```

Note: The `retrieval_document` task type is the only task that accepts a title. To handle batches of strings, pass a list of strings in content:

```
result = genai.embed_content(
    model="models/text-embedding-004",
    content=[
      'What is the meaning of life?',
      'How much wood would a woodchuck chuck?',
      'How does the brain work?'],
    task_type="retrieval_document",
    title="Embedding of list of strings")

# A list of inputs > A list of vectors output
for v in result['embedding']:
  print(str(v)[:50], '... TRIMMED ...')
```

```
[-0.036453035, 0.03325499, -0.03970925, -0.0026286 ... TRIMMED ...
[-0.01591948, 0.032582667, -0.081024624, -0.011298 ... TRIMMED ...
[0.00037063262, 0.03763057, -0.12269569, -0.009518 ... TRIMMED ...
```

While the `genai.embed_content` function accepts simple strings or lists of strings, it is actually built around the `genai.protos.Content` type (like `GenerativeModel.generate_content`). `genai.protos.Content` objects are the primary units of conversation in the API. While the `genai.protos.Content` object is multimodal, the `embed_content method` only supports text embeddings. This design gives the API the possibility to expand to multimodal embeddings.

```
response.candidates[0].content
```

```
parts {
  text: "Think of a computer as a super-powered brain that follows
instructions written in a special language called code.  This code is like
a recipe, outlining step-by-step what the computer should do.  Instead of
using ingredients, the computer works with data, which can be anything
from numbers and letters to images and sounds.  This data is stored in the
computer\'s memory, like a giant recipe book.  \n\nWhen you give the
computer an instruction, it translates the code into electrical signals.
These signals are like tiny messages that tell different parts of the
computer what to do.  The computer\'s central processing unit (CPU), which
is like the brain of the computer, receives these signals and manipulates
the data in its memory according to the instructions.  It might perform
calculations, compare information, or rearrange the data in different
ways.  \n\nFinally, the computer translates the results back into a form
we can understand. This could be text displayed on the screen, sounds
played through speakers, or even actions taken by connected devices like a
robot arm.  \n\nSo, essentially, a computer takes instructions written in
code, processes data according to those instructions, and then outputs the
results in a way that we can understand.  This process happens incredibly
fast, making the computer seem like a magical machine that can do almost
anything! \n"
}
role: "model"
```

```
result = genai.embed_content(
    model = 'models/text-embedding-004',
    content = response.candidates[0].content)

# 1 input > 1 vector output
print(str(result['embedding'])[:50], '... TRIMMED ...')
```

```
[-0.0058264174, 0.030309783, -0.022210436, -0.0212 ... TRIMMED ...
```

Similarly, the chat history contains a list of `genai.protos.Content` objects, which you can pass directly to the `embed_content` function:

```
chat.history
```

```
follow your instructions and do amazing things like show you videos, play
games, and even talk to people far away! \n"
 }
 role: "model",
 parts {
   text: "In one long sentence, explain how a computer works to a young
child."
 }
 role: "user",
 parts {
   text: "A computer is like a super smart toy that you tell what to do by
typing or clicking on the screen, and it uses special numbers and pictures
to follow your instructions and do amazing things like show you pictures,
play music, and even talk to people far away! \n"
 }
 role: "model",
```

```
      parts {
        text: "Okay, how about a more detailed explanation to a high schooler?"
      }
      role: "user",
      parts {
        text: "A computer fundamentally works by processing information, taking
      instructions in the form of code, converting them into electrical signals,
      manipulating those signals according to the instructions, and then
      outputting the results as text, images, sounds, or actions in the real
      world. \n"
      }
      role: "model",
      parts {
        text: "Okay, how about a more detailed explanation to a high schooler?"
      }
      role: "user",
      parts {
        text: "At its core, a computer works by processing information. It
      takes instructions written in a language it understands, called code, and
      converts them into electrical signals. These signals are then manipulated
      based on the instructions, performing calculations, moving data, and
      controlling various components. Finally, the results of these
      manipulations are translated back into something we can understand, like
      text displayed on a screen, sounds played through speakers, or even
      physical actions controlled by connected devices.  Think of it like a
      super fast, complex calculator that can do a whole lot more than just add
      numbers! \n"
      }
      role: "model",
      parts {
        text: "Okay, how about a more detailed explanation to a high schooler?"
      }
      role: "user",
      parts {
        text: "Imagine a computer as a giant, super-fast calculator that can
      understand and follow instructions written in a special language called
      code. You give the computer these instructions, which are essentially a
      set of steps telling it what to do.  The computer translates these
      instructions into electrical signals, which it uses to manipulate data
      stored in its memory.  This data can be anything from text and images to
      numbers and even instructions for controlling other devices.  The computer
      then processes this data according to the instructions, performing
      calculations, comparing information, and making decisions.  Finally, it
      translates the results back into something we can understand, like
```

```
result = genai.embed_content(
    model = 'models/text-embedding-004',
    content = chat.history)

# 1 input > 1 vector output
for i,v in enumerate(result['embedding']):
  print(str(v)[:50], '... TRIMMED...')
```

```
[-0.03417175, 0.005179209, -0.054957863, -0.030872 ... TRIMMED...
[-0.04343897, 0.016338121, -0.010294692, -0.014275 ... TRIMMED...
[-0.030322473, 0.0057153706, -0.05012561, -0.02536 ... TRIMMED...
[-0.039027598, 0.015468359, -0.027367886, -0.00709 ... TRIMMED...
[-0.030322473, 0.0057153706, -0.05012561, -0.02536 ... TRIMMED...
[-0.04381359, 0.020128509, -0.016500289, -0.012888 ... TRIMMED...
[0.032410417, 0.036596797, -0.04735078, -0.0301925 ... TRIMMED...
```

```
[0.032410417, 0.036596797, -0.04735078, -0.0301925 ... TRIMMED...
[-0.04123756, 0.030380573, -0.045898404, -0.020626 ... TRIMMED...
[0.032410417, 0.036596797, -0.04735078, -0.0301925 ... TRIMMED...
[-0.0059954263, 0.028068282, -0.040530898, -0.0235 ... TRIMMED...
[0.032410417, 0.036596797, -0.04735078, -0.0301925 ... TRIMMED...
[0.0033746867, 0.024427418, -0.024487834, -0.02908 ... TRIMMED...
[0.032410417, 0.036596797, -0.04735078, -0.0301925 ... TRIMMED...
[-0.0058264174, 0.030309783, -0.022210436, -0.0212 ... TRIMMED...
```

Advanced Use Cases

The following sections discuss advanced use cases and lower-level details of the Python SDK for the Gemini API.

Safety settings

1. The `safety_settings` argument lets you configure what the model blocks and allows in both prompts and responses.
2. By default, safety settings block content with medium and/or high probability of being unsafe content across all dimensions.
3. Enter a questionable prompt and run the model with the default safety settings, and it will not return any candidates:

```
response = model.generate_content('[Questionable prompt here]')
response.candidates
```

```
[content {
  parts {
    text: "Please provide me with the prompt you\'d like me to evaluate.
I\'m here to help you understand if a prompt is appropriate and how it
might be improved. \n\nRemember, I am designed to be a helpful and
harmless AI assistant. I cannot provide responses that are discriminatory,
offensive, or harmful in any way. \n\nI\'m looking forward to assisting
you! \n"
  }
  role: "model"
}
finish_reason: STOP
index: 0
safety_ratings {
```

```
    category: HARM_CATEGORY_SEXUALLY_EXPLICIT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_HATE_SPEECH
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_HARASSMENT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_DANGEROUS_CONTENT
    probability: NEGLIGIBLE
  }
```

The `prompt_feedback` will tell you which safety filter blocked the prompt:

```
response.prompt_feedback
```

Now provide the same prompt to the model with newly configured safety settings, and one may get a response.

```
response = model.generate_content('[Questionable prompt here]',
                                  safety_settings={'HARASSMENT':'block_none'})
response.text
```

```
'Please provide me with the prompt you'd like me to evaluate. I'm here to help you determine if a prompt is questionable and discuss potential issues. \n\nI understand that you may be hesitant to share the prompt directly. If you'd prefer, you can share some details about the prompt, like:\n\n* **The topic of the prompt:** What is the prompt asking you to generate?\n* **The potential risks:** What are your concerns about the prompt? \n* **The desired outcome:** What kind of response are you hoping to get from the prompt?\n\nBased on this information, I can help you assess the prompt's p
```

Also note that each candidate has its own `safety_ratings`, in case the prompt passes but the individual responses fail the safety checks.

Encode messages

The previous sections relied on the SDK to make it easy for you to send prompts to the API. This section offers a fully-typed equivalent to the previous example, so you can better understand the lower-level details regarding how the SDK encodes messages.

The `google.generativeai.protos` submodule provides access to the low level classes

used by the API behind the scenes: The SDK attempts to convert your message to a `genai.protos.Content` object, which contains a list of `genai.protos.Part` objects that each contain either:

1. a text (`string`)
2. `inline_data` (`genai.protos.Blob`), where a blob contains binary data and
3. a `mime_type`.

You can also pass any of these classes as an equivalent dictionary.

Note: The only accepted mime types are some image types, `image/*`.

So, the `fully-typed` equivalent to the previous example is:

```
model = genai.GenerativeModel('gemini-1.5-flash')
response = model.generate_content(
    genai.protos.Content(
        parts = [
            genai.protos.Part(text="Write a short, engaging blog post based on
            genai.protos.Part(
                inline_data=genai.protos.Blob(
                    mime_type='image/jpeg',
                    data=pathlib.Path('image.jpg').read_bytes()
                )
            ),
        ],
    ),
    stream=True)

response.resolve()

to_markdown(response.text[:100] + "... [TRIMMED] ...")
```

Lunchtime Bliss: Teriyaki Chicken & Veggies

Forget the boring sandwich! This teriyaki chicken a... [TRIMMED] ...

Multi-turn conversations

- While the `genai.ChatSession` class shown earlier can handle many use cases, it does make some assumptions.
- If one's use case doesn't fit into this chat implementation it's good to remember that `genai.ChatSession` is just a wrapper around `GenerativeModel.generate_content`.
- In addition to single requests, it can handle `multi-turn` conversations.
- The individual messages are `genai.protos.Content` objects or compatible dictionaries.

- The individual messages are genai.protos.Content objects or compatible dictionaries, as seen in previous sections.
- As a dictionary, the message requires role and parts keys. The role in a conversation can either be the user, which provides the prompts, or model, which provides the responses.
- Pass a list of genai.protos.Content objects and it will be treated as `multi-turn` chat:

```
model = genai.GenerativeModel('gemini-1.5-flash')

messages = [
    {'role':'user',
     'parts': ["Briefly explain how a computer works to a young child."]}
]
response = model.generate_content(messages)

to_markdown(response.text)
```

Imagine a computer is like a super smart friend who loves playing games.

- **The brain:** The computer's brain is called the **CPU**, like your own brain, it thinks and solves problems.
- **The memory:** It has a big memory, like your notebook, where it remembers things it needs to do.
- **The keyboard:** You can talk to the computer using a **keyboard** to tell it what to do.
- **The screen:** The computer talks back to you using a **screen** to show you pictures and words.

When you type something on the keyboard, the CPU reads it and uses its memory to

To continue the conversation, add the response and another message.

Note: For multi-turn conversations, you need to send the whole conversation history with each request. The API is stateless.

```
messages.append({'role':'model',
                 'parts':[response.text]})

messages.append({'role':'user',
                 'parts':["Okay, how about a more detailed explanation to a hi

response = model.generate_content(messages)

to_markdown(response.text)
```

Okay, let's break down how a computer works in a more detailed way, focusing on the essential components:

1. Hardware: The Physical Foundation

- **CPU (Central Processing Unit):** The "brain" of the computer. Imagine it as a super-fast calculator that can execute billions of instructions per second. It's responsible for:
  - Fetching instructions from memory.
  - Decoding those instructions.
  - Executing the instructions (performing calculations, manipulating data).
  - Storing results back in memory.
- **RAM (Random Access Memory):** This is like the computer's short-term memory. It holds the data and instructions the CPU is currently working with. It's fast, but temporary - information in RAM is lost when the computer is turned off.
  - Imagine it as a notepad where the CPU quickly scribbles down notes it needs to remember while working.
- **Storage Devices (Hard Disk Drive, Solid State Drive):** These are the long-term memory of the computer. They permanently store your files, operating system, and programs.
  - The hard drive uses spinning platters to store data magnetically, while SSDs use flash memory chips, making them faster and quieter.
- **Input Devices:** These allow you to communicate with the computer:
  - **Keyboard:** Used for typing text and commands.
  - **Mouse:** Used for selecting items on the screen and controlling the cursor.
  - **Touchscreen:** Allows you to interact directly with the screen.
- **Output Devices:** These present the computer's results to you:
  - **Monitor:** Displays visuals and text.
  - **Speakers:** Produce sound.
  - **Printers:** Create physical copies of documents.

2. Software: The Instructions

- **Operating System (OS):** The foundation software that manages all the hardware and provides a user interface. Examples include Windows, macOS, and Linux.
  - It acts like a traffic cop, directing the flow of information between hardware components.
  - It provides a way for you to interact with the computer (using a mouse, keyboard, etc.).
- **Applications:** These are programs designed to perform specific tasks. Examples include web browsers, word processors, games, and more.
  - They leverage the operating system to access and utilize the computer's hardware.

How it Works: A Cycle of Instructions

1. **Input:** You provide instructions using the keyboard, mouse, or other input devices.
2. **Processing:** The CPU fetches the instructions from RAM, decodes them, and executes them.
3. **Output:** The results are displayed on the monitor, played through the speakers,

Generation configuration

The `generation_config` argument allows you to modify the generation parameters. Every prompt you send to the model includes parameter values that control how the model generates responses.

```
model = genai.GenerativeModel('gemini-1.5-flash')
response = model.generate_content(
    'Tell me a story about a magic backpack.',
    generation_config=genai.types.GenerationConfig(
        # Only one candidate for now.
        candidate_count=1,
        stop_sequences=['x'],
        max_output_tokens=20,
        temperature=1.0)
)
```

```
text = response.text

if response.candidates[0].finish_reason.name == "MAX_TOKENS":
    text += '...'

to_markdown(text)
```

Elara, a scrawny girl with a thirst for adventure, found the backpack tucked away in