Exploring the World of GenAI with Google AI Studio and Gemini

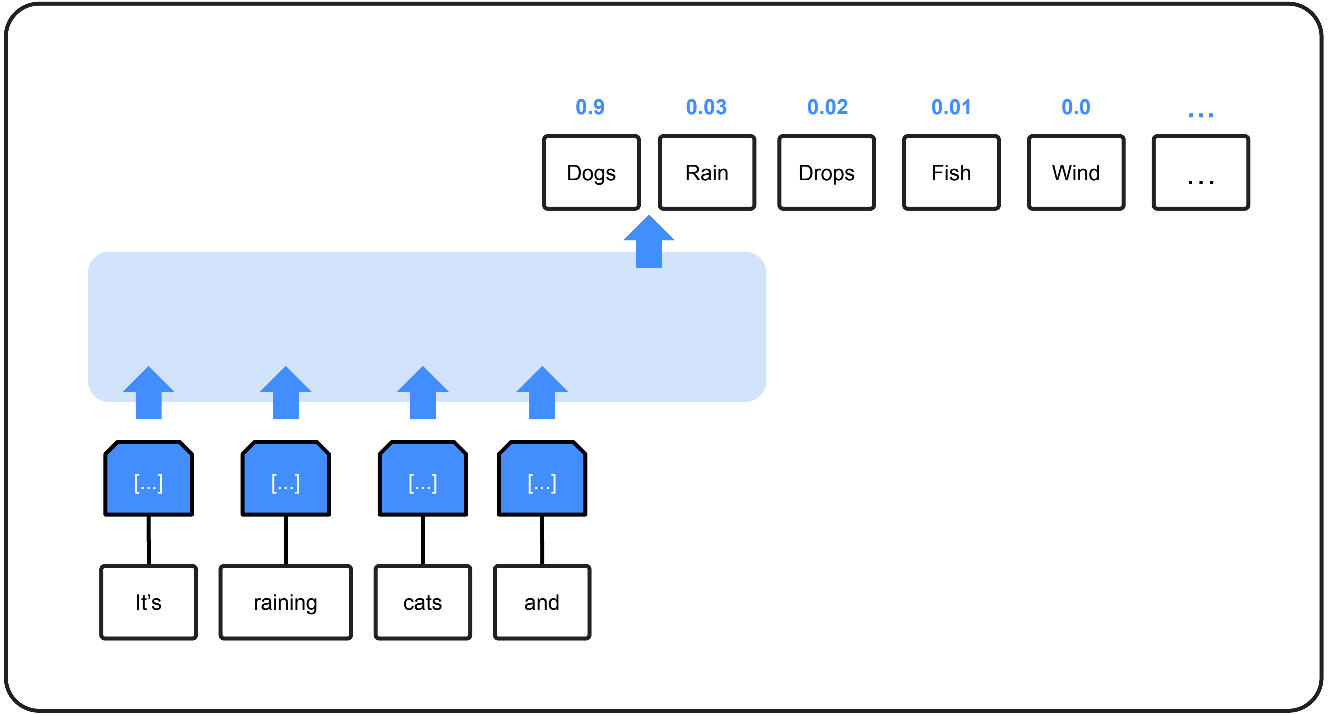
Suresh Peiris
Co-Founder at Inforwaves
GDG Organizer

Chamod Perera
SSE at Aleph Inc.
GDG Organizer & CNCF Ambassador

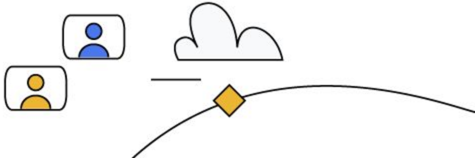
Sundar Pichai

A type of artificial intelligence that generates content for you.

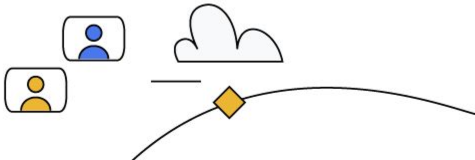
What is an LLM?



Roses are red,



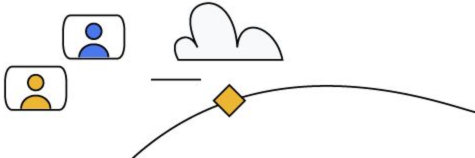
Roses are red,
Violets are blue,
Sugar is sweet,



```
for(var i = 0;
for(var i = 0; i < 10; i++) {
```

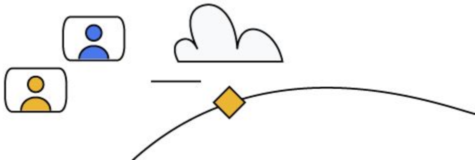
LLMs Explained

Modern LLMs are *large*

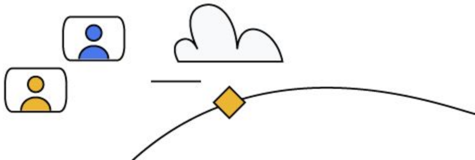


Classic Natural Language Problems

- Entity extraction
- Classification
- Summarization
- Sentiment Analysis
- Translation
- ...



LLMs let us prototype *fast*



User:

Explain this joke: “I used to be addicted to soap, but I'm clean now.”

Model:

Model: The joke is a pun on the word "clean". It has two meanings:

To be free of dirt or other unwanted substances.

To be free of addiction.

The joke is funny because it is unexpected. The listener expects the speaker to be talking about the first meaning of "clean", but they are actually talking about the second meaning. This creates a surprise and a laugh.

User:

Give me five neat ideas for science fair projects.

Model:

1. **Build a volcano.**

This is a classic science fair project that is both fun and educational. You can learn about the science of volcanoes, earthquakes, and plate tectonics.

2. **Make a solar oven.**

This is a great project to learn about renewable energy and how to harness the power of the sun. You can use your solar oven to cook food, dry fruit, or even melt chocolate.

3. **Create a 3D printer.**

This is a more challenging project, but it is also very rewarding. You can learn about 3D printing technology and design your own custom objects.

4. **Build a robot.**

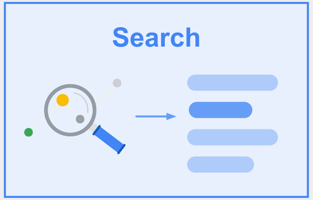
This is a great project for those who are interested in engineering and robotics. You can learn about basic electronics, mechanics, and programming.

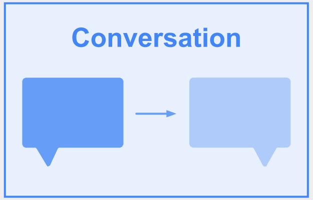
5. **Design a water filtration system.**

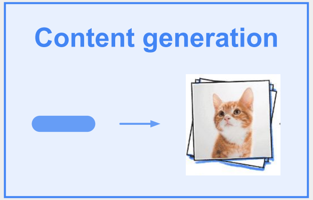
This is a project that can help you learn about the importance of clean water and how to filter it. You can use your water filtration system to clean water from a stream or river.

Why are large language models different?

- LLMs are characterized by **emergent abilities**, or the ability to perform tasks that were not present in smaller models.
- LLMs contextual understanding of human language **changes how we interact** with data and intelligent systems.
- LLMs can find patterns and connections in **massive, disparate data corpora.**



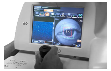




Google is the industry pioneer in AI

2015

Google DeepMind AlphaGo defeats Go champion



2016

Google's DeepMind helps detect eye disease

2017

Google invents Transformer kickstarting LLM revolution

2018

Google's groundbreaking large language model, BERT

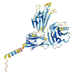
T5

2019

Text-to-Text Transfer Transformer LLM 10B P Model Open Sourced

2020

Google LaMDA Model Trained to converse



2022

AlphaFold predicts structures of all known proteins

2023

A conversational AI Service powered by PaLM2

2024

Family of multimodal LLMs & products

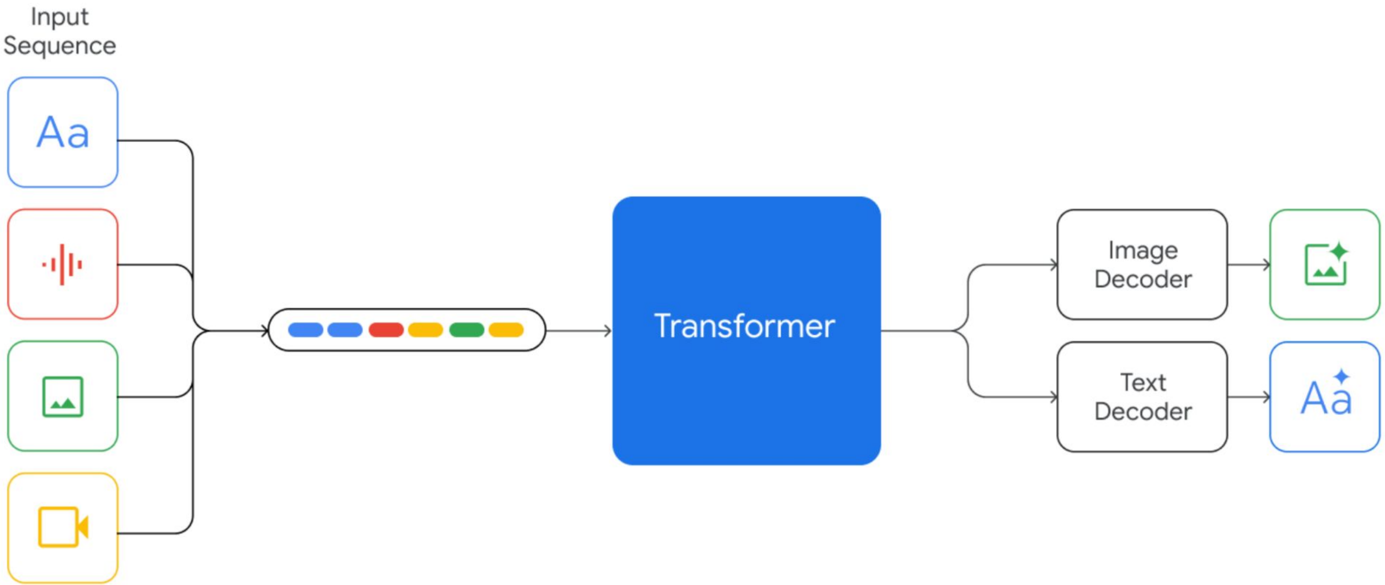
3,000
Researchers

7,000
Publications

Responsible AI

- ✓ Built & Tested for Safety
- ✓ Privacy in design
- ✓ Upholds high scientific standards
- ✓ Accountable to People
- ✓ Socially Beneficial
- ✓ Avoid creating unfair bias

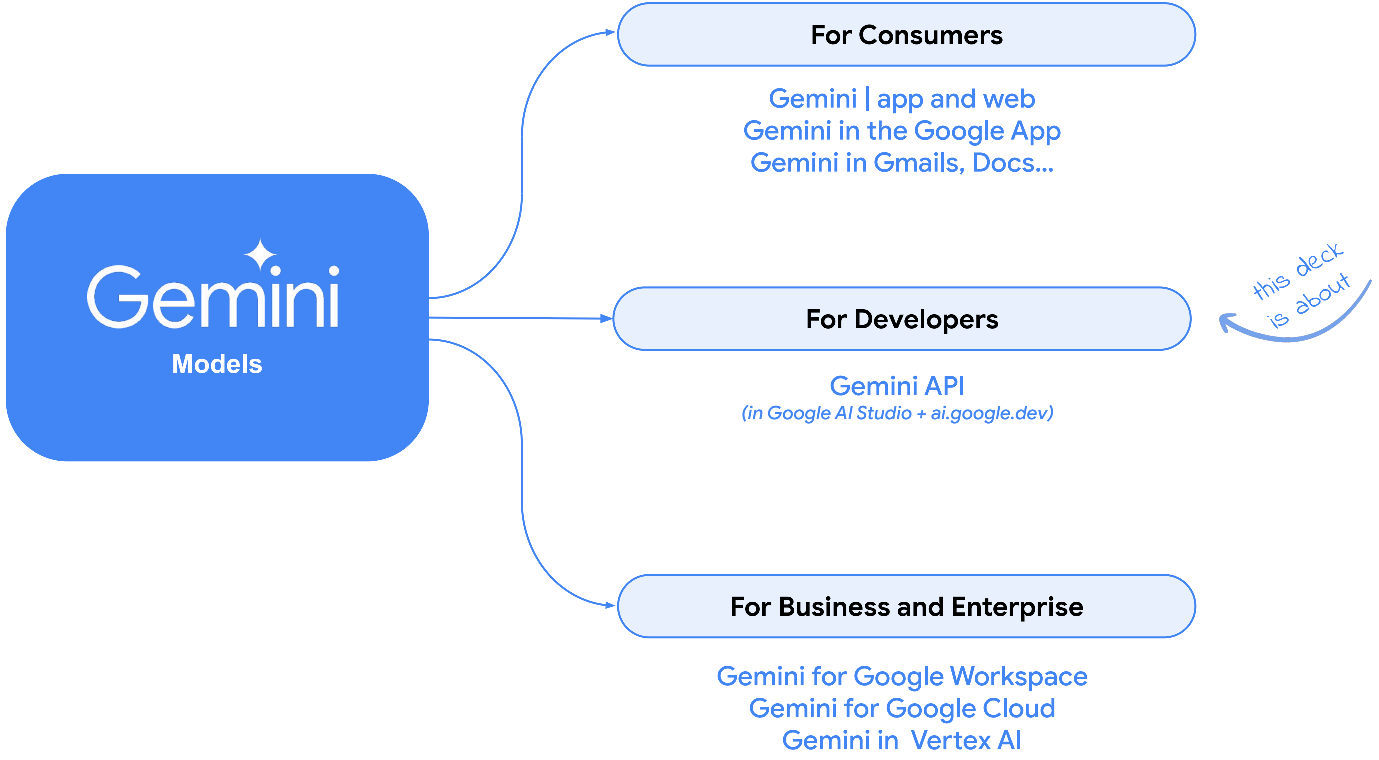
Multimodality



Welcome to the Gemini era

The Gemini Ecosystem

The most advanced AI from Google



Gemini

Models

For Consumers

Gemini | app and web
Gemini in the Google App
Gemini in Gmails, Docs...

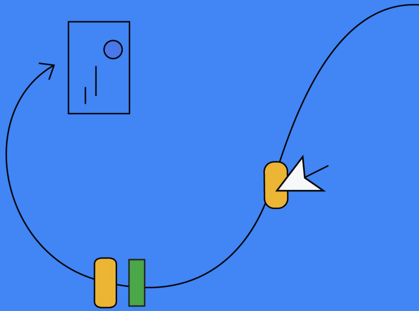
For Developers

this deck is about

Gemini API
(in Google AI Studio + ai.google.dev)

For Business and Enterprise

Gemini for Google Workspace
Gemini for Google Cloud
Gemini in Vertex AI



Getting started with the Gemini API

Get Started

Visit the cookbook: https://github.com/google-gemini/cookbook

Start developing:

1. Go to Google AI Studio.
2. Login with your Google account.
3. Create an API key.
4. Use a quickstart for Python, or call the REST API using curl.

SDKs

ai.google.dev

REST API + Client libraries for Python, Node, Java, and Swift

python™

node JS®

Swift

Java™ POWERED

Platforms

Vertex AI

Gemini API and **Vertex AI**

Both give access Gemini family models.

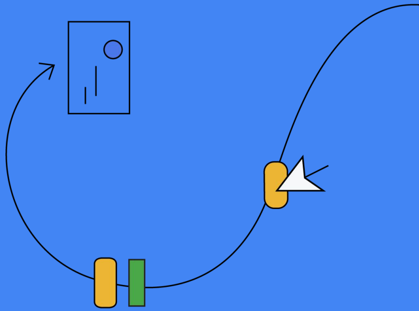
Vertex AI

Enterprise grade support.
Full MLOps (Examples: Model evaluation, monitoring, registry)

Vertex AI

Check it out when you're ready for production

ai.google.dev/docs/migrate_to_cloud



Usage Patterns

Usage Patterns

We’ll cover

- Basic Prompting
- Code generation
- Search, retrieval and information synthesis
- Function calling
- Image understanding
- Tuning

Resources

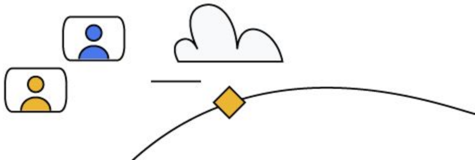
Resources

Gemini Cookbook

Codelab Resources

Usage Patterns

Basic Prompting



aistudio.google.com

Write a poem about a magic backpack.

In a world of wonder, behold,
A backpack with tales yet untold.
Its fabric shimmers, alive with might,
A portal to realms, day and night.

aistudio.google.com

Write a poem about a magic backpack.

In a world of wonder, behold,
A backpack with tales yet untold.
Its fabric shimmers, alive with might,
A portal to realms, day and night.
...

Generate a short tip about JavaScript. Good tips are short and contain a code example showing the tip.

Tip: Use Array.prototype.flat() to flatten an array of arrays.

```javascript
const array = [[1, 2], [3, 4], [5, 6]];
console.log(array.flat());

// Output: [1, 2, 3, 4, 5, 6]
```

Generate a short tip about JavaScript. Good tips are short and contain a code example showing the tip.

Tip: Use Array.prototype.flat() to flatten an array of arrays.

```javascript
const array = [[1, 2], [3, 4], [5, 6]];
console.log(array.flat());

// Output: [1, 2, 3, 4, 5, 6]
```

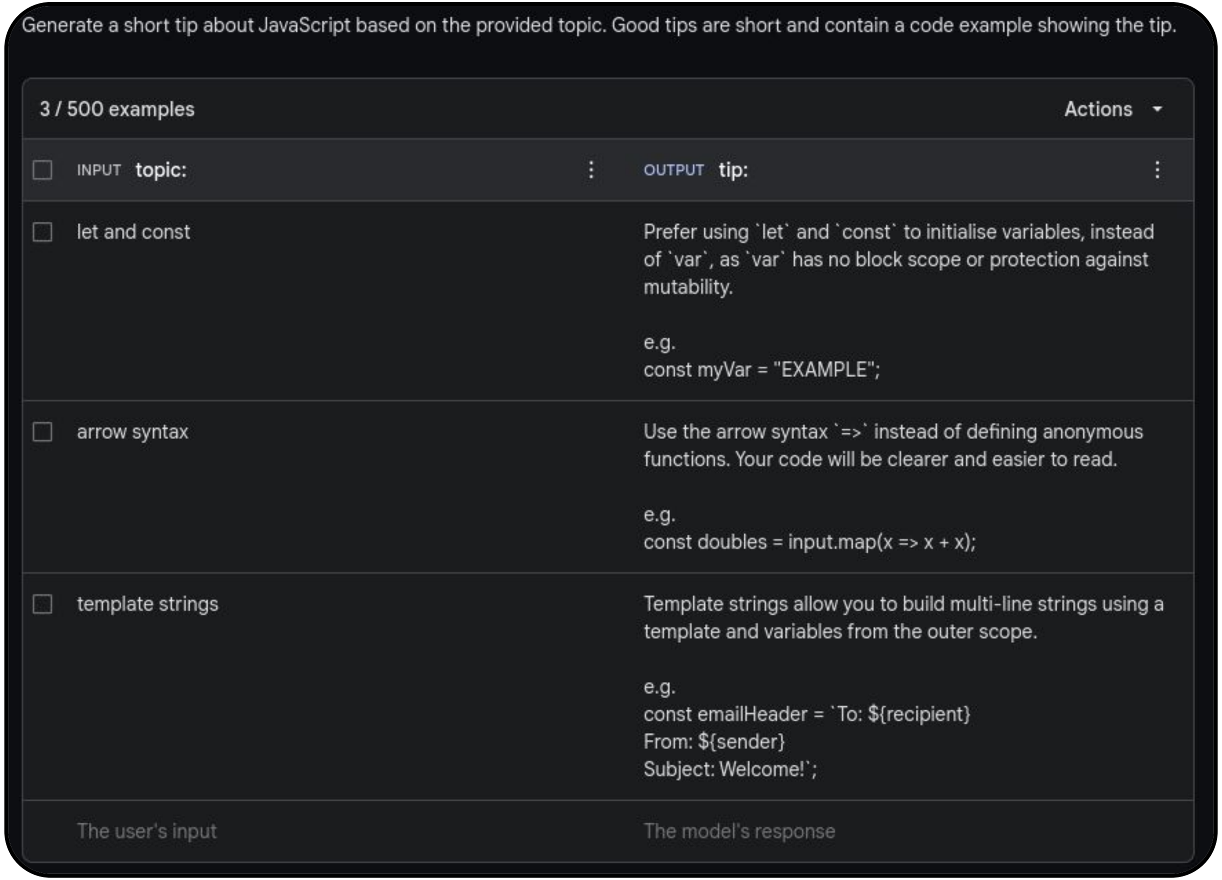
Generate a short tip about JavaScript. Good tips are short and contain a code example showing the tip.

topic: let and const
tip: Prefer using `let` and `const` to initialise variables, instead of `var`, as `var` has no block scope or protection against mutability.

e.g. const myVar = "EXAMPLE";

topic: arrow syntax
tip:

Prompting



Generate a short tip about JavaScript based on the provided topic. Good tips are short and contain a code example showing the tip.

3 / 500 examples

☐	INPUT topic:	OUTPUT tip:
☐	let and const	Prefer using \`let\` and \`const\` to initialise variables, instead of \`var\`, as \`var\` has no block scope or protection against mutability. e.g. const myVar = "EXAMPLE";
☐	arrow syntax	Use the arrow syntax \`=>\` instead of defining anonymous functions. Your code will be clearer and easier to read. e.g. const doubles = input.map(x => x + x);
☐	template strings	Template strings allow you to build multi-line strings using a template and variables from the outer scope. e.g. const emailHeader = \`To: ${recipient} From: ${sender} Subject: Welcome!\`;
	The user's input	The model's response

```
import google.generativeai as genai

model = genai.GenerativeModel('models/gemini-pro')

resp = model.generate_content(
  'Write the first paragraph of a story about a magic backpack')

>>> print(resp.text)
In a bustling city, amidst the vibrant tapestry of human
existence, there existed a peculiar entity named Archie. Archie,
however, was no ordinary backpack...
```

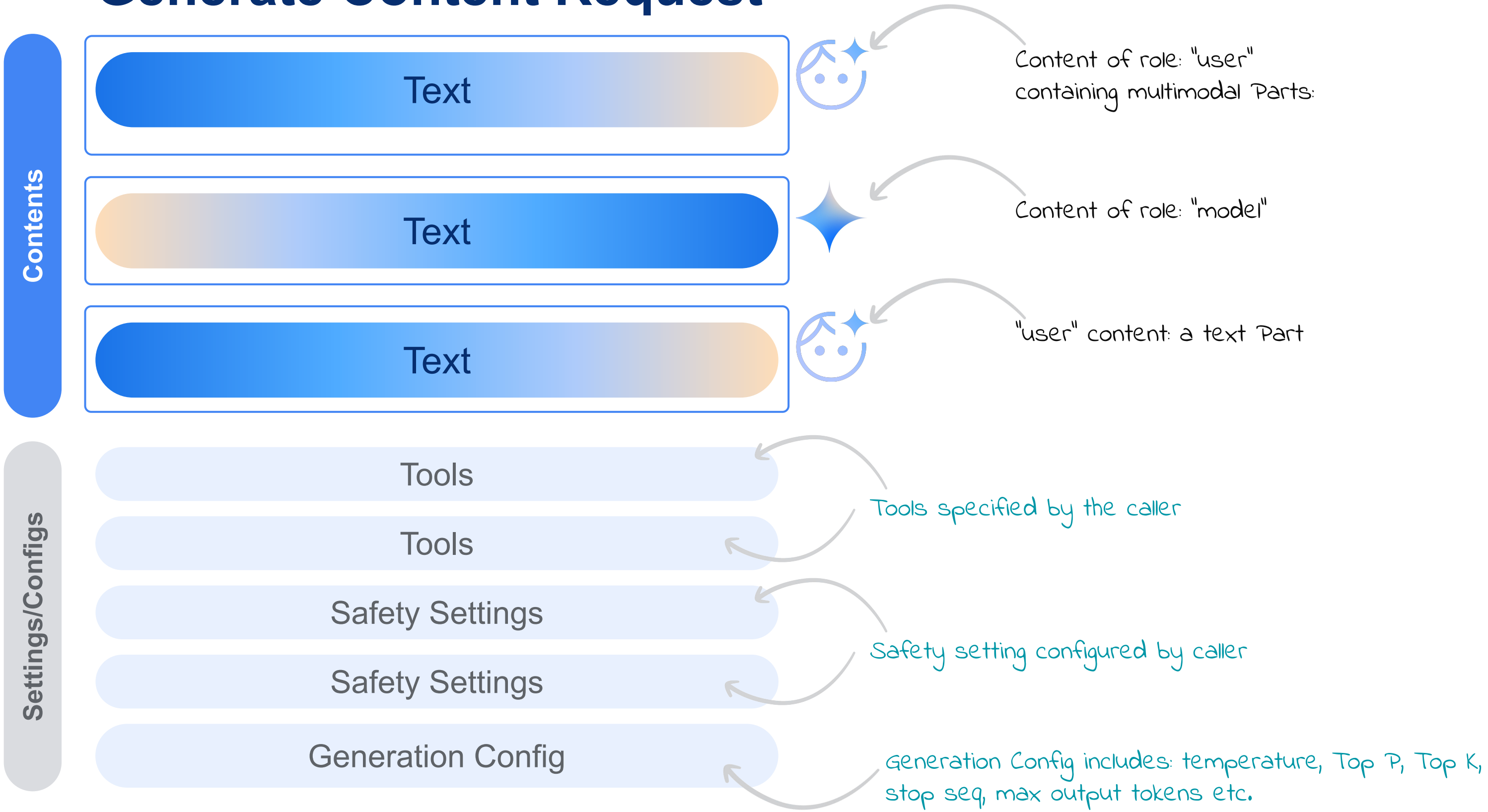
```
chat = model.start_chat()

response = chat.send_message(
    "Hello, what should I have for dinner?")

print(response.text)
    # 'Here are some suggestions...'

response = chat.send_message(
    "How do I cook the first one?")
```

Generate Content Request



Content of role: "user" containing multimodal Parts:

Content of role: "model"

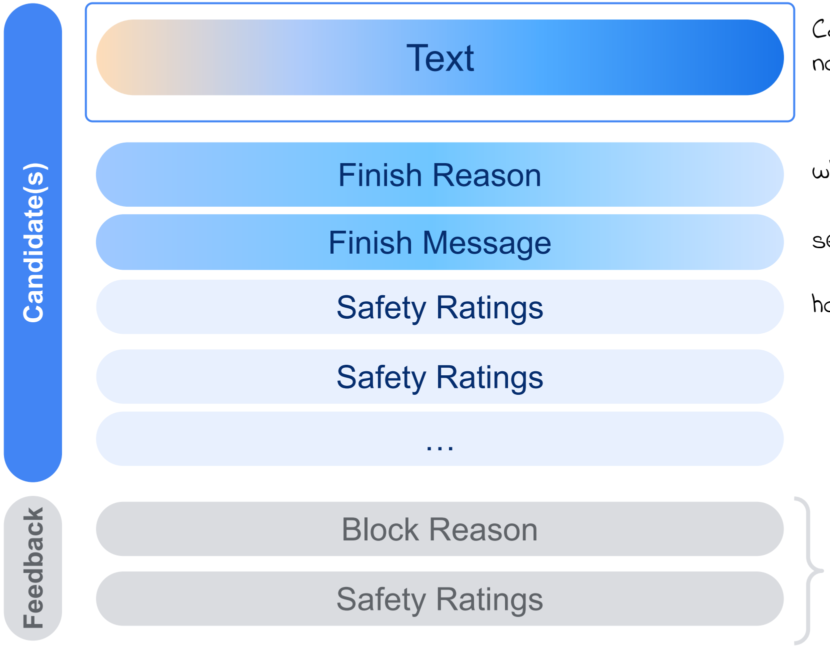
"user" content: a text Part

Tools specified by the caller

Safety setting configured by caller

Generation Config includes: temperature, Top P, Top K, stop seq, max output tokens etc.

Generate Content Response



Candidate(s)

Text

Finish Reason

Finish Message

Safety Ratings

Safety Ratings

…

Feedback

Block Reason

Safety Ratings

Candidate "Content"
note: only one candidate returned today

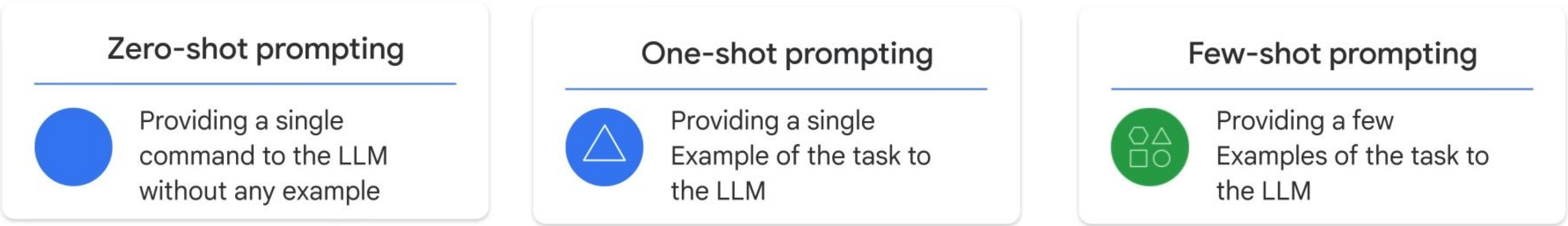
why the model stopped generating

set if finish reason is present

how safe is the response

feedback on the prompt

Prompting



Zero-shot prompting

Providing a single command to the LLM without any example

One-shot prompting

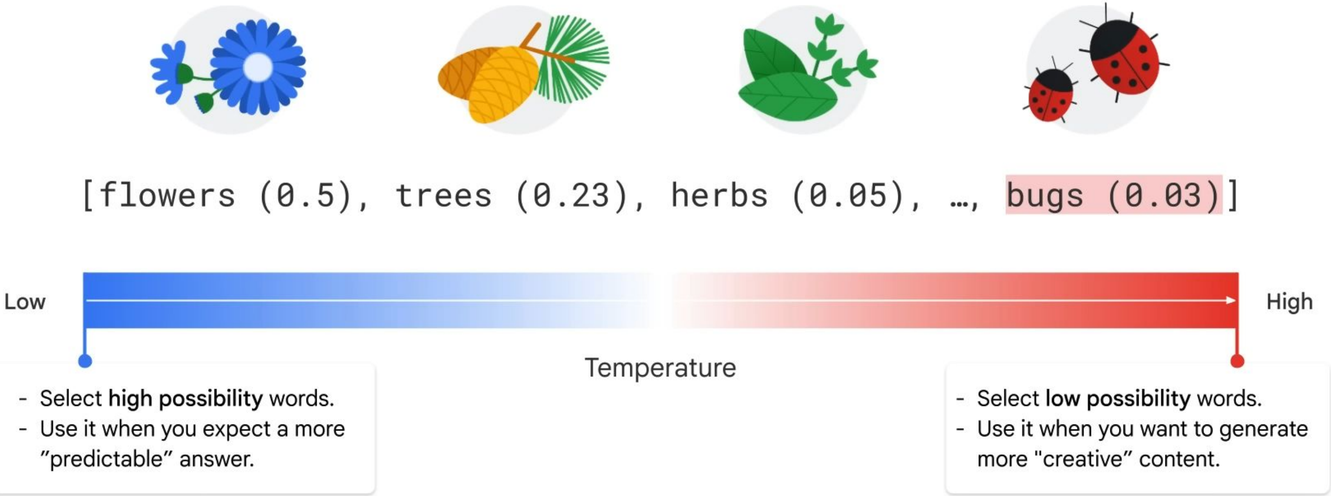
Providing a single Example of the task to the LLM

Few-shot prompting

Providing a few Examples of the task to the LLM

https://ai.google.dev/docs/prompt_best_practices

Prompting



`[flowers (0.5), trees (0.23), herbs (0.05), …, bugs (0.03)]`

- Select **high possibility** words.
- Use it when you expect a more "predictable" answer.

- Select **low possibility** words.
- Use it when you want to generate more "creative" content.

Usage Patterns

Code Generation

Content generation

Code Generation

Generate code based on a user prompt

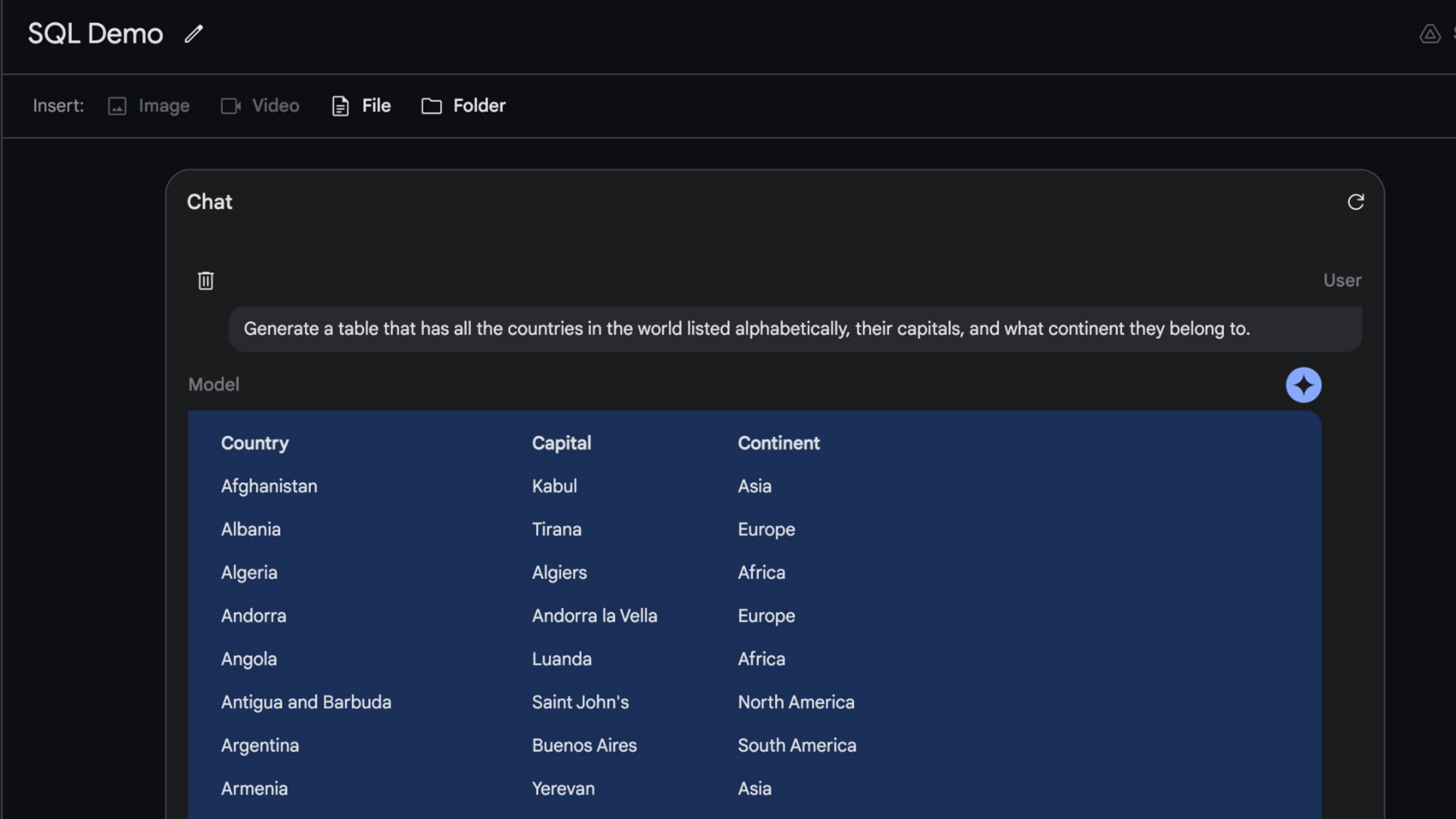
Convert human language requests to machine requests

Rapid tool building

Code generation

Examples

- Generate data



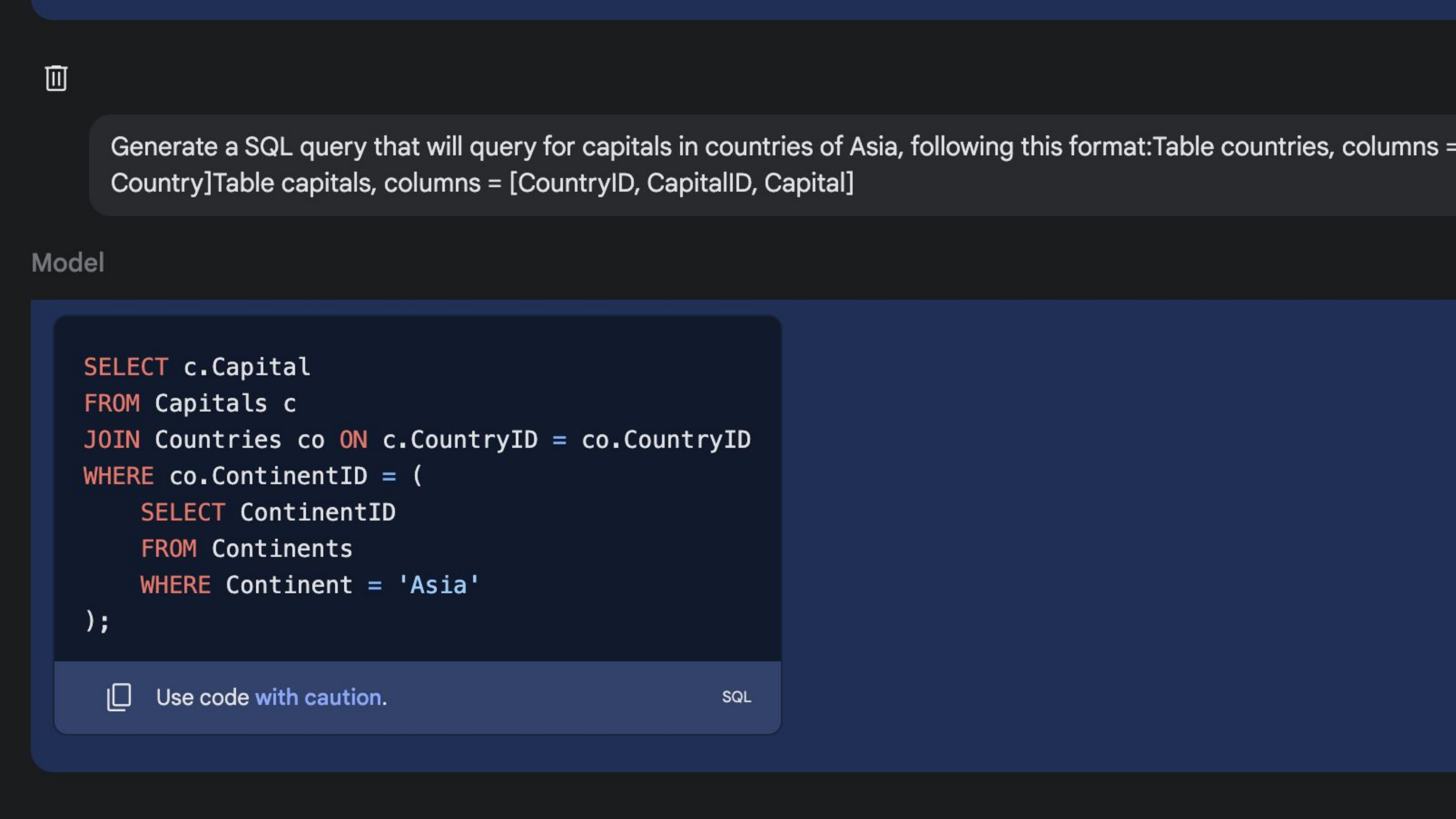
Generate a table that has all the countries in the world listed alphabetically, their capitals, and what continent they belong to.

Country	Capital	Continent
Afghanistan	Kabul	Asia
Albania	Tirana	Europe
Algeria	Algiers	Africa
Andorra	Andorra la Vella	Europe
Angola	Luanda	Africa
Antigua and Barbuda	Saint John's	North America
Argentina	Buenos Aires	South America
Armenia	Yerevan	Asia

Code generation

Examples

- Generate data
- Generate a SQL query



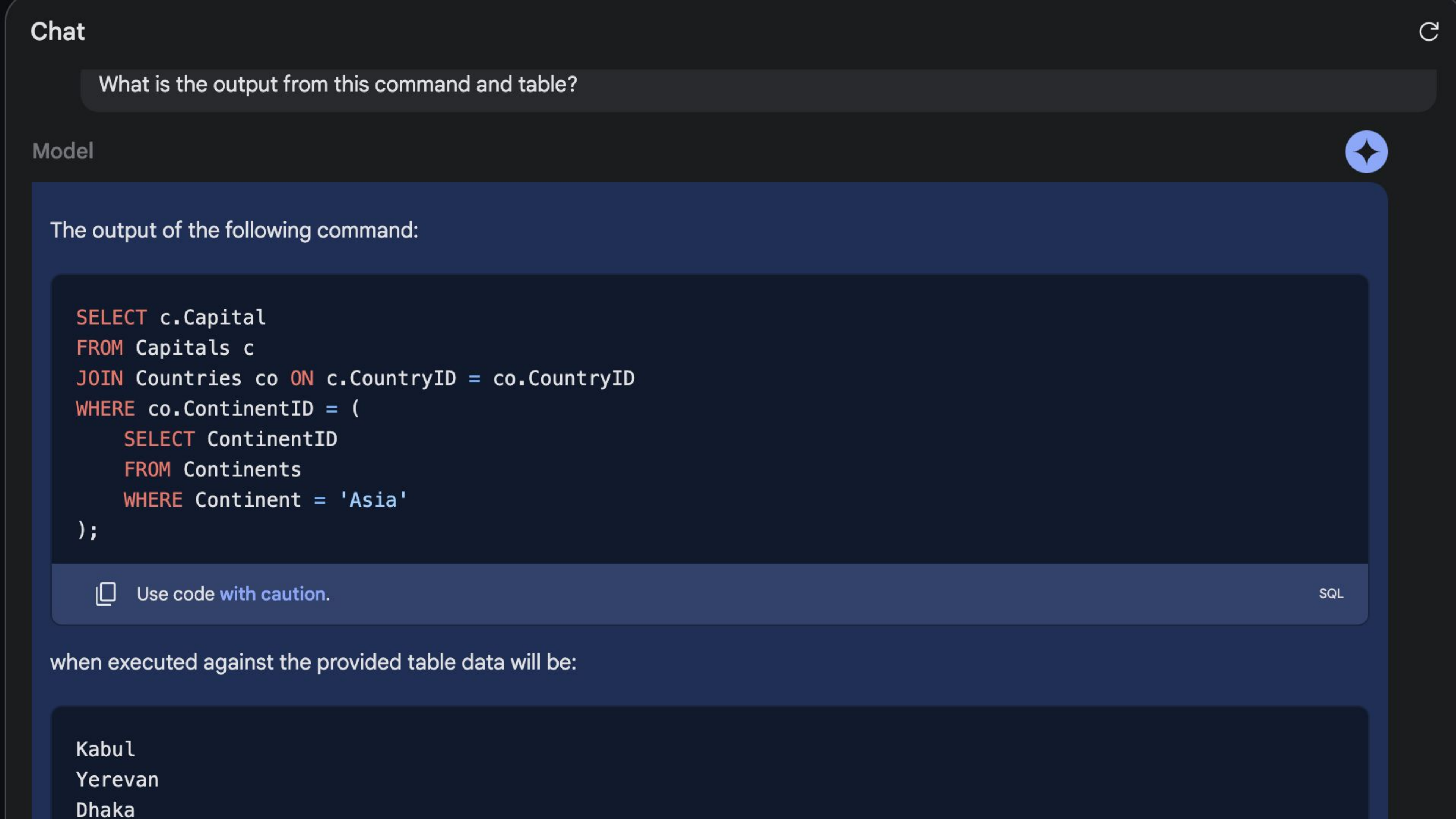
Generate a SQL query that will query for capitals in countries of Asia, following this format:Table countries, columns =
Country]Table capitals, columns = [CountryID, CapitalID, Capital]

```sql
SELECT c.Capital
FROM Capitals c
JOIN Countries co ON c.CountryID = co.CountryID
WHERE co.ContinentID = (
    SELECT ContinentID
    FROM Continents
    WHERE Continent = 'Asia'
);
```

Code generation

Examples

- Generate data
- Generate a SQL query
- Simulate Execution



What is the output from this command and table?

The output of the following command:

```sql
SELECT c.Capital
FROM Capitals c
JOIN Countries co ON c.CountryID = co.CountryID
WHERE co.ContinentID = (
    SELECT ContinentID
    FROM Continents
    WHERE Continent = 'Asia'
);
```

Use code with caution.

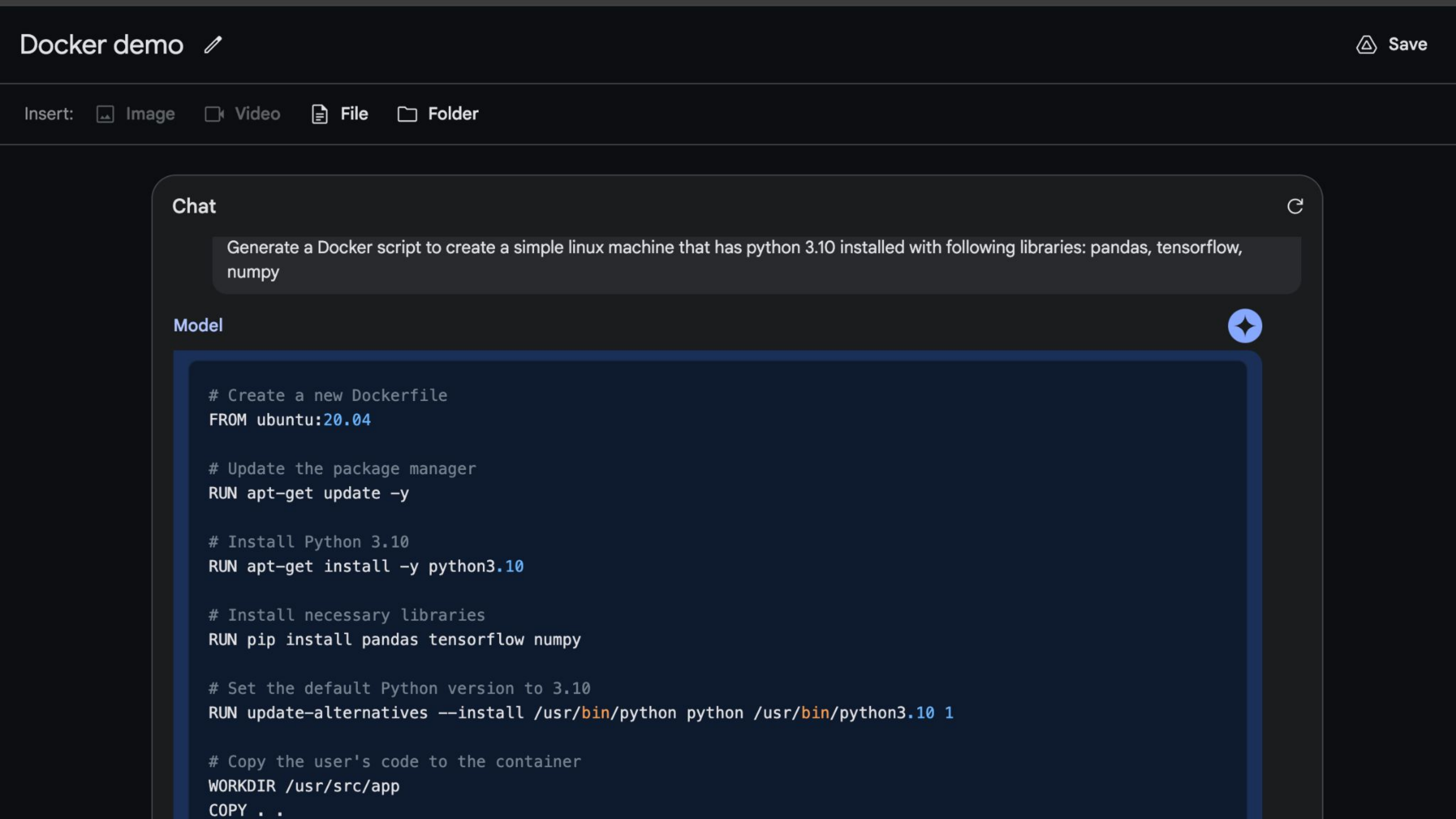
when executed against the provided table data will be:

```
Kabul
Yerevan
Dhaka
```

Code generation

Examples

- Generate data
- Generate a SQL query
- Simulate Execution
- Quick coding help - Smart pair programer.



Generate a Docker script to create a simple linux machine that has python 3.10 installed with following libraries: pandas, tensorflow, numpy

```
# Create a new Dockerfile
FROM ubuntu:20.04

# Update the package manager
RUN apt-get update -y

# Install Python 3.10
RUN apt-get install -y python3.10

# Install necessary libraries
RUN pip install pandas tensorflow numpy

# Set the default Python version to 3.10
RUN update-alternatives --install /usr/bin/python python /usr/bin/python3.10 1

# Copy the user's code to the container
WORKDIR /usr/src/app
COPY . .
```

Write a unit test for this Python function:

```python
def is_diver_neutrally_buoyant(density_water, density_diver):
  """Check if a diver is neutrally buoyant."""
  if density_water == density_diver:
    return True
```

```python
import unittest

class IsDiverNeutrallyBuoyantTest(unittest.TestCase):

    def test_diver_is_neutrally_buoyant(self):
        self.assertTrue(is_diver_neutrally_buoyant(1000, 1000))

    def test_diver_is_not_neutrally_buoyant(self):
        self.assertFalse(is_diver_neutrally_buoyant(1000, 1010))

    def test_invalid_input_types(self):
        with self.assertRaises(TypeError):
            is_diver_neutrally_buoyant("1000", 1000)
        with self.assertRaises(TypeError):
            is_diver_neutrally_buoyant(1000, "1000")
```

Usage Patterns

Search and Information Synthesis

Search & IR

BYO Data

- Models have knowledge cut-offs
- LLMs are not fact engines
- No exposure to private data

Search & IR

Use the prompt's context window

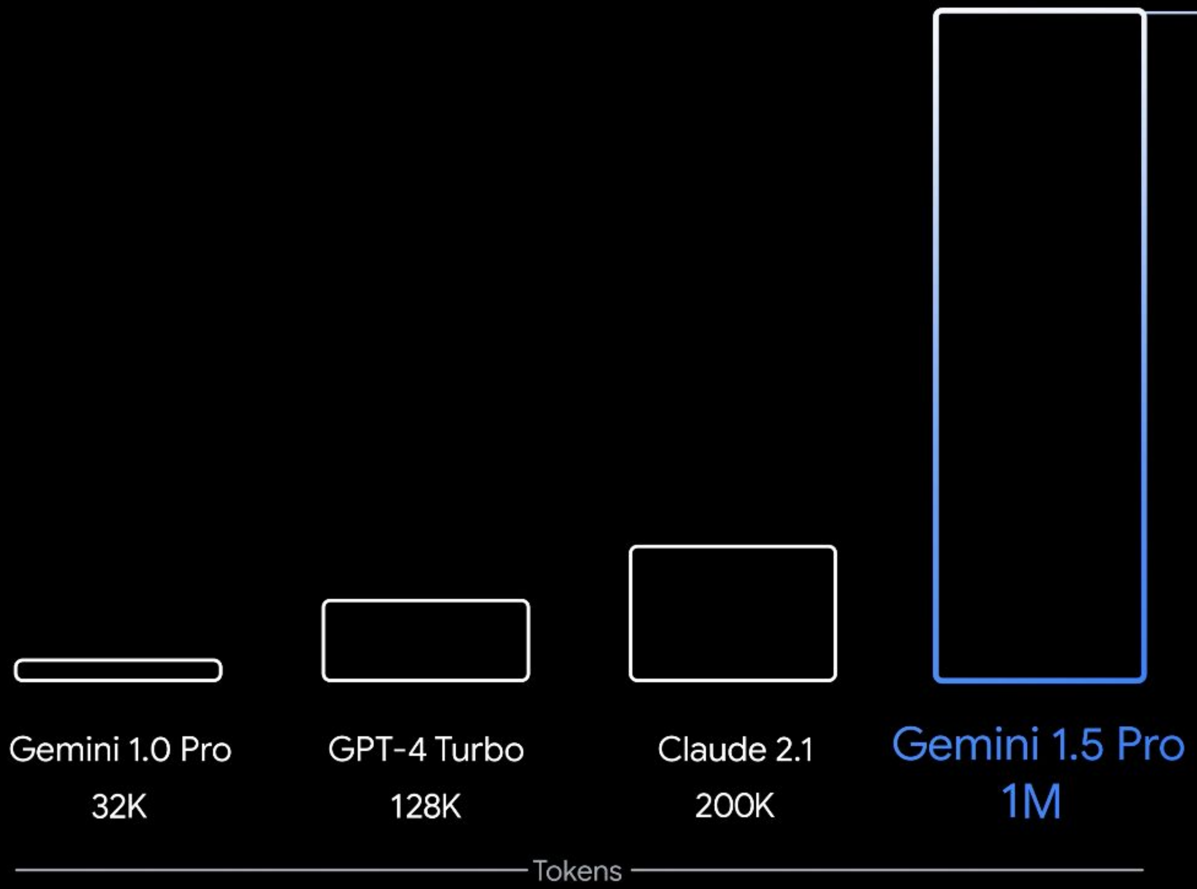
- Instructions + Context + Question all in the prompt
- Easy to implement
    - No extra code, just ask.

```
model = genai.GenerativeModel('gemini-pro')

document = pathlib.Path('document.txt').read_text()

result = model.generate_content(f"""
  Explain how deep-sea life survives.

  Please answer based on the following document:
  {document}""")
```

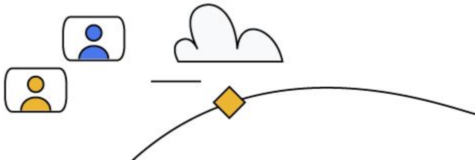


up to
10M
in research

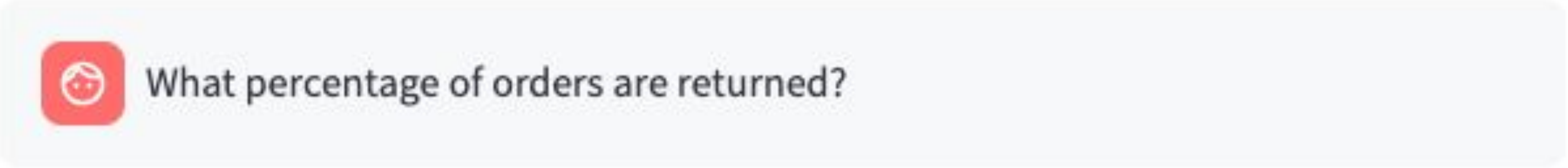
- 1 hour video
- 11 hours audio
- > 30K lines of code
- > 700K words

Usage Patterns

Function Calling



Function Calling

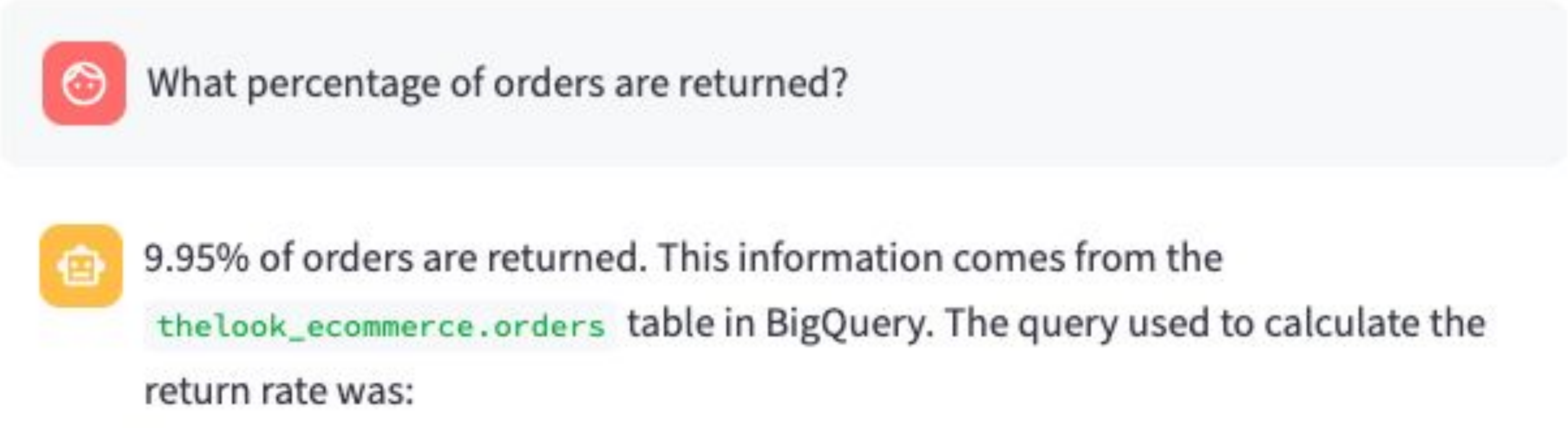


What percentage of orders are returned?

https://github.com/GoogleCloudPlatform/generative-ai/tree/main/gemini/function-calling/sql-talk-app

goo.gle/gemini-fn-call-sql-github

Function Calling



What percentage of orders are returned?

9.95% of orders are returned. This information comes from the `thelook_ecommerce.orders` table in BigQuery. The query used to calculate the return rate was:

```
SELECT ROUND((
COUNT(DISTINCT IF(returned_at IS NOT NULL, order_id, NULL)) /
COUNT(DISTINCT order_id)) * 100, 2)
AS return_rate
FROM thelook_ecommerce.orders
```

https://github.com/GoogleCloudPlatform/generative-ai/tree/main/gemini/function-calling/sql-talk-app

goo.gle/gemini-fn-call-sql-github

Function Calling

- Describe external functions to the model.
- The model may ask **you** to call the function to help it respond to your queries.
- Endless possibilities for integrating external tools.

Function calling - Basics

- How?
- The `google.generativeai` SDK will **inspect the function's type hints to determine the schema.**
- Allowed types are limited:

```
AllowedTypes = (
    int | float | str | bool | list | dict )
```

https://ai.google.dev/tutorials/function_calling_python_quickstart

```python
def multiply(a: float, b: float):
    """Returns a * b."""
    return a*b

model = genai.GenerativeModel(
    model_name='gemini-1.0-pro',
    tools=[multiply])
```

Function calling - Basics

- Because function calling requires alternating turns, it's easiest to use through chat.
- Enable "automatic function calling" when you start a chat, and the `ChatSession` will call the function(s) for you.
  - You don't have to use automatic function calling, it just makes simple cases easier.

https://ai.google.dev/tutorials/function_calling_python_quickstart

```
chat = model.start_chat(
    enable_automatic_function_calling=True)

response = chat.send_message(
    'I have 57 cats, each owns 44 mittens, '
    'how many mittens is that in total?')
```

```
chat = model.start_chat(
    enable_automatic_function_calling=True)

response = chat.send_message(
    'I have 57 cats, each owns 44 mittens, '
    'how many mittens is that in total?')

print(response.text)
# The number of mittens in total is 2508.
```

Function Calling

Function calling - Basics

- What happened? Use the chat history to find out.
- The chat history collects all the function calls and responses that took place.

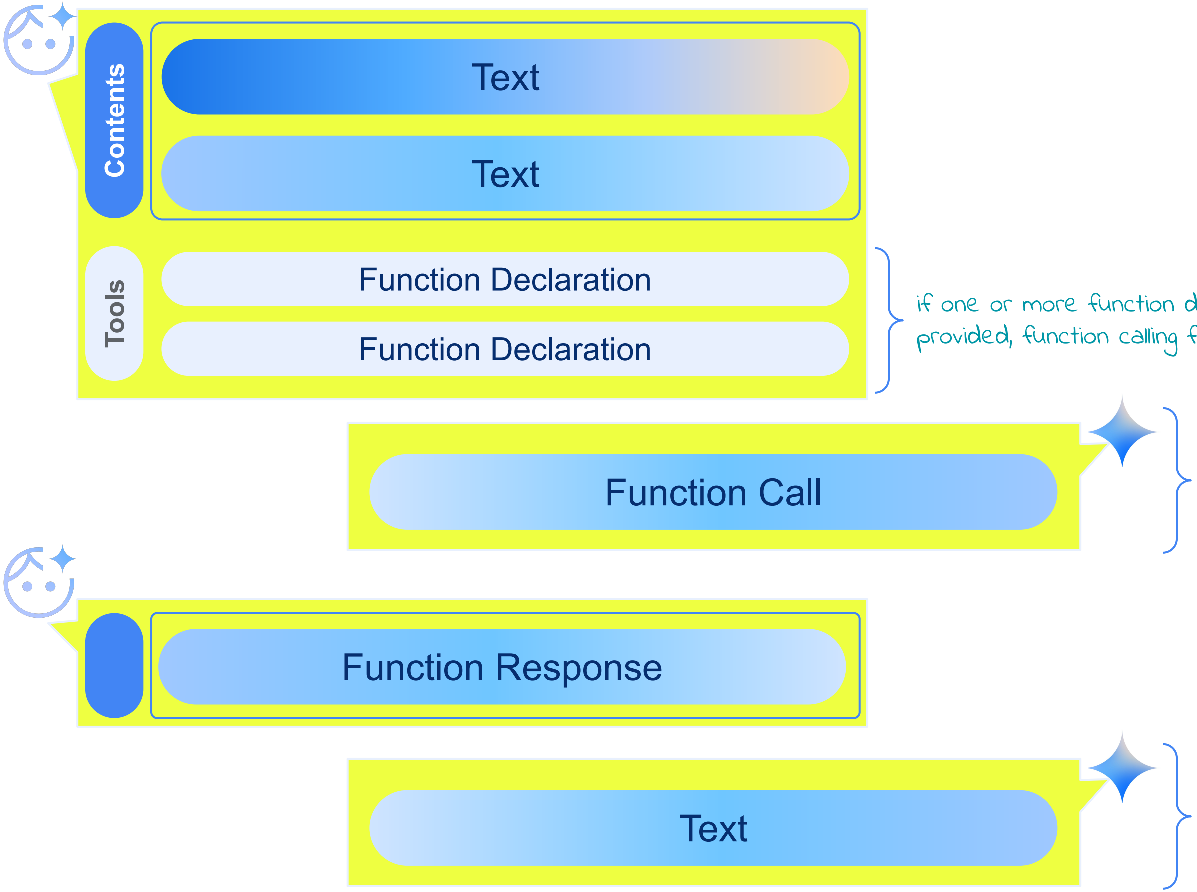
https://ai.google.dev/tutorials/function_calling_python_quickstart

```python
for content in chat.history:
    part = content.parts[0]
    print(content.role, "->", type(part).to_dict(part))
```

```
for content in chat.history:
    part = content.parts[0]
    print(content.role, "->", type(part).to_dict(part))

# user -> {'text': 'I have 57 cats, each owns 44 mittens, '
#                   'how many mittens is that in total?'}
# model -> {'function_call': {'name': 'multiply',
#                              'args': {'a': 57.0, 'b': 44.0}}}
# user -> {'function_response': {'name': 'multiply',
#                                 'response': {'result': 2508.0}}}
# model -> {'text': ' The number of mittens in total is 2508.'}
```

Function Calling interaction



Contents

Text

Text

Tools

Function Declaration

Function Declaration

if one or more function declarations are provided, function calling feature will turn on

Function Call

model may predict a function call based on user content

Function Response

Text

model can understand the function response and generate text
OR another function call

Function calling - More Examples

- Wikipedia research aid
    - Integrates a search tool.
    - Uses the Gemini API inside the function call to summarize pages.

```
def wikipedia_search(queries:list[str]) -> list[str]:
  ...
```

https://ai.google.dev/docs/search_reranking_using_embeddings/

```
model = genai.GenerativeModel('gemini-pro', tools=[wikipedia_search])

chat = model.start_chat(enable_automatic_function_calling=True)

query = "Explain how deep-sea life survives."

res = chat.send_message(instructions.format(query=query))
```

```
model = genai.GenerativeModel('gemini-pro', tools=[wikipedia_search])

chat = model.start_chat(enable_automatic_function_calling=True)

query = "Explain how deep-sea life survives."

res = chat.send_message(instructions.format(query=query))

 # Searching for "How do deep-sea creatures survive the extreme pressure?"
 # Related search terms: ['Deep sea', 'Deep-sea community', 'Deep-sea fish']
 # Fetching page: "Deep sea"
 # Information Source: https://en.wikipedia.org/wiki/Deep_sea
 # Fetching page: "Deep-sea community"
 # Information Source: https://en.wikipedia.org/wiki/Deep-sea_community
 # Fetching page: "Deep-sea fish"
 # Information Source: https://en.wikipedia.org/wiki/Deep-sea_fish
 # Searching for "How do deep-sea creatures survive the cold temperatures?"
 # Related search terms: ['Deep-sea community', 'Deep sea', 'Deep-water coral']
 # Fetching page: "Deep-water coral"
 # Information Source: https://en.wikipedia.org/wiki/Deep-water_coral
```

Function Calling

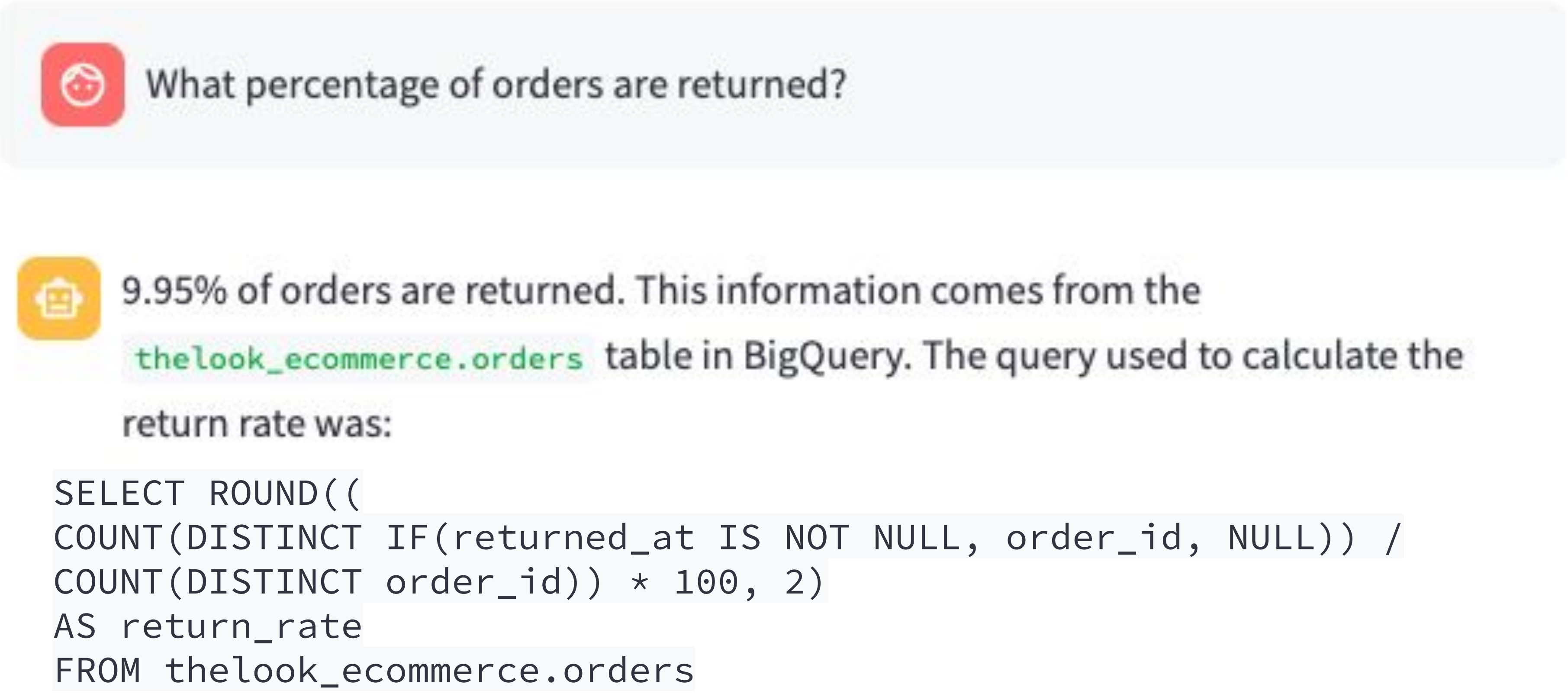
Function calling - More Examples

- SQL Talk
    - Use function calling to talk to a database.
    - Live example: https://goo.gle/gemini-fn-call-sql

https://github.com/GoogleCloudPlatform/generative-ai/tree/main/gemini/function-calling/sql-talk-app

```
sql_query_func = FunctionDeclaration(
    name="sql_query",
    description="Get information from data in BigQuery using SQL queries",
    parameters={
        "type": "object",
        "properties": {
            "query": {
                "type": "string",
                "description": "SQL query on a single line ...
```

Function Calling



What percentage of orders are returned?

9.95% of orders are returned. This information comes from the `thelook_ecommerce.orders` table in BigQuery. The query used to calculate the return rate was:

```
SELECT ROUND((
COUNT(DISTINCT IF(returned_at IS NOT NULL, order_id, NULL)) /
COUNT(DISTINCT order_id)) * 100, 2)
AS return_rate
FROM thelook_ecommerce.orders
```

https://github.com/GoogleCloudPlatform/generative-ai/tree/main/gemini/function-calling/sql-talk-app

Function calling - Schema

- Automatically building the schema from the type hints doesn't [currently 02/2024] work for everything.
- The allowed types are actually:

```
AllowedType= (
  int | float | str | bool |
  list['AllowedType'] |
  dict[str, 'AllowedType']
)
```

https://ai.google.dev/tutorials/function_calling_python_quickstart

Function Calling

Function calling - Schema

- Let's look at how the schema is constructed.

https://ai.google.dev/tutorials/function_calling_python_quickstart

```
model = genai.GenerativeModel(
    model_name='gemini-1.0-pro',
    tools=[multiply])

model._tools.to_proto()
```

```
[function_declarations {
   name: "multiply"
   description: "returns a * b."
   parameters {
       type_: OBJECT
       properties {
           key: "b" value { type_: NUMBER }}
       properties {
           key: "a" value { type_: NUMBER }}
       required: "a" required: "b" }}]
```

Function calling - Schema

- It's an OpenAPI schema, written as a `protobuf`.
- The protobuf-classes are available in the `google.ai.generativelanguage` client library.
- Reference docs:

https://ai.google.dev/api/python/google/ai/generativelanguage/FunctionDeclaration

https://ai.google.dev/tutorials/function_calling_python_quickstart

```
import google.ai.generativelanguage as glm

calculator = glm.Tool(
    function_declarations=[
      glm.FunctionDeclaration(
        name='multiply',
        description="Returns the product of two numbers.",
        parameters=glm.Schema(
            type=glm.Type.OBJECT,
            properties={
                'a': glm.Schema(type=glm.Type.NUMBER),
                'b': glm.Schema(type=glm.Type.NUMBER)},
            required=['a','b']))])
```

Function calling - Schema

- They can be written out as JSON-compatible objects as well.

https://ai.google.dev/tutorials/function_calling_python_quickstart

```
calculator = {
'function_declarations': [{
   'name': 'multiply',
   'description': 'Returns the product of two numbers.',
   'parameters': {
     'type': 'OBJECT',
     'properties': {
       'a': {'type': 'NUMBER'},
       'b': {'type': 'NUMBER'}},
     'required': ['a', 'b']}}]}
```

```
model = genai.GenerativeModel(
    model_name='gemini-1.0-pro',
    tools=[calculator])
```

Function Calling

Function calling - Structured data

- Structured data extraction.
- You *can* just ask the model to do it and return JSON.

https://ai.google.dev/tutorials/structured_data_extraction

```
response = model.generate_content(textwrap.dedent("""\
    Please return JSON describing the the people, places, things and relationships from this
    story using the following schema:

    {"people": list[PERSON], "places":list[PLACE], "things":list[THING], "relationships": list[RELATIONSHI

    PERSON = {"name": str, "description": str, "start_place_name": str, "end_place_name": str}
    PLACE = {"name": str, "description": str}
    THING = {"name": str, "description": str, "start_place_name": str, "end_place_name": str}
    RELATIONSHIP = {"person_1_name": str, "person_2_name": str, "relationship": str}

    Here is the story:

    """) + story)
```

Function Calling

Function calling - Structured data

- Asking for JSON often works.
- Function calling lets you strictly describe the schema.
- With a strict description, we can strictly enforce that that's what gets returned.

https://ai.google.dev/tutorials/structured_data_extraction

```python
add_to_database = glm.FunctionDeclaration(
    name="add_to_database",
    description="Adds entities to the database.",
    parameters=glm.Schema(
        type=glm.Type.OBJECT,
        properties = {
            'people': glm.Schema(
                type=glm.Type.ARRAY,
                items=glm.Schema(
                    type = glm.Type.OBJECT,
                    properties = {
                        'name':  glm.Schema(type=glm.Type.STRING),
                        'description':  glm.Schema(type=glm.Type.STRING),
                        'start_place_name': glm.Schema(type=glm.Type.STRING),
                        'end_place_name': glm.Schema(type=glm.Type.STRING)}))),
            'places': glm.Schema(
            ...
```

```
add_to_database = glm.FunctionDeclaration(
    name="add_to_database",
    description="Adds entities to the database.",
    parameters={
        "type": "OBJECT",
        "properties": {
            'people': {
                "type": "ARRAY",
                "items": {
                    "type": "OBJECT",
                    "properties": {
                        'name':  {"type":"STRING"},
                        'description':  {"type":"STRING"},
                        'start_place_name': {"type":"STRING"},
                        'end_place_name': {"type":"STRING"}}}),
            'places': {...},
            ...
```

```
model = model = genai.GenerativeModel(
    model_name='gemini-1.0-pro',
    tools = [add_to_database])
```

Usage Patterns

Image Understanding

Multimodality

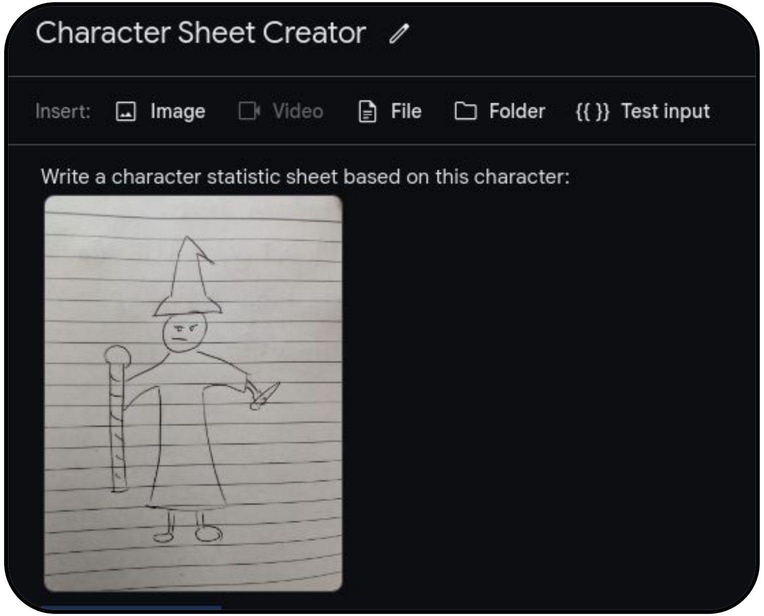
Image understanding

- Structured data extraction
- Image conditioning
- *RAG*

Multimodality

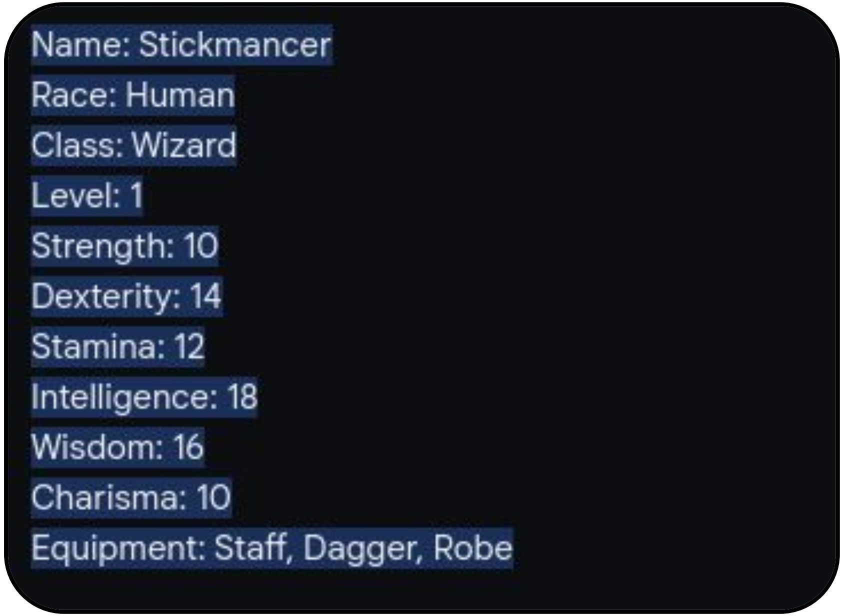
Image conditioning

- Generate text or structured data from images



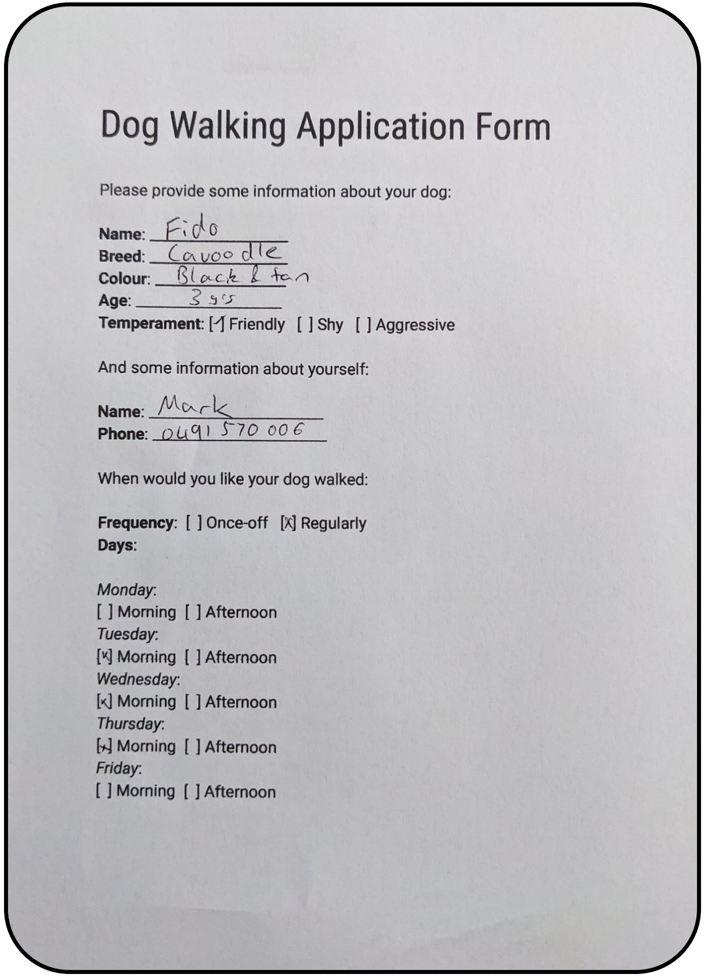
Character Sheet Creator

Write a character statistic sheet based on this character:



Name: Stickmancer
Race: Human
Class: Wizard
Level: 1
Strength: 10
Dexterity: 14
Stamina: 12
Intelligence: 18
Wisdom: 16
Charisma: 10
Equipment: Staff, Dagger, Robe

Structured Data



Dog Walking Application Form

Please provide some information about your dog:

Name: Fido
Breed: Cavoodle
Colour: Black & tan
Age: 3 yrs
Temperament: ☑ Friendly ☐ Shy ☐ Aggressive

And some information about yourself:

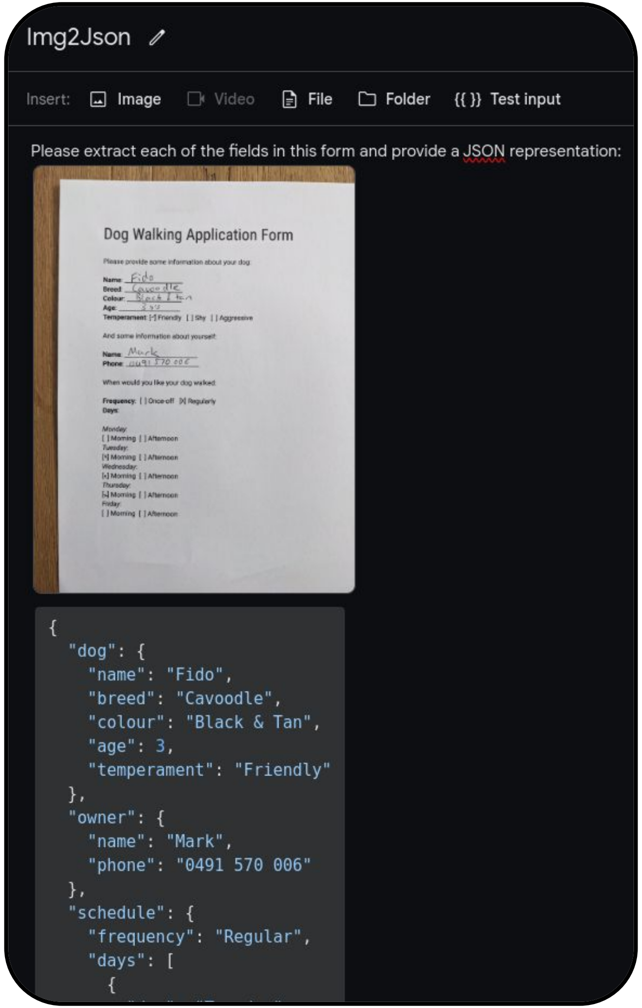
Name: Mark
Phone: 0491 570 006

When would you like your dog walked:

Frequency: ☐ Once-off ☒ Regularly
Days:

Monday:
☐ Morning ☐ Afternoon
Tuesday:
☒ Morning ☐ Afternoon
Wednesday:
☒ Morning ☐ Afternoon
Thursday:
☒ Morning ☐ Afternoon
Friday:
☐ Morning ☐ Afternoon

Structured Data



Please extract each of the fields in this form and provide a JSON representation:

Dog Walking Application Form

Please provide some information about your dog:

Name: Fido
Breed: Cavoodle
Colour: Black & tan
Age: 3 yrs
Temperament: ☒ Friendly ☐ Shy ☐ Aggressive

And some information about yourself:

Name: Mark
Phone: 0491 570 006

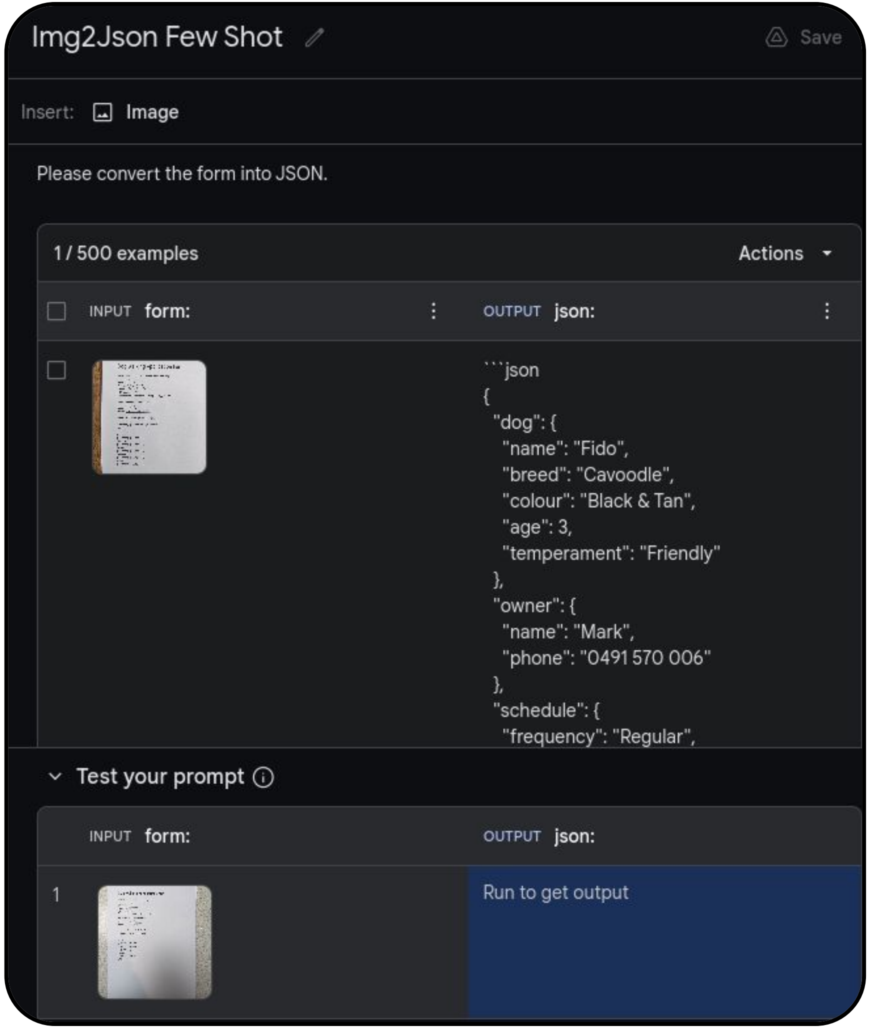
When would you like your dog walked:

Frequency: ☐ Once-off ☒ Regularly
Days:

Monday:
☐ Morning ☐ Afternoon
Tuesday:
☒ Morning ☐ Afternoon
Wednesday:
☒ Morning ☐ Afternoon
Thursday:
☒ Morning ☐ Afternoon
Friday:
☐ Morning ☐ Afternoon

```
{
  "dog": {
    "name": "Fido",
    "breed": "Cavoodle",
    "colour": "Black & Tan",
    "age": 3,
    "temperament": "Friendly"
  },
  "owner": {
    "name": "Mark",
    "phone": "0491 570 006"
  },
  "schedule": {
    "frequency": "Regular",
    "days": [
      {
```

Structured Data



Please convert the form into JSON.

~~~
```json
{
  "dog": {
    "name": "Fido",
    "breed": "Cavoodle",
    "colour": "Black & Tan",
    "age": 3,
    "temperament": "Friendly"
  },
  "owner": {
    "name": "Mark",
    "phone": "0491 570 006"
  },
  "schedule": {
    "frequency": "Regular",
~~~

```python
import enum
from pydantic import BaseModel

class DogApplication(BaseModel):
  """Represents an instance of a completed dog-walking application form."""
  dog: Dog
  owner: Customer
  schedule: Schedule

class Dog(BaseModel):
  name: str
  breed: str
  color: str
  age_years: float
  temperament: Temperament

class Temperament(str, enum.Enum):
  FRIENDLY = 'friendly'
  SHY = 'shy'
  AGGRESSIVE = 'aggressive'
```

```python
class Customer(BaseModel):
  name: str
  phone: str


class Schedule(BaseModel):
  regular: bool
  days_and_times: list[str]
```

```python
from llama_index.multi_modal_llms import GeminiMultiModal
from llama_index.multi_modal_llms.generic_utils import load_image_urls
from llama_index.program import MultiModalLLMCompletionProgram
from llama_index.output_parsers import PydanticOutputParser

images = load_image_urls(['https://goo.gle/dog-form-img'])
prompt = (
    "Please extract each of the fields in this "
    "form and provide a JSON representation: ")

# Load the Gemini Pro Vision model.
gemini_llm = GeminiMultiModal(api_key=GOOGLE_API_KEY, model_name=PRO_VISION)
```

```python
# Build a callable that will map the images to the Pydantic schema.
image_to_application = MultiModalLLMCompletionProgram.from_defaults(
    output_parser=PydanticOutputParser(DogApplication),
    image_documents=images,
    prompt_template_str=prompt,
    multi_modal_llm=gemini_llm,
)

# Image ORM!
form = image_to_application()

assert isinstance(form, DogApplication)
assert form.dog.name == 'Fido'
assert form.owner.name == 'Mark'
assert form.schedule.regular is True
assert 'Wednesday Morning' in form.schedule.days_and_times
```

```
pprint(form.dict())

{'dog': {'age_years': 3.0,
         'breed': 'Cavoodle',
         'color': 'Black & Tan',
         'name': 'Fido',
         'temperament': <Temperament.FRIENDLY: 'friendly'>},
 'owner': {'name': 'Mark', 'phone': '0491570006'},
 'schedule': {'days_and_times': ['Monday Morning',
                                 'Tuesday Morning',
                                 'Wednesday Morning'],
              'regular': True}}
```

Tuning

- Using the API: https://ai.google.dev/tutorials/tuning_quickstart_python

Thank You!

Suresh Peiris
Co-Founder at Inforwaves
GDG Organizer

Chamod Perera
SSE at Aleph Inc.
GDG Organizer & CNCF Ambassador