Version latest

Vertex Generative AI SDK for Python

The Vertex Generative AI SDK helps developers use Google's generative AI Gemini models (http://cloud.google.com/vertex-ai/docs/generative-ai/multimodal/overview) to build AI-powered features and applications. The SDKs support use cases like the following:

- Generate text from texts, images and videos (multimodal generation)
- Build stateful multi-turn conversations (chat)
- Function calling

Installation

To install the google-cloud-aiplatform (https://pypi.org/project/google-cloud-aiplatform/) Python package, run the following command:

```
pip3 install --upgrade --user "google-cloud-aiplatform>=1.38"
```

Usage

For detailed instructions, see quickstart (http://cloud.google.com/vertex-ai/docs/generative-ai/start/quickstarts/quickstart-multimodal) and Introduction to multimodal classes in the Vertex AI SDK (http://cloud.google.com/vertex-ai/docs/generative-ai/multimodal/sdk-for-gemini/gemini-sdk-overview-reference)
.

Imports:

```
import vertexai
```

Initialization:

```
vertexai.init(project='my-project', location='us-central1')
```

Basic generation:

```
from vertexai.generative_models import GenerativeModel
model = GenerativeModel("gemini-pro")
print(model.generate_content("Why is sky blue?"))
```

Using images and videos

```
from vertexai.generative_models import GenerativeModel, Image
vision_model = GenerativeModel("gemini-pro-vision")

# Local image
image = Image.load_from_file("image.jpg")
print(vision_model.generate_content(["What is shown in this image?", image]))

# Image from Cloud Storage
image_part = generative_models.Part.from_uri("gs://download.tensorflow.org/examp
print(vision_model.generate_content([image_part, "Describe this image?"]))

# Text and video
video_part = Part.from_uri("gs://cloud-samples-data/video/animals.mp4", mime_typ
print(vision_model.generate_content(["What is in the video? ", video_part]))
```

Chat

```
from vertexai.generative_models import GenerativeModel, Image
vision_model = GenerativeModel("gemini-ultra-vision")
vision_chat = vision_model.start_chat()
image = Image.load_from_file("image.jpg")
print(vision_chat.send_message(["I like this image.", image]))
print(vision_chat.send_message("What things do I like?."))
```

System instructions

```
from vertexai.generative_models import GenerativeModel
model = GenerativeModel(
    "gemini-1.0-pro",
    system_instruction=[
        "Talk like a pirate.",
        "Don't use rude words.",
    ],
)
print(model.generate_content("Why is sky blue?"))
```

Function calling

```
# First, create tools that the model is can use to answer your questions.
# Describe a function by specifying it's schema (JsonSchema format)
get_current_weather_func = generative_models.FunctionDeclaration(
    name="get_current_weather",
    description="Get the current weather in a given location",
    parameters={
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA"
            },
```

```
            "unit": {
                "type": "string",
                "enum": [
                    "celsius",
                    "fahrenheit",
                ]
            }
        },
        "required": [
            "location"
        ]
    },
)
# Tool is a collection of related functions
weather_tool = generative_models.Tool(
    function_declarations=[get_current_weather_func],
)

# Use tools in chat:
model = GenerativeModel(
    "gemini-pro",
    # You can specify tools when creating a model to avoid having to send them w
    tools=[weather_tool],
)
chat = model.start_chat()
# Send a message to the model. The model will respond with a function call.
print(chat.send_message("What is the weather like in Boston?"))
# Then send a function response to the model. The model will use it to answer.
print(chat.send_message(
    Part.from_function_response(
        name="get_current_weather",
        response={
            "content": {"weather": "super nice"},
        }
    ),
))
```

Automatic Function calling

Note: The `FunctionDeclaration.from_func` converter does not support nested types for parameters. Please provide full `FunctionDeclaration` instead.

```
from vertexai.preview.generative_models import GenerativeModel, Tool, FunctionDe

# First, create functions that the model can use to answer your questions.
def get_current_weather(location: str, unit: str = "centigrade"):
    """Gets weather in the specified location.

    Args:
        location: The location for which to get the weather.
        unit: Optional. Temperature unit. Can be Centigrade or Fahrenheit. Defau
    """
    return dict(
        location=location,
        unit=unit,
        weather="Super nice, but maybe a bit hot.",
    )

# Infer function schema
get_current_weather_func = FunctionDeclaration.from_func(get_current_weather)
# Tool is a collection of related functions
weather_tool = Tool(
    function_declarations=[get_current_weather_func],
)

# Use tools in chat:
model = GenerativeModel(
    "gemini-pro",
    # You can specify tools when creating a model to avoid having to send them w
    tools=[weather_tool],
)

# Activate automatic function calling:
afc_responder = AutomaticFunctionCallingResponder(
    # Optional:
    max_automatic_function_calls=5,
)
chat = model.start_chat(responder=afc_responder)
# Send a message to the model. The model will respond with a function call.
# The SDK will automatically call the requested function and respond to the mode
# The model will use the function call response to answer the original question.
print(chat.send_message("What is the weather like in Boston?"))
```

Evaluation

- To perform bring-your-own-response(BYOR) evaluation, provide the model responses in the **`response`** column in the dataset. If a pairwise metric is used for BYOR evaluation, provide the baseline model responses in the **`baseline_model_response`** column.

```
import pandas as pd
from vertexai.evaluation import EvalTask, MetricPromptTemplateExamples

eval_dataset = pd.DataFrame({
        "prompt"  : [...],
        "reference": [...],
        "response" : [...],
        "baseline_model_response": [...],
})
eval_task = EvalTask(
    dataset=eval_dataset,
    metrics=[
            "bleu",
            "rouge_l_sum",
            MetricPromptTemplateExamples.Pointwise.FLUENCY,
            MetricPromptTemplateExamples.Pairwise.SAFETY
    ],
    experiment="my-experiment",
)
eval_result = eval_task.evaluate(experiment_run_name="eval-experiment-run")
```

- To perform evaluation with Gemini model inference, specify the **`model`** parameter with a **`GenerativeModel`** instance. The input column name to the model is **`prompt`** and must be present in the dataset.

```
from vertexai.evaluation import EvalTask
from vertexai.generative_models import GenerativeModel

eval_dataset = pd.DataFrame({
    "reference": [...],
    "prompt"  : [...],
})
result = EvalTask(
```

```
    dataset=eval_dataset,
    metrics=["exact_match", "bleu", "rouge_1", "rouge_l_sum"],
    experiment="my-experiment",
).evaluate(
    model=GenerativeModel("gemini-1.5-pro"),
    experiment_run_name="gemini-eval-run"
)
```

- If a **`prompt_template`** is specified, the **`prompt`** column is not required. Prompts can be assembled from the evaluation dataset, and all prompt template variable names must be present in the dataset columns.

```
import pandas as pd
from vertexai.evaluation import EvalTask, MetricPromptTemplateExamples
from vertexai.generative_models import GenerativeModel

eval_dataset = pd.DataFrame({
    "context"    : [...],
    "instruction": [...],
})
result = EvalTask(
    dataset=eval_dataset,
    metrics=[MetricPromptTemplateExamples.Pointwise.SUMMARIZATION_QUALITY],
).evaluate(
    model=GenerativeModel("gemini-1.5-pro"),
    prompt_template="{instruction}. Article: {context}. Summary:",
)
```

- To perform evaluation with custom model inference, specify the **`model`** parameter with a custom inference function. The input column name to the custom inference function is **`prompt`** and must be present in the dataset.

```
from openai import OpenAI
from vertexai.evaluation import EvalTask, MetricPromptTemplateExamples


client = OpenAI()
def custom_model_fn(input: str) -> str:
```

```
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
        {"role": "user", "content": input}
        ]
    )
    return response.choices[0].message.content

eval_dataset = pd.DataFrame({
    "prompt"  : [...],
    "reference": [...],
})
result = EvalTask(
    dataset=eval_dataset,
    metrics=[MetricPromptTemplateExamples.Pointwise.SAFETY],
    experiment="my-experiment",
).evaluate(
    model=custom_model_fn,
    experiment_run_name="gpt-eval-run"
)
```

- To perform pairwise metric evaluation with model inference step, specify the **`baseline_model`** input to a **`PairwiseMetric`** instance and the candidate **`model`** input to the **`EvalTask.evaluate()`** function. The input column name to both models is **`prompt`** and must be present in the dataset.

```
import pandas as pd
from vertexai.evaluation import EvalTask, MetricPromptTemplateExamples, Pairwise
from vertexai.generative_models import GenerativeModel

baseline_model = GenerativeModel("gemini-1.0-pro")
candidate_model = GenerativeModel("gemini-1.5-pro")

pairwise_groundedness = PairwiseMetric(
    metric_prompt_template=MetricPromptTemplateExamples.get_prompt_template(
        "pairwise_groundedness"
    ),
    baseline_model=baseline_model,
)
eval_dataset = pd.DataFrame({
    "prompt"  : [...],
```

```
})
result = EvalTask(
    dataset=eval_dataset,
    metrics=[pairwise_groundedness],
    experiment="my-pairwise-experiment",
).evaluate(
    model=candidate_model,
    experiment_run_name="gemini-pairwise-eval-run",
)
```

Documentation

You can find complete documentation for the Vertex AI SDKs and the Gemini model in the Google Cloud documentation (https://cloud.google.com/vertex-ai/docs/generative-ai/learn/overview)

Contributing

See Contributing (https://github.com/googleapis/python-aiplatform/blob/main/CONTRIBUTING.rst) for more information on contributing to the Vertex AI Python SDK.

License

Last updated 2025-06-05 UTC.