Exécution de code

L'API Gemini fournit un outil d'exécution de code qui permet au modèle de générer et d'exécuter du code Python. Le modèle peut ensuite apprendre de manière itérative à partir des résultats de l'exécution du code jusqu'à ce qu'il parvienne à un résultat final. Vous pouvez utiliser l'exécution de code pour créer des applications qui bénéficient d'un raisonnement basé sur du code. Par exemple, vous pouvez utiliser l'exécution de code pour résoudre des équations ou traiter du texte. Vous pouvez également utiliser les bibliothèques (#supported-libraries) incluses dans l'environnement d'exécution du code pour effectuer des tâches plus spécialisées.

Gemini ne peut exécuter du code qu'en Python. Vous pouvez toujours demander à Gemini de générer du code dans une autre langue, mais le modèle ne peut pas utiliser l'outil d'exécution de code pour l'exécuter.

Activer l'exécution de code

Pour activer l'exécution de code, configurez l'outil d'exécution de code sur le modèle. Cela permet au modèle de générer et d'exécuter du code.

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest)

```
from google import genai
from google.genai import types
```

```
client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents="What is the sum of the first 50 prime numbers? "
    "Generate and run code for the calculation, and make sure you get all 50.",
    config=types.GenerateContentConfig(
        tools=[types.Tool(code_execution=types.ToolCodeExecution)]
    ),
)

for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    if part.executable_code is not None:
        print(part.executable_code.code)
    if part.code_execution_result is not None:
        print(part.code_execution_result.output)
```

Le résultat peut ressembler à ce qui suit (mise en forme pour une meilleure lisibilité) :

```
Okay, I need to calculate the sum of the first 50 prime numbers. Here's how I'll
approach this:

1.  **Generate Prime Numbers:** I'll use an iterative method to find prime
    numbers. I'll start with 2 and check if each subsequent number is divisible
    by any number between 2 and its square root. If not, it's a prime.
2.  **Store Primes:** I'll store the prime numbers in a list until I have 50 of
    them.
3.  **Calculate the Sum:**  Finally, I'll sum the prime numbers in the list.
```

```
Here's the Python code to do this:

def is_prime(n):
  """Efficiently checks if a number is prime."""
  if n <= 1:
    return False
  if n <= 3:
    return True
  if n % 2 == 0 or n % 3 == 0:
    return False
  i = 5
  while i * i <= n:
    if n % i == 0 or n % (i + 2) == 0:
      return False
    i += 6
  return True

primes = []
num = 2
while len(primes) < 50:
  if is_prime(num):
    primes.append(num)
  num += 1

sum_of_primes = sum(primes)
print(f'{primes=}')
print(f'{sum_of_primes=}')

primes=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229]
```

```
sum_of_primes=5117

The sum of the first 50 prime numbers is 5117.
```

Cette sortie combine plusieurs parties de contenu que le modèle renvoie lors de l'exécution de code :

- `text` : texte intégré généré par le modèle.
- `executableCode` : code généré par le modèle et destiné à être exécuté
- `codeExecutionResult` : résultat du code exécutable

Les conventions de nommage pour ces parties varient selon le langage de programmation.

Exécution de code avec des images (Gemini 3)

Le modèle Gemini 3 Flash peut désormais écrire et exécuter du code Python pour manipuler et inspecter activement des images. Cette fonctionnalité s'appelle *Visual Thinking*.

Cas d'utilisation

- **Zoom et inspection** : le modèle détecte implicitement lorsque les détails sont trop petits (par exemple, la lecture d'une jauge éloignée) et écrit du code pour recadrer et réexaminer la zone à une résolution plus élevée.
- **Mathématiques visuelles** : le modèle peut effectuer des calculs en plusieurs étapes à l'aide de code (par exemple, en additionnant les lignes d'un reçu).
- **Annotation d'images** : le modèle peut annoter des images pour répondre à des questions, par exemple en dessinant des flèches pour montrer des relations.

Remarque : Bien que le modèle gère automatiquement le zoom pour les petits détails, vous devez lui demander explicitement d'utiliser du code pour d'autres tâches, par exemple "Écris du code pour compter le nombre d'engrenages" ou "Fais pivoter cette image pour la mettre à l'endroit".

Activer la pensée visuelle

La pensée visuelle est officiellement prise en charge dans Gemini 3 Flash. Pour activer ce comportement, vous devez activer à la fois l'exécution de code en tant qu'outil et la réflexion.

Python JavaScript (#javascript) Go (#go) REST (#rest)
(#python)

```python
from google import genai
from google.genai import types
import requests
from PIL import Image
import io

image_path = "https://goo.gle/instrument-img"
image_bytes = requests.get(image_path).content
image = types.Part.from_bytes(
  data=image_bytes, mime_type="image/jpeg"
)

# Ensure you have your API key set
client = genai.Client(api_key="GEMINI_API_KEY")

response = client.models.generate_content(
    model="gemini-3-flash-preview",
```

```
    contents=[image, "Zoom into the expression pedals and tell me how many pedals are there?"],
    config=types.GenerateContentConfig(
        tools=[types.Tool(code_execution=types.ToolCodeExecution)]
    ),
)

for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    if part.executable_code is not None:
        print(part.executable_code.code)
    if part.code_execution_result is not None:
        print(part.code_execution_result.output)
    if part.as_image() is not None:
        # display() is a standard function in Jupyter/Colab notebooks
        display(Image.open(io.BytesIO(part.as_image().image_bytes)))
```

Utiliser l'exécution de code dans le chat

Vous pouvez également utiliser l'exécution de code dans un chat.

Python JavaScript (#javascript) Go (#go) REST (#rest)
(#python)

```
from google import genai
from google.genai import types

client = genai.Client()
```

```
chat = client.chats.create(
    model="gemini-2.5-flash",
    config=types.GenerateContentConfig(
        tools=[types.Tool(code_execution=types.ToolCodeExecution)]
    ),
)

response = chat.send_message("I have a math question for you.")
print(response.text)

response = chat.send_message(
    "What is the sum of the first 50 prime numbers? "
    "Generate and run code for the calculation, and make sure you get all 50."
)

for part in response.candidates[0].content.parts:
    if part.text is not None:
        print(part.text)
    if part.executable_code is not None:
        print(part.executable_code.code)
    if part.code_execution_result is not None:
        print(part.code_execution_result.output)
```

Entrée/Sortie (E/S)

À partir de Gemini 2.0 Flash (https://ai.google.dev/gemini-api/docs/models/gemini?hl=fr#gemini-2.0-flash), l'exécution de code est compatible avec les entrées de fichiers et les sorties de graphiques. Grâce à ces fonctionnalités d'entrée et de sortie, vous pouvez importer des fichiers CSV

et des fichiers texte, poser des questions sur les fichiers et générer des graphiques Matplotlib (https://matplotlib.org/) dans la réponse. Les fichiers de sortie sont renvoyés sous forme d'images intégrées dans la réponse.

Tarification des OI

Lorsque vous utilisez l'exécution de code I/O, les jetons d'entrée et de sortie vous sont facturés :

Jetons d'entrée :

- Prompt de l'utilisateur

Jetons de sortie :

- Code généré par le modèle
- Résultat de l'exécution du code dans l'environnement de code
- Jetons de réflexion
- Résumé généré par le modèle

Détails des E/S

Lorsque vous travaillez avec des E/S d'exécution de code, tenez compte des détails techniques suivants :

- La durée d'exécution maximale de l'environnement de code est de 30 secondes.
- Si l'environnement de code génère une erreur, le modèle peut décider de régénérer le code. Vous pouvez le faire jusqu'à cinq fois.
- La taille maximale des fichiers d'entrée est limitée par la fenêtre de jetons du modèle. Dans AI Studio, avec Gemini Flash 2.0, la taille maximale des fichiers d'entrée est de 1 million de jetons (environ 2 Mo pour les fichiers texte des types d'entrée compatibles). Si

vous importez un fichier trop volumineux, AI Studio ne vous permettra pas de l'envoyer.

- L'exécution de code fonctionne mieux avec les fichiers texte et CSV.
- Le fichier d'entrée peut être transmis au format `part.inlineData` ou `part.fileData` (importé via l'API Files (https://ai.google.dev/gemini-api/docs/files?hl=fr)), et le fichier de sortie est toujours renvoyé au format `part.inlineData`.

	Un seul tour	**Bidirectionnel (API Multimodal Live)**
Modèles compatibles	Tous les modèles Gemini 2.0 et 2.5	Modèles expérimentaux Flash uniquement
Types d'entrées de fichier acceptés	.png, .jpeg, .csv, .xml, .cpp, .java, .py, .js, .ts	.png, .jpeg, .csv, .xml, .cpp, .java, .py, .js, .ts
Bibliothèques de graphiques compatibles	Matplotlib, seaborn	Matplotlib, seaborn
Utilisation de plusieurs outils (https://ai.google.dev/gemini-api/docs/function-calling?hl=fr#multi-tool-use)	Oui (exécution de code et ancrage uniquement)	Oui

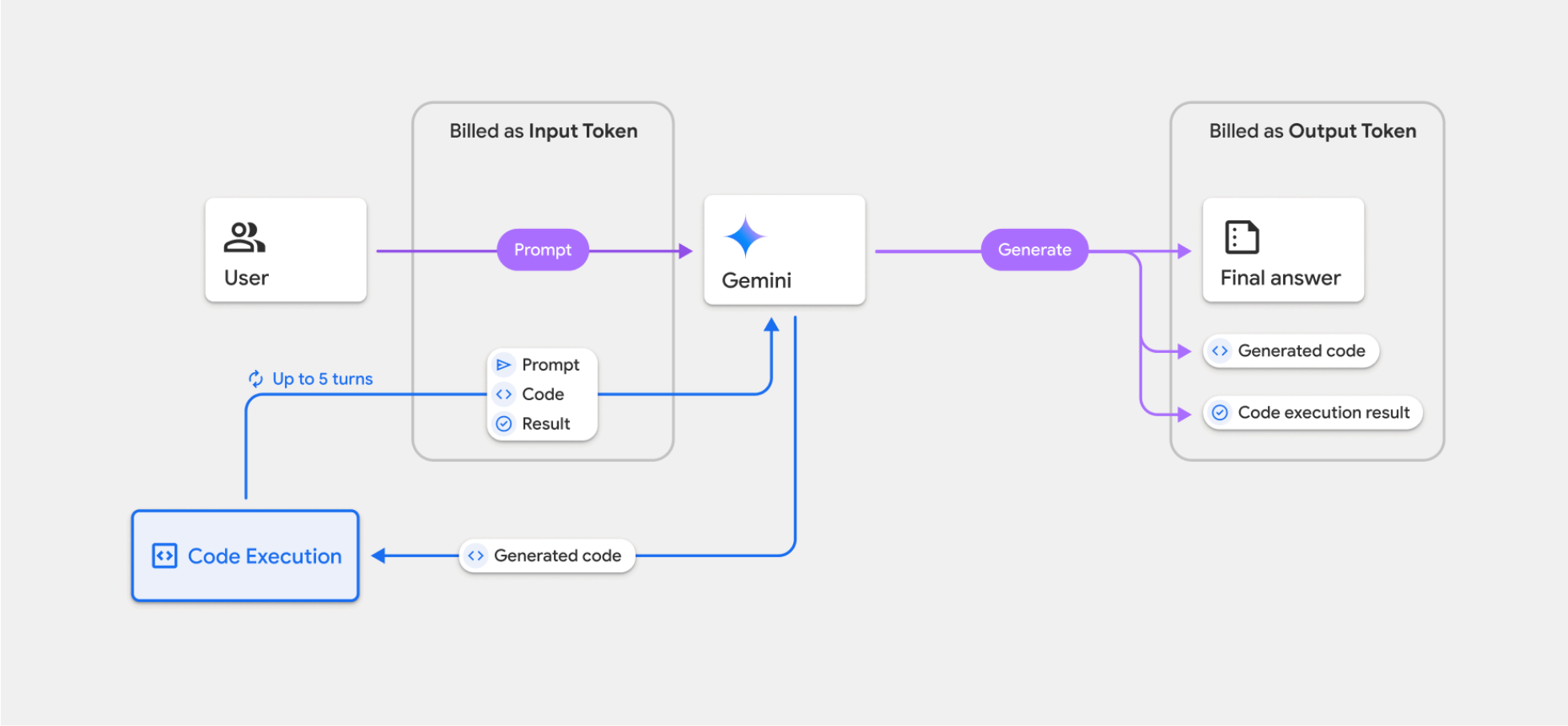
Facturation

L'exécution de code à partir de l'API Gemini n'entraîne aucuns frais supplémentaires. Vous serez facturé au tarif actuel des jetons d'entrée et de sortie en fonction du modèle Gemini que vous utilisez.

Voici quelques autres points à connaître concernant la facturation de l'exécution du code :

- Vous ne serez facturé qu'une seule fois pour les jetons d'entrée que vous transmettez au modèle, et vous serez facturé pour les jetons de sortie finaux qui vous sont renvoyés par le modèle.
- Les jetons représentant le code généré sont comptabilisés comme jetons de sortie. Le code généré peut inclure du texte et des sorties multimodales, comme des images.
- Les résultats de l'exécution du code sont également comptabilisés comme jetons de sortie.

Le modèle de facturation est illustré dans le schéma suivant :



- Vous êtes facturé au tarif actuel des jetons d'entrée et de sortie en fonction du modèle Gemini que vous utilisez.
- Si Gemini utilise l'exécution de code pour générer votre réponse, le prompt d'origine, le code généré et le résultat du code exécuté sont désignés comme des *jetons intermédiaires* et sont facturés en tant que *jetons d'entrée*.
- Gemini génère ensuite un résumé et renvoie le code généré, le résultat du code exécuté et le résumé final. Ils sont facturés en tant que *jetons de sortie*.

- L'API Gemini inclut un nombre de jetons intermédiaires dans la réponse de l'API. Vous savez ainsi pourquoi vous obtenez des jetons d'entrée supplémentaires au-delà de votre prompt initial.

Limites

- Le modèle ne peut que générer et exécuter du code. Il ne peut pas renvoyer d'autres artefacts tels que des fichiers multimédias.
- Dans certains cas, l'activation de l'exécution du code peut entraîner des régressions dans d'autres domaines de la sortie du modèle (par exemple, l'écriture d'une histoire).
- La capacité des différents modèles à exécuter du code avec succès varie.

Combinaisons d'outils compatibles

L'outil d'exécution de code peut être combiné à l'ancrage avec la recherche Google (https://ai.google.dev/gemini-api/docs/google-search?hl=fr) pour alimenter des cas d'utilisation plus complexes.

Bibliothèques prises en charge

L'environnement d'exécution du code inclut les bibliothèques suivantes :

- attrs
- échecs

- contourpy
- fpdf
- geopandas
- imageio
- jinja2
- joblib
- jsonschema
- jsonschema-specifications
- lxml
- matplotlib
- mpmath
- numpy
- opencv-python
- openpyxl
- packaging
- pandas
- pillow
- protobuf

- pylatex
- pyparsing
- PyPDF2
- python-dateutil
- python-docx
- python-pptx
- reportlab
- scikit-learn
- scipy
- seaborn
- six
- striprtf
- sympy
- tabuler
- tensorflow
- toolz
- xlrd

Vous ne pouvez pas installer vos propres bibliothèques.

Remarque : Seul `matplotlib` est compatible avec le rendu des graphiques à l'aide de l'exécution de code.

Étape suivante

- Essayez le Colab d'exécution de code (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Code_Execution.ipynb?hl=fr).
- Découvrez d'autres outils de l'API Gemini :
  - Appel de fonction (https://ai.google.dev/gemini-api/docs/function-calling?hl=fr)
  - Ancrage avec la recherche Google (https://ai.google.dev/gemini-api/docs/grounding?hl=fr)

Dernière mise à jour le 2025/12/18 (UTC).