# vila

PREVIEW

Multi-modal vision-language model that understands text/img/video and creates informative responses

[vlm](https://build.nvidia.com/search?q=VLM)[vision language model](https://build.nvidia.com/search?q=Vision+language+model)[image caption](https://build.nvidia.com/search?q=image+caption)[image to text](https://build.nvidia.com/search?q=image+to+text)

import requests

import os

import uuid

import sys

invoke\_url = "https://ai.api.nvidia.com/v1/vlm/nvidia/vila"

stream = False

query = "Describe the scene"

kApiKey = os.getenv("TEST\_NVCF\_API\_KEY", "")

assert kApiKey, "Generate API\_KEY and export TEST\_NVCF\_API\_KEY=xxxx"

kNvcfAssetUrl = "https://api.nvcf.nvidia.com/v2/nvcf/assets"

# ext: {mime, media}

kSupportedList = {

"png": ["image/png", "img"],

"jpg": ["image/jpg", "img"],

"jpeg": ["image/jpeg", "img"],

"mp4": ["video/mp4", "video"],

}

def get\_extention(filename):

\_, ext = os.path.splitext(filename)

ext = ext[1:].lower()

return ext

def mime\_type(ext):

return kSupportedList[ext][0]

def media\_type(ext):

return kSupportedList[ext][1]

def \_upload\_asset(media\_file, description):

ext = get\_extention(media\_file)

assert ext in kSupportedList

data\_input = open(media\_file, "rb")

headers={

"Authorization": f"Bearer {kApiKey}",

"Content-Type": "application/json",

"accept": "application/json",

}

assert\_url = kNvcfAssetUrl

authorize = requests.post(

assert\_url,

headers = headers,

json={"contentType": f"{mime\_type(ext)}", "description": description},

timeout=30,

)

authorize.raise\_for\_status()

authorize\_res = authorize.json()

print(f"uploadUrl: {authorize\_res['uploadUrl']}")

response = requests.put(

authorize\_res["uploadUrl"],

data=data\_input,

headers={

"x-amz-meta-nvcf-asset-description": description,

"content-type": mime\_type(ext),

},

timeout=300,

)

response.raise\_for\_status()

if response.status\_code == 200:

print(f"upload asset\_id {authorize\_res['assetId']} successfully!")

else:

print(f"upload asset\_id {authorize\_res['assetId']} failed.")

return uuid.UUID(authorize\_res["assetId"])

def \_delete\_asset(asset\_id):

headers = {

"Authorization": f"Bearer {kApiKey}",

}

assert\_url = f"{kNvcfAssetUrl}/{asset\_id}"

response = requests.delete(

assert\_url, headers=headers, timeout=30

)

response.raise\_for\_status()

def chat\_with\_media\_nvcf(infer\_url, media\_files, query: str, stream: bool = False):

asset\_list = []

ext\_list = []

media\_content = ""

assert isinstance(media\_files, list), f"{media\_files}"

print("uploading {media\_files} into s3")

has\_video = False

for media\_file in media\_files:

ext = get\_extention(media\_file)

assert ext in kSupportedList, f"{media\_file} format is not supported"

if media\_type(ext) == "video":

has\_video = True

asset\_id = \_upload\_asset(media\_file, "Reference media file")

asset\_list.append(f"{asset\_id}")

ext\_list.append(ext)

media\_content += f'<{media\_type(ext)} src="data:{mime\_type(ext)};asset\_id,{asset\_id}" />'

if has\_video:

assert len(media\_files) == 1, "Only single video supported."

asset\_seq = ",".join(asset\_list)

print(f"received asset\_id list: {asset\_seq}")

headers = {

"Authorization": f"Bearer {kApiKey}",

"Content-Type": "application/json",

"NVCF-INPUT-ASSET-REFERENCES": asset\_seq,

"NVCF-FUNCTION-ASSET-IDS": asset\_seq,

"Accept": "application/json",

}

if stream:

headers["Accept"] = "text/event-stream"

response = None

messages = [

{

"role": "user",

"content": f"{query} {media\_content}",

}

]

payload = {

"max\_tokens": 1024,

"temperature": 0.2,

"top\_p": 0.7,

"seed": 50,

"num\_frames\_per\_inference": 8,

"messages": messages,

"stream": stream,

"model": "nvidia/vila",

}

response = requests.post(infer\_url, headers=headers, json=payload, stream=stream)

if stream:

for line in response.iter\_lines():

if line:

print(line.decode("utf-8"))

else:

print(response.json())

print(f"deleting assets: {asset\_list}")

for asset\_id in asset\_list:

\_delete\_asset(asset\_id)

if \_\_name\_\_ == "\_\_main\_\_":

""" export TEST\_NVCF\_API\_KEY=xxx.

python test.py sample.mp4

python test.py sample1.png sample2.png

"""

if len(sys.argv) <= 1:

print("Usage: export TEST\_NVCF\_API\_KEY=xxx")

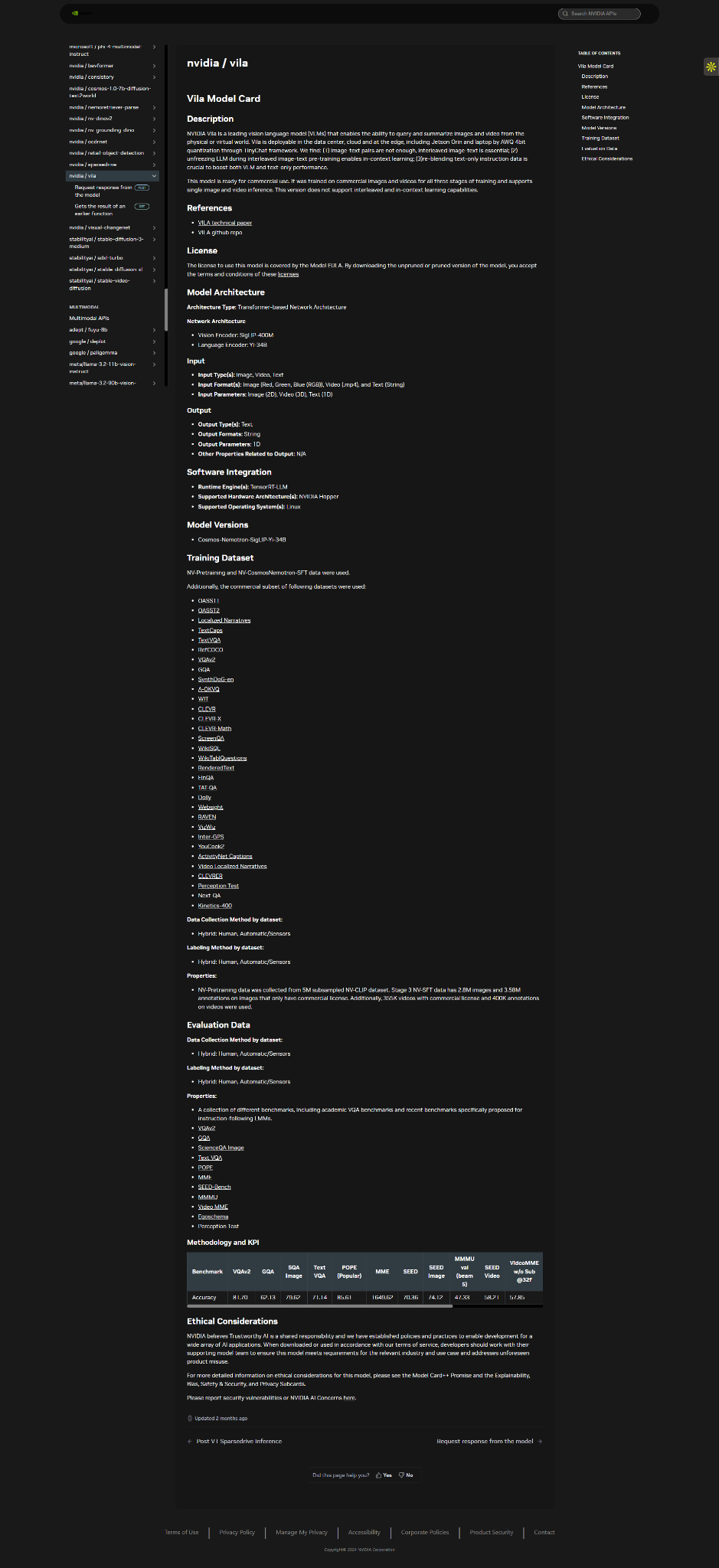
print(f" python {sys.argv[0]} sample1.png sample2.png ... sample16.png")

print(f" python {sys.argv[0]} sample.mp4")

sys.exit(1)

media\_samples = list(sys.argv[1:])

chat\_with\_media\_nvcf(invoke\_url, media\_samples, query, stream)

  
  
Request response from the model

posthttps://ai.api.nvidia.com/v1/vlm/nvidia/vila

Invokes inference using the model chat parameters. If uploading large images, this POST should be used in conjunction with the NVCF API which allows for the upload of large assets.  
You can find details on how to use NVCF Asset APIs here: <https://docs.api.nvidia.com/cloud-functions/reference/createasset>

Log in to see full request history

| time | status | user agent |  |
| --- | --- | --- | --- |
| Make a request to see history. | | | |

0 Requests This Month

Body Params

messages

array of objects

required

length between 1 and 1024

A list of messages comprising the conversation so far. The roles of the messages must be alternating between user and assistant. The last input message should have role user or assistant. A message with the system role is optional, and must be the very first message if it is present.

object

role

string

required

The role of the message author.

userassistant

content

required

Defaults to null

The contents of the message.  
  
Can only be null as part of a last request message with role=assistant (for "completion mode", i.e. providing the beginning of the assistant response).  
  
To pass images (only with role=user):  
  
- When content is a string, images can be passed together with the text with img HTML tags with base64 data: <img src="data:image/{format};base64,{base64encodedimage}" /> .  
If the size of an image is more than 180KB, it needs to be uploaded to a presigned S3 bucket using NVCF Asset APIs. Once uploaded you can refer to it using the following format: <img src="data:image/png;asset\_id,{asset\_id}" /> .  
  
- When content is a list of objects, images can be passed with objects with type=image\_url, and image\_url containing the base64 image data: data:image/{format};base64,{base64encodedimage}. HTML img tags will not be parsed from objects with type=text.  
  
- In both cases, images can be PNG, JPG or JPEG.  
  
For system and assistant roles, the object list format is not supported.

string

array

null

ADD object

model

string

required

Defaults to nvidia/vila

The model to use.

temperature

number

0 to 1

Defaults to 0.2

The sampling temperature to use for text generation. The higher the temperature value is, the less deterministic the output text will be. It is not recommended to modify both temperature and top\_p in the same call.

top\_p

number

≤ 1

Defaults to 0.7

An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top\_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. NVIDIA recommends that you alter this option or temperature but not both.

max\_tokens

integer

1 to 2048

Defaults to 1024

The maximum number of tokens to generate in any given call. Note that the model is not aware of this value, and generation will simply stop at the number of tokens specified.

seed

Defaults to 50

If specified, our system will make a best effort to sample deterministically, such that repeated requests with the same seed and parameters should return the same result.

integer

null

stream

Defaults to false

If set, partial message deltas will be sent. Tokens will be sent as data-only server-sent events (SSE) as they become available (JSON responses are prefixed by data: ), with the stream terminated by a data: [DONE] message.

boolean

null

num\_frames\_per\_inference

Defaults to 8

Number of frames to sample from the video or stream. They will be the input to model.

integer

null

Headers

NVCF-INPUT-ASSET-REFERENCES

uuid

length ≤ 370

String of asset IDs separated by commas. Data is uploaded to AWS S3 using NVCF Asset APIs and associated with these asset IDs.If the size of an image is more than 180KB, it needs to be uploaded to a presigned S3 URL bucket. The presigned URL allows for secure and temporary access to the S3 bucket for uploading the image. Once the asset is requested, an asset ID is generated for it. Please include this asset ID in this header and to use the uploaded image in a prompt, you need to refer to it using the following format: <img src="data:image/png;asset\_id,{asset\_id}" />.  
  
import requests

url = "https://ai.api.nvidia.com/v1/vlm/nvidia/vila"

payload = {

"messages": [

{

"role": "user",

"content": None

}

],

"model": "nvidia/vila",

"temperature": 0.2,

"top\_p": 0.7,

"max\_tokens": 1024,

"seed": 50,

"stream": False,

"num\_frames\_per\_inference": 8

}

headers = {

"accept": "application/json",

"content-type": "application/json"

}

response = requests.post(url, json=payload, headers=headers)

print(response.text)

—--------------------------------  
  
Gets the result of an earlier function invocation request that returned a status of 202.

gethttps://ai.api.nvidia.com/v1/status/{requestId}

import requests

url = "https://ai.api.nvidia.com/v1/status/requestId"

headers = {"accept": "application/json"}

response = requests.get(url, headers=headers)

print(response.text)