Use GPT-4 Turbo with Vision

01/26/2024

GPT-4 Turbo with Vision is a large multimodal model (LMM) developed by OpenAI that can analyze images and provide textual responses to questions about them. It incorporates both natural language processing and visual understanding.

The GPT-4 Turbo with Vision model answers general questions about what's present in the images. You can also show it video if you use Vision enhancement.

Tip

To use GPT-4 Turbo with Vision, you call the Chat Completion API on a GPT-4 Turbo with Vision model that you have deployed. If you're not familiar with the Chat Completion API, see the GPT-4 Turbo & GPT-4 how-to guide.

Call the Chat Completion APIs

The following command shows the most basic way to use the GPT-4 Turbo with Vision model with code. If this is your first time using these models programmatically, we recommend starting with our GPT-4 Turbo with Vision quickstart.

REST
Python

Send a POST request to `https://{RESOURCE_NAME}.openai.azure.com/openai/deployments/{DEPLOYMENT_NAME}/chat/completions?api-version=2023-12-01-preview` where

- RESOURCE_NAME is the name of your Azure OpenAI resource
- DEPLOYMENT_NAME is the name of your GPT-4 Turbo with Vision model deployment

Required headers:

- `Content-Type`: application/json
- `api-key`: {API_KEY}

Body: The following is a sample request body. The format is the same as the chat completions API for GPT-4, except that the message content can be an array containing text and images (either a valid HTTP or HTTPS URL to an image, or a base-64-encoded image).

Important

Remember to set a `"max_tokens"` value, or the return output will be cut off.

```
{
    "messages": [
        {
            "role": "system",
            "content": "You are a helpful assistant."
        },
        {
            "role": "user",
            "content": [
                    {
                        "type": "text",
                        "text": "Describe this picture:"
                    },
                    {
                        "type": "image_url",
                        "image_url": {
                        "url": "<image URL>"
                    }
                }
            ]
        }
    ],
    "max_tokens": 100,
    "stream": false
}
```

Tip

Use a local image

If you want to use a local image, you can use the following Python code to convert it to base64 so it can be passed to the API. Alternative file conversion tools are available online.

```
import base64
from mimetypes import guess_type

# Function to encode a local image into data URL
def local_image_to_data_url(image_path):
    # Guess the MIME type of the image based on the file extension
    mime_type, _ = guess_type(image_path)
    if mime_type is None:
        mime_type = 'application/octet-stream'  # Default MIME type if none is found

    # Read and encode the image file
    with open(image_path, "rb") as image_file:
        base64_encoded_data = base64.b64encode(image_file.read()).decode('utf-8')

    # Construct the data URL
    return f"data:{mime_type};base64,{base64_encoded_data}"

# Example usage
image_path = '<path_to_image>'
data_url = local_image_to_data_url(image_path)
print("Data URL:", data_url)
```

When your base64 image data is ready, you can pass it to the API in the request body like this:

```
...
"type": "image_url",
"image_url": {
   "url": "data:image/jpeg;base64,<your_image_data>"
}
...
```

The API response should look like the following.

```
{
    "id": "chatcmpl-8VAVx58veW9RCm5K1ttmxU6Cm4XDX",
    "object": "chat.completion",
    "created": 1702439277,
    "model": "gpt-4",
    "prompt_filter_results": [
        {
            "prompt_index": 0,
            "content_filter_results": {
                "hate": {
                    "filtered": false,
                    "severity": "safe"
                },
                "self_harm": {
                    "filtered": false,
                    "severity": "safe"
                },
                "sexual": {
                    "filtered": false,
                    "severity": "safe"
                },
                "violence": {
                    "filtered": false,
                    "severity": "safe"
                }
            }
        }
    ],
    "choices": [
        {
            "finish_reason":"stop",
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "The picture shows an individual dressed in formal attire, which includes a black tuxedo with a black bow tie.
            },
            "content_filter_results": {
                "hate": {
                    "filtered": false,
                    "severity": "safe"
                },
                "self_harm": {
                    "filtered": false,
                    "severity": "safe"
                },
                "sexual": {
                    "filtered": false,
                    "severity": "safe"
                },
                "violence": {
                    "filtered": false,
                    "severity": "safe"
                }
            }
        }
    ],
    "usage": {
        "prompt_tokens": 1156,
        "completion_tokens": 80,
        "total_tokens": 1236
    }
}
```

Every response includes a `"finish_details"` field. It has the following possible values:

- `stop`: API returned complete model output.
- `length`: Incomplete model output due to the `max_tokens` input parameter or model's token limit.
- `content_filter`: Omitted content due to a flag from our content filters.

Detail parameter settings in image processing: Low, High, Auto

The *detail* parameter in the model offers three choices: `low`, `high`, or `auto`, to adjust the way the model interprets and processes images. The default setting is auto, where the model decides between low or high based on the size of the image input.

- `low` setting: the model does not activate the "high res" mode, instead processes a lower resolution 512x512 version, resulting in quicker responses and reduced token consumption for scenarios where fine detail isn't crucial.
- `high` setting: the model activates "high res" mode. Here, the model initially views the low-resolution image and then generates detailed 512x512 segments from the input image. Each segment uses double the token budget, allowing for a more detailed interpretation of the image.''

For details on how the image parameters impact tokens used and pricing please see - What is OpenAI? Image Tokens with GPT-4 Turbo with Vision

GPT-4 Turbo with Vision provides exclusive access to Azure AI Services tailored enhancements. When combined with Azure AI Vision, it enhances your chat experience by providing the chat model with more detailed information about visible text in the image and the locations of objects.

The **Optical character recognition (OCR)** integration allows the model to produce higher quality responses for dense text, transformed images, and number-heavy financial documents. It also covers a wider range of languages.

The **object grounding** integration brings a new layer to data analysis and user interaction, as the feature can visually distinguish and highlight important elements in the images it processes.

Important

To use Vision enhancement, you need a Computer Vision resource. It must be in the paid (S0) tier and in the same Azure region as your GPT-4 Turbo with Vision resource.

Caution

Azure AI enhancements for GPT-4 Turbo with Vision will be billed separately from the core functionalities. Each specific Azure AI enhancement for GPT-4 Turbo with Vision has its own distinct charges. For details, see the special pricing information.

REST
Python

Send a POST request to
`https://{RESOURCE_NAME}.openai.azure.com/openai/deployments/{DEPLOYMENT_NAME}/extensions/chat/completions?api-`

`version=2023-12-01-preview` where

- RESOURCE_NAME is the name of your Azure OpenAI resource
- DEPLOYMENT_NAME is the name of your GPT-4 Turbo with Vision model deployment

Required headers:

- `Content-Type`: application/json
- `api-key`: {API_KEY}

Body:

The format is similar to that of the chat completions API for GPT-4, but the message content can be an array containing strings and images (either a valid HTTP or HTTPS URL to an image, or a base-64-encoded image).

You must also include the `enhancements` and `dataSources` objects. `enhancements` represents the specific Vision enhancement features requested in the chat. It has a `grounding` and `ocr` property, which both have a boolean `enabled` property. Use these to request the OCR service and/or the object detection/grounding service. `dataSources` represents the Computer Vision resource data that's needed for Vision enhancement. It has a `type` property which should be `"AzureComputerVision"` and a `parameters` property. Set the `endpoint` and `key` to the endpoint URL and access key of your Computer Vision resource.

Important

Remember to set a `"max_tokens"` value, or the return output will be cut off.

```
{
    "enhancements": {
            "ocr": {
              "enabled": true
            },
            "grounding": {
              "enabled": true
            }
    },
    "dataSources": [
    {
        "type": "AzureComputerVision",
        "parameters": {
            "endpoint": "<your_computer_vision_endpoint>",
            "key": "<your_computer_vision_key>"
        }
    }],
    "messages": [
        {
            "role": "system",
            "content": "You are a helpful assistant."
        },
        {
            "role": "user",
            "content": [
                    {
                        "type": "text",
                        "text": "Describe this picture:"
                    },
                    {
                        "type": "image_url",
                        "image_url": {
                        "url":"<image URL>"
                    }
                }
           ]
        }
    ],
    "max_tokens": 100,
    "stream": false
}
```

The chat responses you receive from the model should now include enhanced information about the image, such as object labels and bounding boxes, and OCR results. The API response should look like the following.

```
{
    "id": "chatcmpl-8UyuhLfzwTj34zpevT3tWlVIgCpPg",
    "object": "chat.completion",
    "created": 1702394683,
    "model": "gpt-4",
    "choices":
    [
        {
            "finish_details": {
                "type": "stop",
                "stop": "<|fim_suffix|>"
            },
            "index": 0,
            "message":
            {
                "role": "assistant",
                "content": "The image shows a close-up of an individual with dark hair and what appears to be a short haircut. The person
            },
            "enhancements":
            {
                "grounding":
                {
                    "lines":
                    [
                        {
                            "text": "The image shows a close-up of an individual with dark hair and what appears to be a short haircut. T
                            "spans":
                            [
                                {
                                    "text": "the person",
                                    "length": 10,
                                    "offset": 99,
                                    "polygon": [{"x":0.11950000375509262,"y":0.4124999940395355},{"x":0.8034999370574951,"y":0.4124999940
                                }
                            ]
                        }
                    ],
                    "status": "Success"
                }
            }
        }
    ],
    "usage":
    {
        "prompt_tokens": 816,
        "completion_tokens": 49,
        "total_tokens": 865
    }
}
```

Every response includes a "`finish_details`" field. It has the following possible values:

- `stop`: API returned complete model output.
- `length`: Incomplete model output due to the `max_tokens` input parameter or model's token limit.
- `content_filter`: Omitted content due to a flag from our content filters.

GPT-4 Turbo with Vision provides exclusive access to Azure AI Services tailored enhancements. The **video prompt** integration uses Azure AI Vision video retrieval to sample a set of frames from a video and create a transcript of the speech in the video. It enables the AI model to give summaries and answers about video content.

Important

To use Vision enhancement, you need a Computer Vision resource. It must be in the paid (S0) tier and in the same Azure region as your GPT-4 Turbo with Vision resource.

Caution

Azure AI enhancements for GPT-4 Turbo with Vision will be billed separately from the core functionalities. Each specific Azure AI enhancement for GPT-4 Turbo with Vision has its own distinct charges. For details, see the special pricing information.

Follow these steps to set up a video retrieval system to integrate with your AI chat model:

1. Get an Azure AI Vision resource in the same region as the Azure OpenAI resource you're using.
2. Follow the instructions in Do video retrieval using vectorization to create a video retrieval index. Return to this guide once your index is created.

3. Save the index name, the `documentId` values of your videos, and the blob storage SAS URLs of your videos to a temporary location. You'll need these values the next steps.

Call the Chat Completion API

REST
Python

1. Prepare a POST request to `https://{RESOURCE_NAME}.openai.azure.com/openai/deployments/{DEPLOYMENT_NAME}/extensions/chat/completions?api-version=2023-12-01-preview` where

    - RESOURCE_NAME is the name of your Azure OpenAI resource
    - DEPLOYMENT_NAME is the name of your GPT-4 Vision model deployment

    Required headers:

    - `Content-Type`: application/json
    - `api-key`: {API_KEY}

2. Add the following JSON structure in the request body:

```
{
    "enhancements": {
            "video": {
              "enabled": true
            }
    },
    "dataSources": [
    {
        "type": "AzureComputerVisionVideoIndex",
        "parameters": {
            "endpoint": "<your_computer_vision_endpoint>",
            "computerVisionApiKey": "<your_computer_vision_key>",
            "indexName": "<name_of_your_index>",
            "videoUrls": ["<your_video_SAS_URL>"]
        }
    }],
    "messages": [
        {
            "role": "system",
            "content": "You are a helpful assistant."
        },
        {
            "role": "user",
            "content": [
                    {
                        "type": "text",
                        "text": "Describe this video:"
                    },
                    {
                        "type": "acv_document_id",
                        "acv_document_id": "<your_video_ID>"
                    }
                ]
        }
    ],
    "max_tokens": 100,
}
```

    The request includes the `enhancements` and `dataSources` objects. `enhancements` represents the specific Vision enhancement features requested in the chat. `dataSources` represents the Computer Vision resource data that's needed for Vision enhancement. It has a `type` property which should be `"AzureComputerVisionVideoIndex"` and a `parameters` property which contains your AI Vision and video information.

3. Fill in all the `<placeholder>` fields above with your own information: enter the endpoint URLs and keys of your OpenAI and AI Vision resources where appropriate, and retrieve the video index information from the earlier step.

4. Send the POST request to the API endpoint. It should contain your OpenAI and AI Vision credentials, the name of your video index, and the ID and SAS URL of a single video.

The chat responses you receive from the model should include information about the video. The API response should look like the following.

```
{
    "id": "chatcmpl-8V4J2cFo7TWO7rIfs47XuDzTKvbct",
    "object": "chat.completion",
    "created": 1702415412,
    "model": "gpt-4",
    "choices":
    [
        {
            "finish_reason":"stop",
            "index": 0,
            "message":
            {
                "role": "assistant",
                "content": "The advertisement video opens with a blurred background that suggests a serene and aesthetically pleasing env
            }
        }
    ],
    "usage":
    {
        "prompt_tokens": 2068,
        "completion_tokens": 341,
        "total_tokens": 2409
    }
}
```

Every response includes a `"finish_details"` field. It has the following possible values:

- `stop`: API returned complete model output.
- `length`: Incomplete model output due to the `max_tokens` input parameter or model's token limit.
- `content_filter`: Omitted content due to a flag from our content filters.

Pricing example for Video prompts

The pricing for GPT-4 Turbo with Vision is dynamic and depends on the specific features and inputs used. For a comprehensive view of Azure OpenAI pricing see [Azure OpenAI Pricing](#).

The base charges and additional features are outlined below:

Base Pricing for GPT-4 Turbo with Vision is:

- Input: $0.01 per 1000 tokens
- Output: $0.03 per 1000 tokens

Video prompt integration with Video Retrieval Add-on:

- Ingestion: $0.05 per minute of video
- Transactions: $0.25 per 1000 queries of the Video Retrieval indexer

- Learn more about Azure OpenAI.
- GPT-4 Turbo with Vision quickstart
- GPT-4 Turbo with Vision frequently asked questions
- GPT-4 Turbo with Vision API reference