

26 | Visual ChatGPT是如何做到边聊边画的？

2023-05-10 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

过去三讲里，我们分别体验了 CLIP、Stable Diffusion 和 ControlNet 这三个模型。我们用这些模型来识别图片的内容，或者通过输入一段文本指令来画图。这些模型都是所谓的多模态模型，能够把图片和文本信息联系在一起。

不过，如果我们不仅仅是要随便找几个关键词画两张画玩个票，而是要在实际的工作环境里生成能用的图片，那么现在的体验还是远远不够的。对于画出来的图我们总有各种各样的修改和编辑的需求。比如，我们总是会遇到各个团队的人对着设计师的图指手画脚地提出各种各样的意见：“能不能把小狗移到图片的右边？” “能不能把背景从草地改成森林？” “我想要一个色彩斑斓的黑。” 等等。

所以，理想中的 AI 画画的功能，最好还能配上一个听得懂人话的 AI，能够根据我们这些外行的指手画脚来修改生成的图片。针对这个需求，我们就来介绍一下微软开源的 Visual

ChatGPT。

和之前我们自己写代码不同，这一讲我们一起来读一读 [🔗Visual ChatGPT](#) 这个开源项目的代码，看看它是如何做到能让我们聊着天就把图片给修改完了的。

体验 Visual ChatGPT

我们先来体验一下 Visual ChatGPT 的效果是怎么样的。这一次，Colab 里的 GPU 也不够我们用了。Visual ChatGPT 要加载很多个不同的图片相关的模型，这些模型加起来的显存得有 40GB 以上。

好在，微软通过 HuggingFace 的 Space 功能提供了一个 [🔗免费的 Space](#)，让你可以直接体验 Visual ChatGPT 的功能。不过，考虑到用的人很多，使用的过程中你的请求会被排队处理，往往要等待很长时间才能完成一条指令。所以，我建议你花个几美元的小钱，部署一个自己的 Visual ChatGPT 的 Space 来体验一下它的功能。

Visual ChatGPT

This is a demo to the work [Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models](#).

This space connects ChatGPT and a series of Visual Foundation Models to enable sending and receiving images during chatting.

Language



Chinese



English

.....

Visual ChatGPT

Examples

Generate a figure of a cat running in the garden

Replace the cat with a dog

Remove the dog in this image

Can you detect the canny edge of this image?

Can you use this canny image to generate an oil painting of a dog

Make it like water-color painting

What is the background color

Describe this image

please detect the depth of this image

Can you use this depth image to generate a cute dog

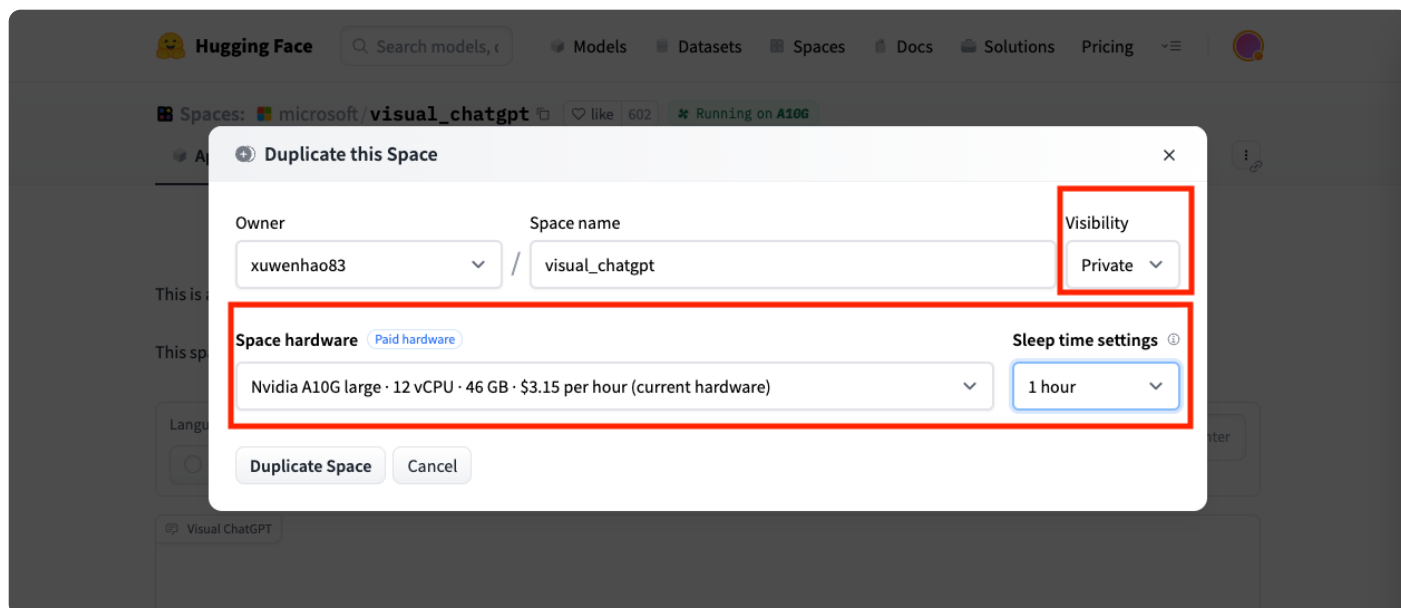
You can duplicate this Space to skip the queue:



Duplicate Space

微软在 HuggingFace 的 Space 里提供了一个可以免费体验的 Visual ChatGPT

你可以点击微软提供的 Space 底部的 Duplicate Space，部署一个完全一样的 Visual ChatGPT Space 到自己的账号下，这样你就不用和其他人排队等着了。



因为我们复制的是原先的 Space，所以对应的硬件配置也是和微软免费提供的一样。通过一块 46G 显存的 A10 显卡，我们可以直接装载所有用到的模型。不过，使用 A10 显卡的 Space 是有成本的，一个小时就要花去你 3.15 美元。所以在复制 Space 的时候，有两个参数你需要注意。

第一个是 Sleep time settings，我建议你设置成 5 分钟。这样，一旦 5 分钟没有人使用这个 Space，它就会进入休眠状态。而 HuggingFace 在休眠状态下就不会收取你任何费用。只是下一次你要使用这个 Space 的时候，会重新启动整个 Space，需要一点点时间。

另一个是 Space 的 Visibility，我建议你选择 Private。这样，这个 Space 只有你能看到。不然的话，就算你设置了自动休眠，也免不了有人看到你的 Space 上来试一试。如果不断有人来使用你的 Space 的话，它会一直在线运行，而你就是那个付账单的人。

在复制的 Space 部署完成之后，你就可以先来试一下 Visual ChatGPT 了。首先你需要在右上角输入你的 OpenAI 的 API Key，并按下回车键。这样，整个窗口的下方就会出现可以输入文本的聊天窗口。你可以选择自由输入你想要画的内容，也可以在下方的 Examples 里选择预设好的一些指令。

Visual ChatGPT

This is a demo to the work [Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models](#).

This space connects ChatGPT and a series of Visual Foundation Models to enable sending and receiving images during chatting.

Language

☐ Chinese ☒ English

Visual ChatGPT

Enter text and press enter, or upload an image

Run

Clear

Upload

Examples

Generate a figure of a cat running in the garden

Replace the cat with a dog

Remove the dog in this image

Can you detect the canny edge of this image?

Can you use this canny image to generate an oil painting of a dog

Make it like water-color painting

What is the background color

Describe this image

please detect the depth of this image

Can you use this depth image to generate a cute dog

输入 API KEY 之后一定要按下回车，否则下面的对话输入框无法出现

我们可以分别试一下 Examples 里面给出的指令。

1. 我们先让 Visual ChatGPT 画一幅小猫在花园里奔跑的图片。

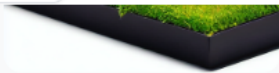
Generate a figure of a cat running in the garden

I have generated an image of a cat running in the garden, the file name is



image/3fad8569.png.

2. 然后再把画面里的小猫换成小狗。



image/3fad8569.png.

Replace the cat with a dog

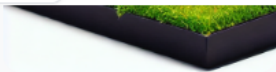
I have replaced the cat with a dog in the image, the new file name is



image/3ed5_replace-something_3fad8569_3fad8569.png.

3. 接着去掉画面中的小狗。

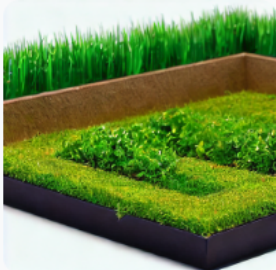
Visual ChatGPT



image/3ed5_replace-something_3fad8569_3fad8569.png.

Remove the dog in this image

I have removed the dog from the image, the new file name is



image/b3d6_replace-something_3ed5_3fad8569.png.

4. 再把图片风格换成水彩画。

Visual ChatGPT



image/b3d6_replace-something_3ed5_3fad8569.png.

Make it like water-color painting

I have generated an image of a water-color painting, the file name is



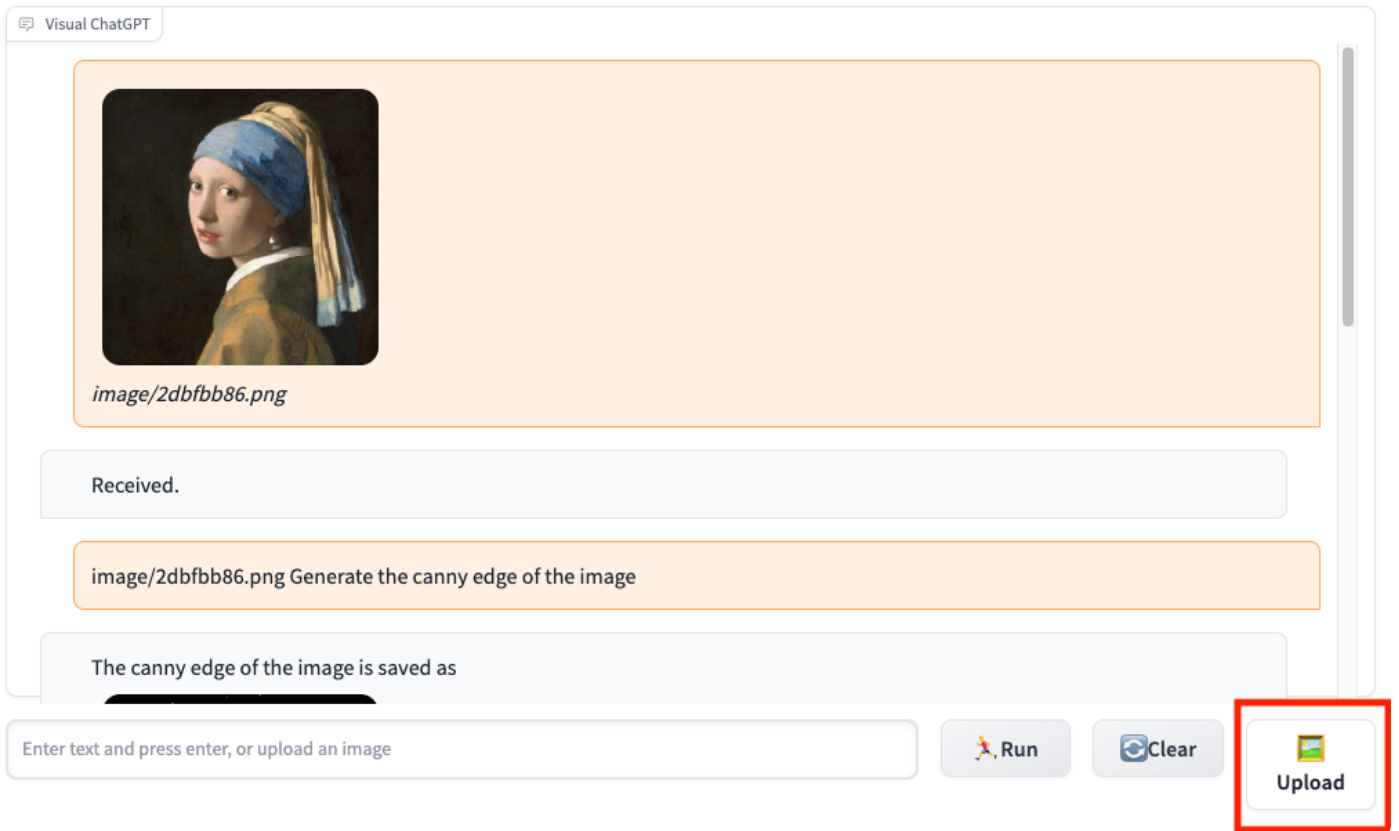
image/4657_pix2pix_b3d6_3fad8569.png.

5. 最后我们让 Visual ChatGPT 描述一下图片的内容。



可以看到，Visual ChatGPT 很好地完成了我们的每一条指令。而这样的交互体验大大提升了我们用 AI 画画的实际体验。比如，上一讲里我们通过 Canny 算法获取了“戴珍珠耳环的少女”画像的轮廓，然后通过 ControlNet 绘制了同样姿势的其他明星的头像。原本这个过程我们是通过一系列的代码来实现的，现在我们完全可以用 Visual ChatGPT，通过一系列的对话向 AI 下达指令来实现。

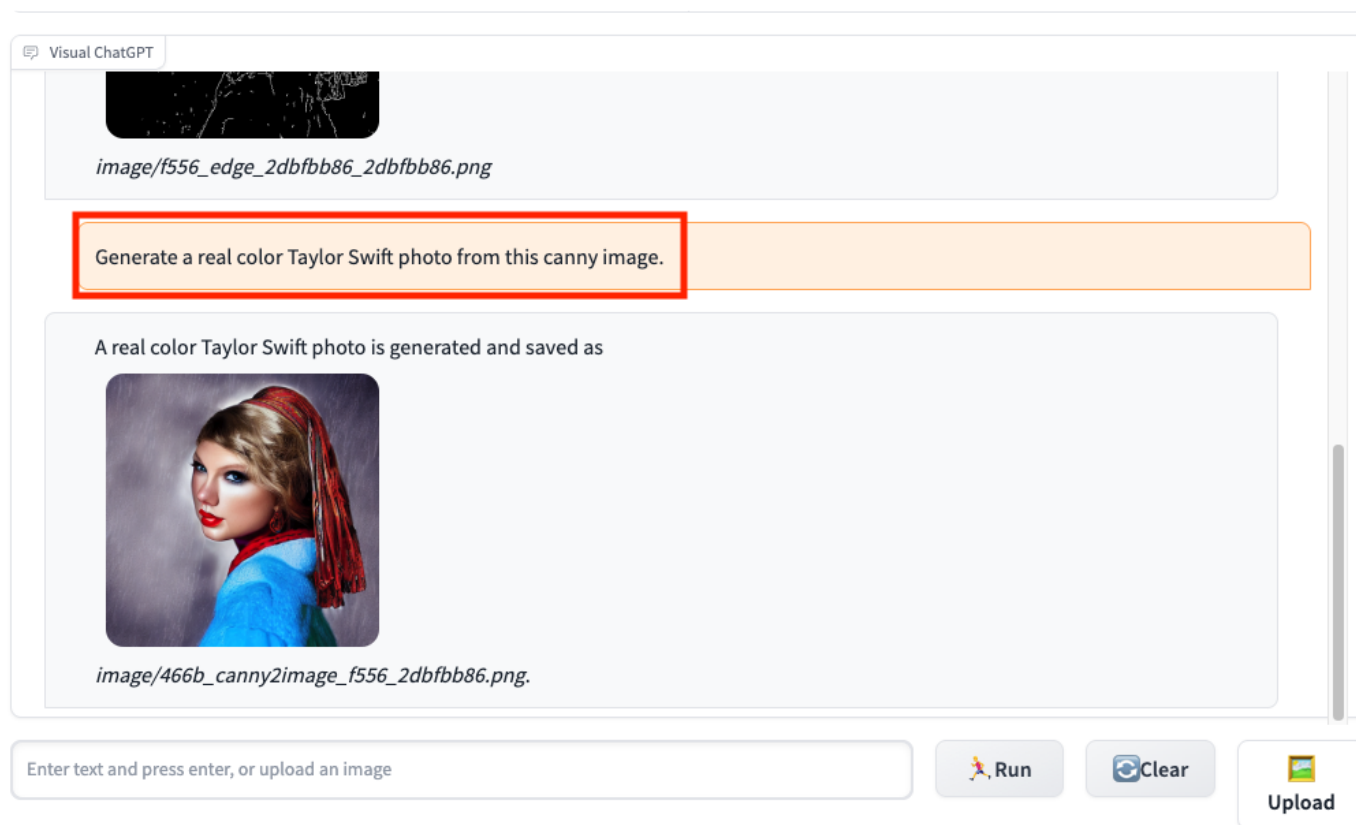
首先，我们通过 Upload 按钮把“戴珍珠耳环的少女”图片上传上去。



接着，我们在对话框里输入文本 “Generate the canny edge of the image”，Visual ChatGPT 就会把上传图片的边缘提取出来，并且生成一张新的图片。



最后，我们在对话框里输入 “Generate a real color Taylor Swift photo from this canny image” ， Visual ChatGPT 就会基于上面边缘检测的轮廓图生成一个新的人像图。



这样，我们不需要撰写任何代码，通过一个聊天窗口就能完成对图片的编辑和修改工作。

Visual ChatGPT 的原理与实现

Visual ChatGPT 的效果非常神奇，但是其实内部原理却非常简单。Visual ChatGPT 解决问题的办法就是使用 [第 17 讲](#) 我们介绍过的 LangChain 的 ReAct Agent 模式，它做了这样几件事情。

1. 它把各种各样图像处理的视觉基础模型（Visual Foundation Model）都封装成了一个 Tool。
2. 然后，将这些 Tool 都交给了一个 conversation-react-description 类型的 Agent。每次你输入文本的时候，其实就是和这个 Agent 在交流。Agent 接收到你的文本，就要判断自己应该使用哪一个 Tool，还有应该从输入的内容里提取什么参数给到这个 Tool。这些输入

参数中既包括需要修改哪一个图片，也包括使用什么样的提示语。这里的 Agent 背后使用的就是 ChatGPT。

3. 最后，Agent 会实际去调用这个 Tool，生成一张新的图片返回给你。

那接下来，我们就进入 Visual ChatGPT 的代码，来看一下具体的代码是怎么做的。Visual ChatGPT 的源代码只有一个文件 [visual_chatgpt.py](#)。整个文件从头到尾可以分成四个部分。

1. 一系列预先定义好的 ChatGPT 的 Prompt，以及一些会被调用的辅助函数。
2. 一系列视觉模型的 Class，每一个 Class 都代表了一个或者多个图片处理的工具。
3. 一个叫做 ConversationBot 的 Class，实际封装了通过对话调用各种视觉模型工具的流程。
4. 实际从命令行启动整个应用的入口，其实就是对 ConversationBot 提供了一个 Gradio 应用的封装。

Visual ChatGPT 的调用入口

对于源码，我们可以倒过来从下往上看，从 Visual ChatGPT 的启动入口来学习代码。对应的启动代码其实很简单，就是干了两件事情。

1. 从命令行加载了 load 参数，并且把对应的字符串解析成一个 load_dict，然后通过 load_dict 创建了 ConversationBot。

 复制代码

```
1 if __name__ == '__main__':
2     if not os.path.exists("checkpoints"):
3         os.mkdir("checkpoints")
4     parser = argparse.ArgumentParser()
5     parser.add_argument('--load', type=str, default="ImageCaptioning_cuda:0,Text2
6     args = parser.parse_args()
7     load_dict = {e.split('_')[0].strip(): e.split('_')[1].strip() for e in args.l
8     bot = ConversationBot(load_dict=load_dict)
9     .....
```

2. 然后将整个 ConversationBot 做成了一个有界面的 Gradio 应用。

📄 复制代码

```
1 .....
2 with gr.Blocks(css="#chatbot .overflow-y-auto{height:500px}") as demo:
3     lang = gr.Radio(choices = ['Chinese','English'], value=None, label='Langu
4     chatbot = gr.Chatbot(elem_id="chatbot", label="Visual ChatGPT")
5     state = gr.State([])
6     with gr.Row(visible=False) as input_rows:
7         with gr.Column(scale=0.7):
8             txt = gr.Textbox(show_label=False, placeholder="Enter text and pr
9                 container=False)
10        with gr.Column(scale=0.15, min_width=0):
11            clear = gr.Button("Clear")
12        with gr.Column(scale=0.15, min_width=0):
13            btn = gr.UploadButton(label="🖼️",file_types=["image"])
14
15        lang.change(bot.init_agent, [lang], [input_rows, lang, txt, clear])
16        txt.submit(bot.run_text, [txt, state], [chatbot, state])
17        txt.submit(lambda: "", None, txt)
18        btn.upload(bot.run_image, [btn, state, txt, lang], [chatbot, state, txt])
19        clear.click(bot.memory.clear)
20        clear.click(lambda: [], None, chatbot)
21        clear.click(lambda: [], None, state)
22    demo.launch(server_name="0.0.0.0", server_port=7860)
```

我们看一下 load 参数，其实就是指定了不同的工具类应该使用 CPU 还是 GPU，以及对应的模型应该加载到哪一个 GPU 上。

📄 复制代码

```
1 python visual_chatgpt.py --load "Text2Box_cuda:0,Segmenting_cuda:0,
2     Inpainting_cuda:0,ImageCaptioning_cuda:0,
3     Text2Image_cuda:1,Image2Canny_cpu,CannyText2Image_cuda:1,
4     Image2Depth_cpu,DepthText2Image_cuda:1,VisualQuestionAnswering_cuda:2,
5     InstructPix2Pix_cuda:2,Image2Scribble_cpu,ScribbleText2Image_cuda:2,
6     SegText2Image_cuda:2,Image2Pose_cpu,PoseText2Image_cuda:2,
7     Image2Hed_cpu,HedText2Image_cuda:3,Image2Normal_cpu,
8     NormalText2Image_cuda:3,Image2Line_cpu,LineText2Image_cuda:3"
```

ConversationBot, 一个 Langchain Agnet

接下来我们再来看看控制 Visual ChatGPT 的核心运转流程的 ConversationBot 是什么样的。ConversationBot 里面包含了四个函数，分别是 `__init__` 的构造函数、`init_agent` 函数、`run_text` 和 `run_image` 函数。

`__init__` 的构造函数用来加载使用的各个视觉基础模型 (Visual Foundation Model) 。

`init_agent` 函数构造了一个 LangChain 的 `conversation-react-description` 类型的 Agent，用来实际处理整个 AI 对话过程。

`run_text` 处理用户的文本输入。

`run_image` 处理用户的图片输入。

`__init__` 构造函数

我们先来看看 `__init__` 这个构造函数。

```
1 def __init__(self, load_dict):
2     # load_dict = {'VisualQuestionAnswering':'cuda:0', 'ImageCaptioning':'cuda:0'}
3     print(f"Initializing VisualChatGPT, load_dict={load_dict}")
4     if 'ImageCaptioning' not in load_dict:
5         raise ValueError("You have to load ImageCaptioning as a basic function")
6
7     self.models = {}
8     # Load Basic Foundation Models
9     for class_name, device in load_dict.items():
10         self.models[class_name] = globals()[class_name](device=device)
11
12     # Load Template Foundation Models
13     for class_name, module in globals().items():
14         if getattr(module, 'template_model', False):
15             template_required_names = {k for k in inspect.signature(module.__init__).parameters if k != 'self'}
16             loaded_names = set([type(e).__name__ for e in self.models.values()])
17             if template_required_names.issubset(loaded_names):
18                 self.models[class_name] = globals()[class_name](
19                     **{name: self.models[name] for name in template_required_names})
20
21     print(f"All the Available Functions: {self.models}")
22
23 .....
```

代码其实很简单，首先就是将前面从命令行读入的 `load_dict` 里面的每一个 Class 都实例化，后面我们会看到，在实例化的过程中，这些 Class 都会把各种预训练好的模型加载到 CPU 或者 GPU 里面来。

这其中有一个情况需要注意，有些模型 Class 在我们这里叫做 **template_model**，其实就是能够组合多个模型组合，来解决一个单一的任务。这些 Class 不是通过命令行传入的参数名称来加载的，而是去判断这个 Class 需要的其他模型是否都已经加载了，如果都加载了，那么这个 Class 自然可以用，不需要额外占用显存或者内存。否则的话，这个 Class 就不会被加载。

接下来就是遍历所有的这些 Class，找到里面以 `inference` 开头的函数。每一个函数都会被当作是一个 `LangChain` 里面的 `Tool`，放到当前实例的 `Tools` 数组中去。

然后就创建了 Agent 需要的 LLM 和 Memory，如果你想要用 **GPT-4** 来管理对话过程以取得更好的效果，你就可以在这里用 **GPT-4** 替换掉 LLM 来做到这一点。

```

1 .....
2     self.tools = []
3     for instance in self.models.values():
4         for e in dir(instance):
5             if e.startswith('inference'):
6                 func = getattr(instance, e)
7                 self.tools.append(Tool(name=func.name, description=func.descr
8 self.llm = OpenAI(temperature=0)
9 self.memory = ConversationBufferMemory(memory_key="chat_history", output_

```

init_agent 函数

接下来的 `init_agent` 函数就特别简单了，它其实就是利用上面我们加载的 Tools、Memory 以及 LLM 创建了一个 conversational-react-description 类型的 Agent。这个 Agent 我们在 [第 17 讲](#) 其实已经介绍过了一遍。

不过，这里我们通过 `agent_kwargs` 为这个 Agent 专门定制了对应的提示语。这部分提示语我们晚一点再介绍。这里，对于中文和英文 Visual ChatGPT 只是通过简单的 `if...else` 提供了一组不同的提示语而已。


```

1     def init_agent(self, lang):
2         self.memory.clear() #clear previous history
3         if lang=='English':
4             PREFIX, FORMAT_INSTRUCTIONS, SUFFIX = VISUAL_CHATGPT_PREFIX, VISUAL_C
5             place = "Enter text and press enter, or upload an image"
6             label_clear = "Clear"
7         else:
8             PREFIX, FORMAT_INSTRUCTIONS, SUFFIX = VISUAL_CHATGPT_PREFIX_CN, VISUA
9             place = "输入文字并回车，或者上传图片"
10            label_clear = "清除"
11        self.agent = initialize_agent(
12            self.tools,
13            self.llm,
14            agent="conversational-react-description",
15            verbose=True,
16            memory=self.memory,
17            return_intermediate_steps=True,
18            agent_kwargs={'prefix': PREFIX, 'format_instructions': FORMAT_INSTRUC
19                          'suffix': SUFFIX}, )

```


run_text 和 run_image 函数

实际处理对话的 run_text 和 run_image 函数也非常简单。run_text 函数就是先确保 Memory 不要超出我们设置的上下文长度的限制。然后直接调用 Agent 来应对用户输入的文本。并且对于输出的结果，它只是做了一些文件名上的字符串显示的处理而已。

 复制代码

```
1 def run_text(self, text, state):
2     self.agent.memory.buffer = cut_dialogue_history(self.agent.memory.buffer,
3     res = self.agent({"input": text.strip()})
4     res['output'] = res['output'].replace("\\", "/")
5     response = re.sub('(image/[-\w]*.png)', lambda m: f'})
6     state = state + [(text, response)]
7     print(f"\nProcessed run_text, Input text: {text}\nCurrent state: {state}\
8         f"Current Memory: {self.agent.memory.buffer}")
9     return state, state
```

而 run_image 函数，则是把用户上传的图片转换成完全相同的尺寸和格式。此外，它还会使用 ImageCaptioning 这个模型拿到图片的描述。最后，模型将图片名称和描述等信息也作为一轮对话拼接接到 Agent 的 memory 里面去。

 复制代码

```
1
2 def run_image(self, image, state, txt, lang):
3     image_filename = os.path.join('image', f"{str(uuid.uuid4())[:8]}.png")
4     print("=====>Auto Resize Image...")
5     img = Image.open(image.name)
6     width, height = img.size
7     ratio = min(512 / width, 512 / height)
8     width_new, height_new = (round(width * ratio), round(height * ratio))
9     width_new = int(np.round(width_new / 64.0)) * 64
10    height_new = int(np.round(height_new / 64.0)) * 64
11    img = img.resize((width_new, height_new))
12    img = img.convert('RGB')
13    img.save(image_filename, "PNG")
14    print(f"Resize image form {width}x{height} to {width_new}x{height_new}")
15    description = self.models['ImageCaptioning'].inference(image_filename)
16    if lang == 'Chinese':
```



```

17         Human_prompt = f'\nHuman: 提供一张名为 {image_filename}的图片。它的描述是:
18         AI_prompt = "收到。  "
19     else:
20         Human_prompt = f'\nHuman: provide a figure named {image_filename}. Th
21         AI_prompt = "Received.  "
22     self.agent.memory.buffer = self.agent.memory.buffer + Human_prompt + 'AI:
23     state = state + [(f"*{image_filename}*", AI_pro
24     print(f"\nProcessed run_image, Input image: {image_filename}\nCurrent sta
25         f"Current Memory: {self.agent.memory.buffer}")
26     return state, state, f'{txt} {image_filename} '
27


```

Visual Foundation Model, 实际处理图片的工具

看完了 ConversationBot, 我们就知道其实我们向 Visual ChatGPT 输入的各种文本指令, 都会变成对某一个视觉基础模型 (Visual Foundation Model) 的调用。那么, 我们就挑一两个视觉基础模型, 来看看具体里面是如何调用的。

我们还是拿之前我们比较熟悉的通过 Canny 算法进行边缘检测的 CannyText2Image 来演示好了。在这个 Class 的构造函数里, 我们还是通过 Diffusers 的 Pipeline 加载了 Stable Diffusion 和 ControlNet 的模型。这样, 后面我们就可以用这个 Pipeline 来对图片进行了。

另外, 在构造函数的最后, 它还设置了一系列正面和负面的提示语内容。负面的提示语用于排除低质量的图片, 而正面的提示语则会和用户输入的提示语拼接到一起, 用来生成图片。

 复制代码

```


1 class CannyText2Image:
2     def __init__(self, device):
3         print(f"Initializing CannyText2Image to {device}")
4         self.torch_dtype = torch.float16 if 'cuda' in device else torch.float32
5         self.controlnet = ControlNetModel.from_pretrained("fusing/stable-diffusio
6                                     torch_dtype=self.torch_
7         self.pipe = StableDiffusionControlNetPipeline.from_pretrained(
8             "runwayml/stable-diffusion-v1-5", controlnet=self.controlnet, safety_
9             torch_dtype=self.torch_dtype)
10        self.pipe.scheduler = UniPCMultistepScheduler.from_config(self.pipe.sched
11        self.pipe.to(device)
12        self.seed = -1
13        self.a_prompt = 'best quality, extremely detailed'

```

```
14         self.n_prompt = 'longbody, lowres, bad anatomy, bad hands, missing finger
15                           fewer digits cropped worst quality low quality'
```

除了构造函数，这个 Class 还有一个 inference 函数。这个函数通过 Prompts 这个 decorator 定义了 name 和 description 属性，这些属性也是我们实际加载 Tools 的时候使用的参数。Agent 就会根据这些描述判断用户输入的文本是否应该使用当前这个工具。比如这里 CannyText2Image 的 description 里，就告诉你用户一般会通过 “generate a real image of a object or something from this canny image” 这样的指令来调用当前的工具。

而对应的 inference 函数的内容，就是简单地根据输入的文本调用模型来处理图片。不过要注意，这里作为输入的 Inputs 文本，并不是用户原始输入的内容。而是通过 ConversationBot 的 Agent，经过 “思考” 之后拿到的 Action Inputs。这个 Inputs 里面，既会包含需要处理的图片路径，也会包含对应的 Prompts。

 复制代码

```
1  @prompts(name="Generate Image Condition On Canny Image",
2         description="useful when you want to generate a new real image from
3                     " like: generate a real image of a object or something f
4                     " or generate a new real image of a object or something
5                     "The input to this tool should be a comma separated stri
6                     "representing the image_path and the user description. "
7  def inference(self, inputs):
8      image_path, instruct_text = inputs.split(",")[0], ','.join(inputs.split('
9      image = Image.open(image_path)
10     self.seed = random.randint(0, 65535)
11     seed_everything(self.seed)
12     prompt = f'{instruct_text}, {self.a_prompt}'
13     image = self.pipe(prompt, image, num_inference_steps=20, eta=0.0, negativ
14                  guidance_scale=9.0).images[0]
15     updated_image_path = get_new_image_name(image_path, func_name="canny2imag
16     image.save(updated_image_path)
17     print(f"\nProcessed CannyText2Image, Input Canny: {image_path}, Input Tex
18           f"Output Text: {updated_image_path}")
19     return updated_image_path
```


当然，不是所有模型 Class 的 inference 函数都这么简单。有些 Class，特别是我们前面介绍过的 template_model 的 Class，它的 inference 函数会复杂一些，需要多次调用多个模型组

合来完成任务。比如 InfinityOutPainting 这个 Class，就需要不断循环调用 VisualQuestionAnswering 和 ImageCaption 来获取图片的描述，然后通过 Inpainting 来补全图片中没有画出来的部分，最终实现把图片无限扩大，补全扩大出来的部分背景的能力。

复盘 Prompt，理解 Task Matrix 机制

我们刚才说过，每个 Class 模型拿到的 inputs 输入都是模型“思考”之后根据用户输入提取出来的 Action Inputs。这一点，其实还是要归功于 ChatGPT 强大的逻辑推理能力。我们只要回到代码的开头，看一下对应的 Prompts 其实就能知道 Visual ChatGPT 是怎么做到的了。实际上，每次用户输入的内容，都是通过 VISUAL_CHATGPT_PREFIX、VISUAL_CHATGPT_FORMAT_INSTRUCTIONS 和 VISUAL_CHATGPT_SUFFIX 这三段 Prompts 拼接而成的。

而 AI 的思考过程，其实就是 VISUAL_CHATGPT_FORMAT_INSTRUCTIONS 这一小段。

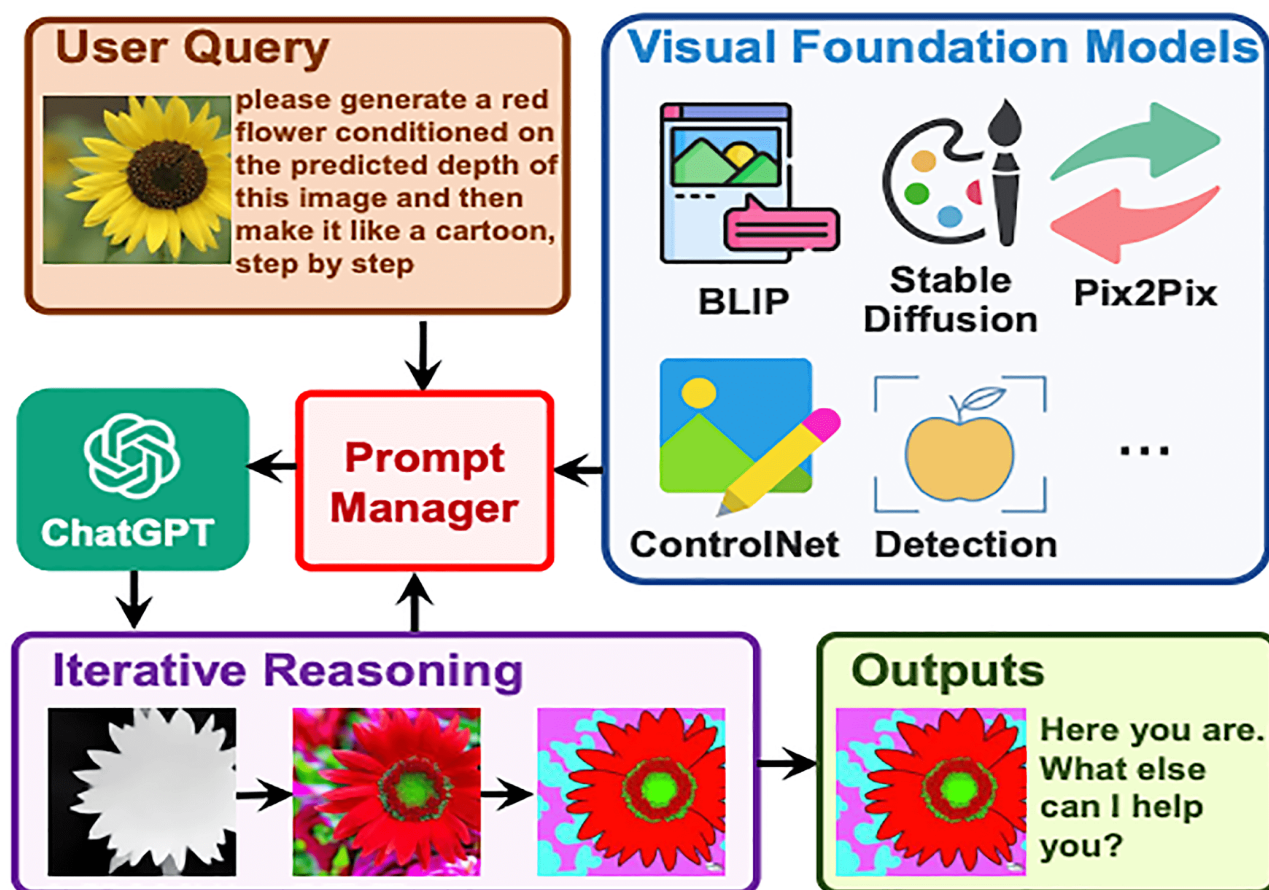
 复制代码

```
1 VISUAL_CHATGPT_FORMAT_INSTRUCTIONS = """To use a tool, please use the following f
2
3 Thought: Do I need to use a tool? Yes
4 Action: the action to take, should be one of [{tool_names}]
5 Action Input: the input to the action
6 Observation: the result of the action
7
8 When you have a response to say to the Human, or if you do not need to use a tool
9
10 Thought: Do I need to use a tool? No
11 {ai_prefix}: [your response here]
12
13 """
```

这个其实就是我在 [第 17 讲](#) 里给你看过的 MRKL 的提示语模版。AI 通过 Thought、Action、Action Input 和 Observation 这样的四轮循环，完成一次 Tools 的判定与调用。并且，每次给到 Agent 的输入里，都可以包含多个迭代的 Action 和 Action Input，调用多次模型 Class 来解决问题。

小结

看完这个代码之后，相信你完全有能力修改代码满足自己的需求了。如果有一天出现了一个更好用的视觉基础模型，你完全可以把这个新的模型 Class 加入到 Visual ChatGPT 中。你只需要通过 `__init__` 函数加载模型，然后定义好它对应的提示语以及 inference 函数，就能让 Visual ChatGPT 支持一种新的图片编辑和绘制功能了。



来自 Visual ChatGPT 论文中的图 1

回顾整个 Visual ChatGPT 的代码，其实并不复杂。它就是将 [第 17 讲](#) 我们介绍过的 LangChain 的 Agent，和过去 3 讲我们介绍的各种视觉大模型组合起来，通过 ChatGPT 的语言和逻辑推理能力处理用户输入，通过 LangChain 的 Agent 机制来调度推理过程和工具的使用，通过视觉大模型实际来处理图片以及理解图片的内容。

这样的机制其实不仅可以用来处理图片，也可以用在其他机器学习的模型里，比如语音、视频，甚至你还可以利用它再去调用别的大语言模型。而这个机制，后来也被微软进一步扩展到 Task Matrix 这个概念里。Visual ChatGPT 的代码库的名称今天也被改成了 Task Matrix，也就是“任务矩阵”的意思。

思考题

最后，按照惯例还是给你留一道思考题。

Stable Diffusion 的提示语是不支持中文的，但是 Visual ChatGPT 支持你通过中文输入你对图片的绘制和修改要求。你觉得它目前的实现有没有什么问题？如果我们想要让它不只支持中文和英文，也能支持德文、法文、西班牙文等各个语种，我们可以怎么做？欢迎你把思考后的结果分享到评论区，我们一起讨论，也欢迎你把这一讲分享给需要的朋友，我们下一讲再见！

推荐阅读

在 Visual ChatGPT 之后，微软更进一步将这个概念扩展成为 Task Matrix，也对应发表了一篇 [论文](#)。你有兴趣的话，可以去阅读一下。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (5)



Toni

2023-05-10 来自瑞士

人工智能的快速发展带来各类相关模型的数量爆炸性增长，选择的多样性使得应用者总能挑选出几样趁手的工具，但同时也带来了一大痛点，哪个模型是最好的呢？Visual ChatGPT 的核心板块“任务矩阵”(Task Matrix)就是想为客户自动选出适合解决任务的模型。诚如论文的作者指出的“TaskMatrix.AI 可以理解这些 API 并学习新的 API，然后根据用户说明推荐合适的API。作者举例中的用对话的形式帮助用户生成 PPT，并根据要求不断修改，展示了 TaskMatrix.AI 方便之处，过程流畅，但实际上这里隐含了一个前提，即用户的问题非常明确，比如：“对于每家公司，让我们创建一张幻灯片来介绍它的创始人、位置、使命、产品、子公司” (Fig.6, 7)，在这样的情况下，TaskMatrix.AI 能给出准确的反应；还有 Fig.3 的绘图任务，Fig.9 的智能家居场景也是如此。

ChatGPT3.5 以后的版本 AI 能够表现出很强的“推理能力”，这一能力本质是将自然语言中的‘字’，‘语义’，‘句子’，‘位置’，‘段落’等，经过大量的监督的无监督的“阅读学习”，在映射的高维空间中不断调整优化所形成的不断“进化”的模型，然后根据人们给出的问题或曰提示，“猜出”或称“推断出”后面的意思。通常情况下 AI 有亮丽的表现，虽然不免夸夸其谈。但这带来 AI 另一大痛点，它不能对不清楚的问题进行反问。比如在用户提出这样的问题时：“你知

道如何从头开始写一篇文章并提供给我一个解决方案大纲？我有几篇论文的截止日期。我只有题目以及每个的一些要点。最好在其中包含图像。"(Fig.4) TaskMatrix.AI 并没有提问"你的题目是什么?"，而是直接开聊。

AI 日新月异，各种模型为应对使用痛点而生，期待。

老师已先人一步用上ChatGPT4，它有反问或帮助用户捋清问题思路的能力了吗？

作者回复: GPT-4用了一段时间了，如果AutoGPT之类的流程，的确可以让AI反向和你确认问题。

但是如果不是专门在Prompt里面提到，默认还是会习惯于用“遵循指令”的方式，而不是来和你澄清问题。



👍 6



王永旺

2023-05-10 来自日本

Semantic Kernel 老师有了解么，能不能也介绍一下这个项目？

作者回复: 大概看了一点点，本质上就是一个微软版本的“LangChain”，看起来在利用大语言模型的“计划能力”上有些特色。其他的功能，虽然API和Langchain有差别，但是其实本质上差异不大，一个基于LLM能力的中间层。



👍 1



yu

2023-05-16 来自美国

剛看完先留言：先讓他判斷用戶的語言，然後翻譯成英文處理。

作者回复: 对，这的确是一个更加通用的办法。



peter

2023-05-10 来自北京

请教老师：网站想提供几首歌，是否有合适的软件能实现？即用软件唱歌，声音随便，用人声或机器自己产生的声音都可以。不仅仅限于chatGPT；如果chatGPT没有该功能，是否有其他软件能实现？

作者回复: <https://github.com/svc-develop-team/so-vits-svc>



金□

2023-05-10 来自广东

有编辑音频的chatgpt吗？最近ai孙燕姿很火，效果也不错，但是跟本人唱的还是差一些东西，旋律比较平，有办法调教吗？

作者回复: AI孙燕姿来自于 <https://github.com/svc-develop-team/so-vits-svc> 这开源项目下根据数据训练出来的

调教可能要熟悉音乐领域知识的朋友们会有更好的idea，我是个音盲

