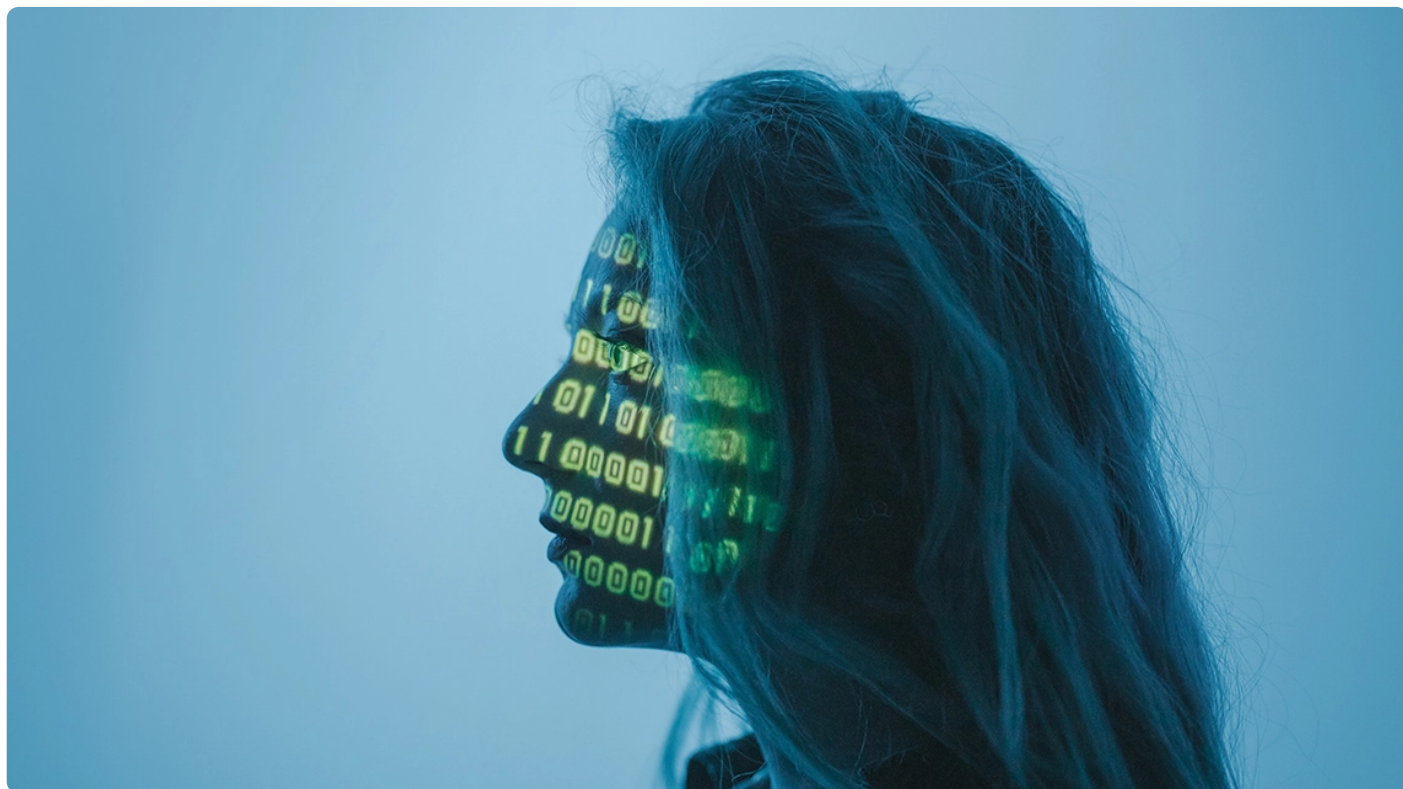


22 | 再探HuggingFace：一键部署自己的大模型

2023-04-26 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

过去几讲里，我们一起为 AI 加上了语音能力。而且相对于大语言模型，语音识别和语音合成都有完全可以用于商业应用的开源模型。事实上，Huggingface 的火爆离不开他们开源的这个 Transformers 库。这个开源库里有数万个我们可以直接调用的模型。很多场景下，这个开源模型已经足够我们使用了。

不过，在使用这些开源模型的过程中，你会发现大部分模型都需要一块不错的显卡。而如果回到我们更早使用过的开源大语言模型，就更是这样了。

在课程里面，我们是通过用 Colab 免费的 GPU 资源来搞定的。但是如果我们想要投入生产环境使用，免费的 Colab 就远远不够用了。而且，Colab 的 GPU 资源对于大语言模型来说还是太小了。我们在前面不得不使用小尺寸的 T5-base 和裁剪过的 ChatGLM-6B-INT4，而不是 FLAN-UL2 或者 ChatGLM-130B 这样真正的大模型。

那么，这一讲我们就来看看，Transformers 可以给我们提供哪些模型，以及如何在云端使用真正的大模型。而想要解决这两个问题啊，都少不了要使用 HuggingFace 这个目前最大的开源模型社区。

Transformers Pipeline

Pipeline 的基本功能

我们先来看看，Transformers 这个开源库到底能干些什么。下面的代码都是直接使用开源模型，需要利用 GPU 的算力，所以你最好还是在 Colab 里运行，注意不要忘记把 Runtime 的类型修改为 GPU。

 复制代码

```
1 from transformers import pipeline
2
3 classifier = pipeline(task="sentiment-analysis", device=0)
4 preds = classifier("I am really happy today!")
5 print(preds)
```


输出结果：

 复制代码

```
1 No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english
2 Using a pipeline without specifying a model name and revision in production is no
3 [{'label': 'POSITIVE', 'score': 0.9998762607574463}]
```


这个代码非常简单，第一行代码，我们定义了一个 task 是 sentimental-analysis 的 Pipeline，也就是一个情感分析的分类器。里面 device=0 的意思是我们指定让 Transformer 使用 GPU 资源。如果你想要让它使用 CPU，你可以设置 device=-1。然后，调用这个分类器对一段文本进行情感分析。从输出结果看，它给出了正确的 Positive 预测，也给出了具体的预测分数。因为我们在这里没有指定任何模型，所以 Transformers 自动选择了默认模型，也就是日志里看到的 distilbert-base-uncased-finetuned-sst-2-english 这个模型。

看名字我们可以知道，这个模型是一个针对英语的模型。如果想要支持中文，我们也可以换一个模型来试试。

 复制代码

```
1 classifier = pipeline(model="uer/roberta-base-finetuned-jd-binary-chinese", task=
2 preds = classifier("这个餐馆太难吃了。")
3 print(preds)
```

输出结果：


 复制代码

```
1 [{ 'label': 'negative (stars 1, 2 and 3)', 'score': 0.934112012386322}]
```

这里，我们指定模型的名称，就能换用另一个模型来进行情感分析了。这次我们选用的是 roberta-base-finetuned-jd-binary-chinese 这个模型。RoBERTa 这个模型是基于 BERT 做了一些设计上的修改而得来的。而后面的 finetuned-jd-binary-chinese 是基于京东的数据进行微调过的一个模型。


Pipeline 是 Transformers 库里面的一个核心功能，它封装了所有托管在 HuggingFace 上的模型推理预测的入口。你不需要关心具体每个模型的架构、输入数据格式是什么样子的。我们只要通过 model 参数指定使用的模型，通过 task 参数来指定任务类型，运行一下就能直接获得结果。

比如，我们现在不想做情感分析了，而是想要做英译中，我们只需要把 task 换成 translation_en_to_zh，然后选用一个合适的模型就好了。

 复制代码

```
1 translation = pipeline(task="translation_en_to_zh", model="Helsinki-NLP/opus-mt-e
2
3 text = "I like to learn data science and AI."
4 translated_text = translation(text)
5 print(translated_text)
```

输出结果：

 复制代码

```
1 [{ 'translation_text': '我喜欢学习数据科学和人工智能' }]
```

在这里，我们选用了赫尔辛基大学的 opus-mt-en-zh 这个模型来做英译中，运行一下就可以看到，我们输入的英文被翻译成了中文。不过，我们怎么知道应该选用哪个模型呢？这个如魔法一般的 Helsinki-NLP/opus-mt-en-zh 模型名字从哪里可以找到呢？

如何寻找自己需要的模型？

这个时候，我们就需要去 HuggingFace 的网站里找一找了。你点击网站的 [🔗 Models 板块](#)，就可以看到一个界面，左侧是一系列的筛选器，而右侧则是筛选出来的模型。比如刚才的英译中的模型，我们就是先在左侧的筛选器里，选中 Task 下的 Translation 这种任务类型。然后再在 Languages 里面选择 Chinese，就能找到所有能够翻译中文的模型。默认模型排序是按照用户下载数量从高到低排序的。一般来说，下载的数量越多，往往也意味着大家觉得这个模型可能更加靠谱。



Tasks 1 Libraries Datasets Languages Licenses Other

Filter Tasks by name

Reset Tasks

Multimodal

Feature Extraction Text-to-Image

Image-to-Text Text-to-Video

Visual Question Answering

Document Question Answering

Graph Machine Learning

Computer Vision

Depth Estimation Image Classification

Object Detection Image Segmentation

Image-to-Image

Unconditional Image Generation

Video Classification

Zero-Shot Image Classification

Natural Language Processing

Text Classification Token Classification

Table Question Answering Question Answering

Zero-Shot Classification Translation

Summarization Conversational

Text Generation Text2Text Generation

Fill-Mask Sentence Similarity

Audio

Text-to-Speech Automatic Speech Recognition

Audio-to-Audio Audio Classification

Voice Activity Detection

Tabular

Tabular Classification Tabular Regression

Reinforcement Learning

Reinforcement Learning Robotics

Models 2,008

Filter

new Full-text search

Sort: Most Downloads

t5-base

Updated 11 days ago • 5.76M • 189

Helsinki-NLP/opus-mt-en-es

Updated Jan 24 • 2.6M • 36

t5-small

Updated 11 days ago • 2.17M • 89

Helsinki-NLP/opus-mt-fr-en

Updated Jan 24 • 469k • 15

Helsinki-NLP/opus-mt-en-fr

Updated Jan 24 • 423k • 11

Helsinki-NLP/opus-mt-es-en

Updated Jan 24 • 421k • 20

t5-large

Updated 11 days ago • 240k • 54

Helsinki-NLP/opus-mt-de-en

Updated Jan 24 • 238k • 12

Helsinki-NLP/opus-mt-ru-en

Updated Jan 24 • 222k • 19

facebook/nllb-200-distilled-600M

Updated Feb 12 • 210k • 108

Helsinki-NLP/opus-mt-tc-big-en-pt

Updated Jan 25 • 204k • 8

Helsinki-NLP/opus-mt-en-de

Updated Jan 24 • 182k • 9

Helsinki-NLP/opus-mt-ar-en

Updated Jan 24 • 162k • 9

Helsinki-NLP/opus-mt-ROMANCE-en

Updated Jan 24 • 134k

点击Model的板块，然后选择筛选器里面的Translation任务

Hugging Face Search models, c Models Datasets Spaces Docs Solutions Pricing

Tasks Libraries Datasets Languages Licenses Other

Filter Languages by name Reset Languages

English Spanish French Swedish Finnish German Russian Ukrainian Romanian Italian Dutch Chinese x Arabic Esperanto Bulgarian Polish Catalan Hebrew Japanese

Models 57 Filter Full-text search Sort: Most Downloads

facebook/nllb-200-distilled-600M Updated Feb 12 210k 108

Helsinki-NLP/opus-mt-en-zh Updated Jan 24 132k 95

Helsinki-NLP/opus-mt-zh-en Updated Jan 24 119k 144

点击 Languages，选择Chinese，就能筛选出潜在的合适的模型

我们点击 Helsinki-NLP/opus-mt-en-zh 进入这个模型的卡片页，就能看到更详细的介绍。并且很多模型，都在右侧提供了对应的示例。不使用代码，你也可以直接体验一下模型的能力和效果。

Hugging Face Search models, c Models Datasets Spaces Docs Solutions Pricing

Helsinki-NLP/opus-mt-en-zh like 95

Translation PyTorch TensorFlow JAX Rust Transformers English Chinese marian text2text-generation AutoTrain Compatible License: apache-2.0

Model card Files Community 8 Train Deploy Use in Transformers

eng-zho

- source group: English
- target group: Chinese
- OPUS readme: [eng-zho](#)
- model: transformer
- source language(s): eng
- target language(s): cjt_Hans cjt_Hant cmn cmn_Hans cmn_Hant gan lzh lzh_Hans nan wuu yue yue_Hans yue_Hant
- model: transformer
- pre-processing: normalization + SentencePiece (spm32k,spm32k)

Downloads last month 131,942

Hosted inference API

Translation Examples

My name is Sarah and I live in London

Compute Enter 0.5

Computation time on Intel Xeon 3rd Gen Scalable cpu: cached

我叫莎拉,我住伦敦

JSON Output Maximize

Pipeline 支持的自然语言处理任务

Transformers 的 Pipeline 模块，支持的 task 是非常丰富的。可以说大部分常见的自然语言处理任务都被囊括在内了，经常会用到的有这么几个。

feature-extraction，其实它和 OpenAI 的 Embedding 差不多，也就是把文本变成一段向量。

fill-mask，也就是完形填空。你可以把一句话中的一部分遮盖掉，然后让模型预测遮盖掉的地方的词是什么。

ner，命名实体识别。我们常常用它来提取文本里面的时间、地点、人名、邮箱、电话号码、地址等信息，然后进一步用这些信息来处理其他任务。

question-answering 和 table-question-answering，专门针对问题进行自动问答，在客服的 FAQ 领域常常会用到这类任务。

sentiment-analysis 和 text-classification，也就是我们之前见过的情感分析，以及类目更自由的文本分类问题。


text-generation 和 text2text-generation，文本生成类型的任务。我们之前让 AI 写代码或者写故事，其实都是这一类的任务。

剩下的还有 summarization 文本摘要、translation 机器翻译，以及 zero-shot-classification，也就是我们课程一开始介绍的零样本分类。

看到这里，你有没有发现 ChatGPT 的强大之处？上面这些自然语言处理任务，常常需要切换使用不同的专有模型。但是在 **ChatGPT 里，我们只需要一个通用的模型，就能直接解决所有的问题**。这也是很多人惊呼“通用人工智能”来了的原因。

通过 Pipeline 进行语音识别


Pipeline 不仅支持自然语言处理相关的任务，它还支持语音和视觉类的任务。比如，我们同样可以通过 Pipeline 使用 OpenAI 的 Whisper 模型来做语音识别。

 复制代码

```
1 from transformers import pipeline
```


```
2 transcriber = pipeline(model="openai/whisper-medium", device=0)
3 result = transcriber("./data/podcast_clip.mp3")
4 print(result)
5
```

输出结果：

 复制代码

```
1 {'text': " Welcome to OnBoard, a real first-line experience, a new investment thi
```

不过，这里你会发现一个小小的问题。我们原本中文的内容，在通过 Pipeline 调用 Whisper 模型之后，输出就变成了英文。这个是因为 Pipeline 对整个数据处理进行了封装。在实际调用 Whisper 模型的时候，它会在最终生成文本的过程里面，加入一个 `<|en|>`，导致文本生成的时候强行被指定成了英文。我们可以修改一下这个 decoder 生成文本时的设置，让输出的内容变成中文。

 复制代码

```
1 from transformers import pipeline
2 from transformers import WhisperProcessor, WhisperForConditionalGeneration
3 processor = WhisperProcessor.from_pretrained("openai/whisper-medium")
4 forced_decoder_ids = processor.get_decoder_prompt_ids(language="zh", task="transc
5
6 transcriber = pipeline(model="openai/whisper-medium", device=0,
7                        generate_kwargs={"forced_decoder_ids": forced_decoder_ids}
8 result = transcriber("./data/podcast_clip.mp3")
9 print(result)
```

输出结果：

 复制代码

```
1 {'text': '欢迎来到Onboard真实的一线经验走新的投资思考我是Monica我是高宁我们一起聊聊软件如何改
```

不过，即使转录成了中文，也会有一些小小的问题。你会看到转录后的内容没有标点符号。目前，Transformers 库的 Pipeline 还没有比较简单的方法给转录的内容加上 Prompt。这也是

Pipeline 的抽象封装带来的一个缺点。如果你有兴趣，也可以看看是否可以为 Transformers 库贡献代码，让它能够为 Whisper 模型支持 Prompt 的功能。

除了语音之外，Transformers 也支持图像类问题的处理。不过我们还没有讲到那一块，今天就先不介绍了。在课程后面的图像部分，我们再详细介绍。

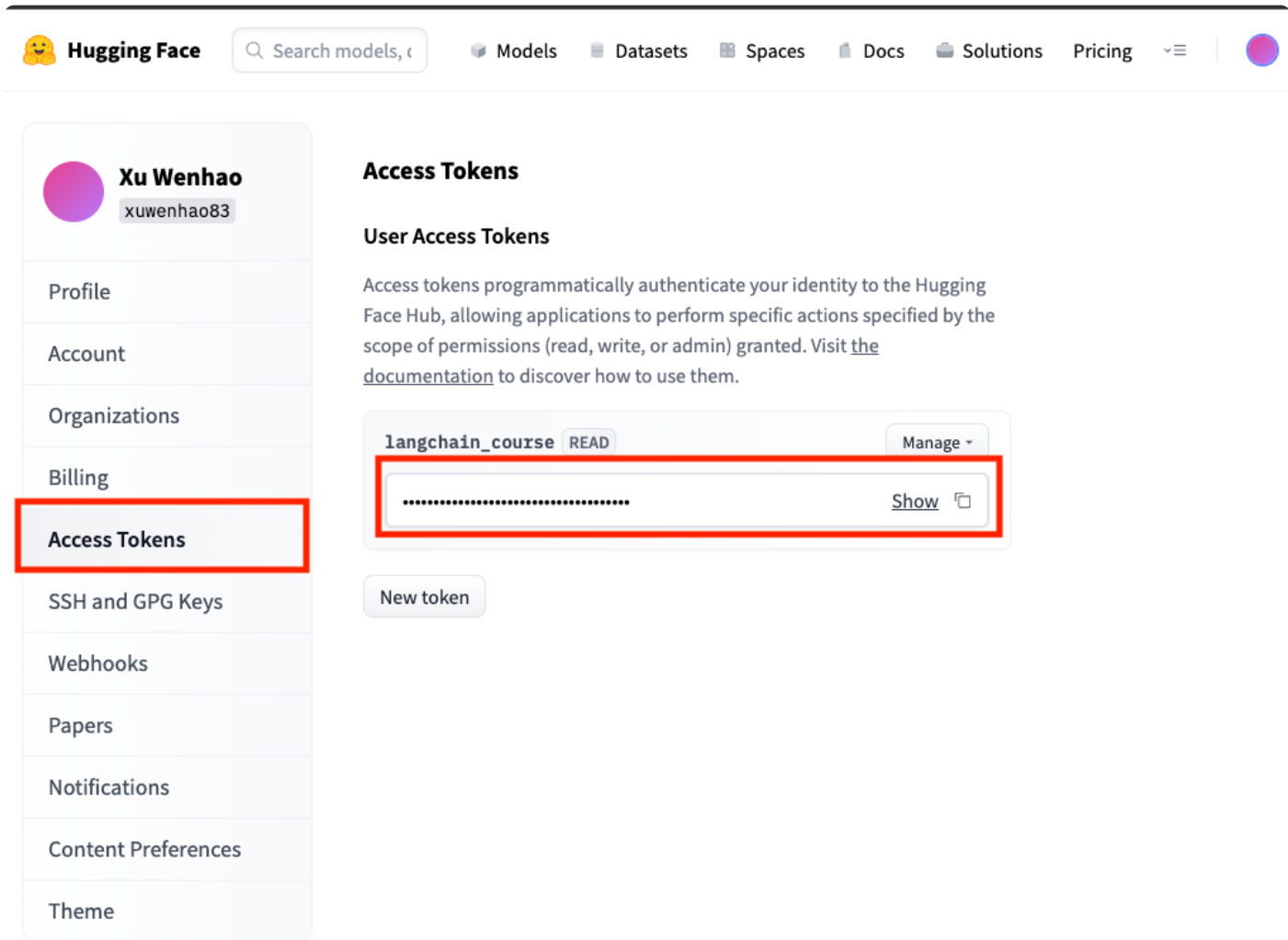
如何使用 Inference API?

如果你实际运行了上面我们使用的 Pipeline 代码，你就会发现其实大量的时间，都被浪费在下载模型的过程里了。而且，因为 Colab 的内存和显存大小的限制，我们还没办法运行尺寸太大的模型。比如，flan-t5-xxl 这样大尺寸的模型有 110 亿参数，Colab 和一般的游戏显卡根本放不下。

但是这些模型的效果往往又比单机能够加载的小模型要好很多。那么这个时候，如果你想测试体验一下效果，就可以试试 Inference API。它是 HuggingFace 免费提供的，让你可以通过 API 调用的方式先试用这些模型。

尝试 Inference API

首先，和其他的 API Key 一样，我们还是通过环境变量来设置一下 Huggingface 的 Access Token。你可以在 Huggingface 的 [🔗个人设置](#) 里面拿到这个 Key，然后通过 export 设置到环境变量里就好了。



在你的个人Settings页面，点击左边的Access Tokens，然后在右侧点击Show拿到自己的Access Token

设置环境变量：

```
1 export HUGGINGFACE_API_KEY=YOUR_HUGGINGFACE_ACCESS_TOKEN
```

📋 复制代码

然后，我们就可以通过简单的 HTTP 请求，调用托管在 Huggingace 里的模型了。比如，我们可以通过下面的代码，直接用 flan-t5-xxl 这个模型来进行问答。


```
1 import os, requests, json
2
3 API_TOKEN = os.environ.get("HUGGINGFACE_API_KEY")
4
5 model = "google/flan-t5-xxl"
```

📋 复制代码

```
6 API_URL = f"https://api-inference.huggingface.co/models/{model}"
7 headers = {"Authorization": f"Bearer {API_TOKEN}", "Content-Type": "application/j
8
9 def query(payload, api_url=API_URL, headers=headers):
10     data = json.dumps(payload)
11     response = requests.request("POST", api_url, headers=headers, data=data)
12     return json.loads(response.content.decode("utf-8"))
13
14 question = "Please answer the following question. What is the capital of France?"
15 data = query({"inputs" : question})
16
17 print(data)
```

输出结果：

```
1 [{"generated_text": 'paris'}]
```

 复制代码


上面的演示代码也很简单，需要做到三点。

1. 我们向 HuggingFace 的 api-inference 这个域名发起一个请求，在对应的路径里跟上模型的名字。
2. 在请求头里，带上我们拿到的 ACCESS TOKEN，来通过权限的校验。
3. 通过一个以 inputs 为 key 的 JSON，作为请求体发送过去就好了。




运行这个例子你就可以看到，flan-t5-xxl 这样的模型也有一定的知识和问答能力。在这个例子里，它就准确地回答出了法国的首都是巴黎。








等待模型加载完毕






同样的，Inference API 也支持各种各样的任务。我们在模型页的卡片里，如果能够看到一个带着闪电标记⚡的 Hosted Inference API 字样，就代表着这个模型可以通过 Inference API 调用。并且下面可以让你测试的示例，就是这个 Inference API 支持的任务。

 **Hugging Face**

Models Datasets Spaces Docs Solutions Pricing

 **hfl/chinese-pert-base**   like 9

 Feature Extraction  PyTorch  TensorFlow  Transformers  Chinese  bert  License: cc-by-nc-sa-4.0


 Model card  Files  Train  Deploy  Use in Transformers



Please use 'Bert' related functions to load this model!


Under construction...


Please visit our GitHub repo for more information:
<https://github.com/ymcui/PERT>

Downloads last month
4,579





 **Hosted inference API** 

 Feature Extraction




Computation time on Intel Xeon 3rd Gen Scalable cpu: 1.156 s

	0	1	2	3	4	5	6
0	-0.048	0.017	0.007	0.066	0.048	0.034	0.044
1	-0.013	-0.047	-0.033	0.102	0.044	-0.006	0.057
2	-0.067	-0.051	-0.003	0.079	-0.018	-0.069	0.046
3	-0.052	0.028	0.014	0.086	0.022	0.042	0.049


 JSON Output  Maximize

比如上面截图里的 `hfl/chinese-pert-base` 模型支持的就是 `feature-extraction` 的任务，它能够让你把自己的文本变成向量。我们不妨来试一试。

 复制代码

```
1 model = "hfl/chinese-pert-base"
2 API_URL = f"https://api-inference.huggingface.co/models/{model}"
3
4 question = "今天天气真不错！"
5 data = query({"inputs": question}, api_url=API_URL)
6
7 print(data)
```

输出结果：

 复制代码

```
1 {'error': 'Model hfl/chinese-pert-base is currently loading', 'estimated_time': 2}
```


第一次尝试去调用这个 Inference API，我们得到了一个报错信息。这个消息说的是，模型还在加载，并且预计还需要 20 秒才会加载完。因为 Inference API 是 Huggingface 免费提供给大家的，它也没有足够的 GPU 资源把所有模型（约几万个）都随时加载到线上。所以实际上，很多模型在没有人调用的时候，就会把 GPU 资源释放出来。只有当我们调用的时候，它才会加载模型，运行对应的推理逻辑。

我们有两个选择，一个是等待一会儿，等模型加载完了再调用。或者，我们可以在调用的时候就直接加上一个参数 **wait_for_model=True**。这个参数，会让服务端等待模型加载完成之后，再把结果返回给我们，而不是立刻返回一个模型正在加载的报错信息。

 复制代码

```
1 data = query({"inputs" : question, "wait_for_model" : True}, api_url=API_URL)
2
3 print(data)
```

输出结果：

 复制代码

```
1 [[[-0.05410267040133476, -0.0140887051820755, 0.017411280423402786, 0.10337194055
```

我们在 Pipeline 里介绍的任务，基本都可以通过 Inference API 的方式来调用。如果你想深入了解每一个任务的 API 的参数，可以去看一下 HuggingFace 的 [🔗 官方文档](#)。

如何部署自己的大模型？

不过，Inference API 只能给你提供试用各个模型的接口。因为是免费的资源，自然不能无限使用，所以 HuggingFace 为它设置了限额（Rate Limit）。如果你觉得大模型真的好用，那么最好的办法，就是在云平台上找一些有 GPU 的机器，把自己需要的模型部署起来。

HuggingFace 自己就提供了一个非常简便的部署开源模型的产品，叫做 Inference Endpoint。你不需要自己去云平台申请服务器，搭建各种环境。只需要选择想要部署的模型、使用的服务器资源，一键就能把自己需要的模型部署到云平台上。

把模型部署到 Endpoint 上

其实 GPT2 的论文里，已经体现了大语言模型不少潜力了。那么，下面我们就试着来部署一下 GPT2 这个模型。

1. 首先，进入 [创建 Endpoint 的界面](#)，你可以选择自己想要部署的模型，我们这里选择了 GPT2 这个模型。
2. Endpoint Name，你可以自己设置一个，也可以直接使用系统自动生成的。
3. 系统默认会为你选择云服务商、对应的区域，以及需要的硬件资源。如果你选择的硬件资源不足以部署这个模型，页面上也会有对应的提示告诉你。GPT2 的模型连 GPU 也不需要，有 CPU 就能运行起来。
4. 最后你需要选择一下这个 Endpoint 的安全等级，一共有三种，分别是 Protected、Public 和 Private。

Public 是指这个模型部署好之后，互联网上的任何人都能调用，不需要做任何权限验证。一般情况下，你不太会选择这一个安全等级。

Protected，需要 HuggingFace 的 Access Token 的验证。我们在这里就选用这个方式，这也是测试使用最常用的方式。

Private，不仅需要权限验证，还需要通过一个 AWS 或者 Azure 云里面的私有网络才能访问。如果你实际部署一个应用在线上，对应 API 访问都是通过自己在云上的服务器进行的，那么选择这个方式是最合理的。

Create a new Endpoint

Easily deploy machine learning models for your applications

Model Repository

Import a model from the Hugging Face hub

gpt2



Endpoint Name

Input a name for your new endpoint

aws-gpt2-5509

Cloud Provider

You can choose between Amazon Web Services, Microsoft Azure and Google Cloud Platform.

[Contact us](#) if you need a custom solution.



N. Virginia
us-east-1



Ireland
eu-west-1



Frankfurt
eu-central-1



Oregon
us-west-2



Hong Kong
ap-east-1

Instance type

Select a CPU or GPU accelerated compute option for inference.

If the instance type cannot be selected, you need to [contact us](#) and request instance quota.

CPU [medium] · 2vCPU 4GB · Intel Ice Lake



► Advanced configuration [click to expand](#)

Endpoint security level

Choose your endpoint's level of privacy.



Protected



Public



Private

A Protected Endpoint is available from the Internet, secured with TLS/SSL and requires a valid [Hugging Face Token](#) for Authentication.

Create Endpoint

Create with cURL

Estimated cost:

\$87.61 / month

创建Endpoint的界面

设置好了之后，你再点击最下面的 **Create Endpoint**，HuggingFace 就会开始帮你创建机器资源，部署对应的模型了。

Your endpoint is starting. Once it's available, you'll be able to send requests using the URL provided.

🔔 Unsure about how to use your endpoint ? Check out the [documentation!](#)

Model Repository

gpt2 ↗

Endpoint Type

Protected

Instance Type (CPU)

CPU • medium

Task

text-generation

Revision

e7da7f221d5bf496a48136c0cd264e630fe9fcc8

Provider

AWS • us-east-1

创建Endpoint之后，需要等待HuggingFace为你把模型部署起来

我们只要等待几分钟，模型就能部署起来。当然，这是因为 GPT2 的模型比较小。如果你尝试部署一些大尺寸的模型，可能需要 1-2 个小时才能完成。因为 HuggingFace 要完成模型下载、Docker 镜像打包等一系列的工作，单个模型又很大，所以需要更长时间。


测试体验一下大模型

部署完成之后，我们会自动进入对应的 Endpoint 详情页里。上面的 Endpoint URL 就表示你可以像调用 Inference API 一样调用模型的 API_URL。而下面，也给出了一个测试输入框，这个测试输入框我们在 HuggingFace 模型卡片页面里也能够看到。


[←](#) **aws-gpt2-5509** Running Pause endpoint

[Overview](#) [Analytics](#) [Logs](#) [Settings](#)



Endpoint URL

https://abmlvcliaa98k9ct.us-east-1.aws.endpoints.huggingface.cloud 

📘 Unsure about how to use your endpoint ? Check out the [documentation!](#)

Model Repository	Task
gpt2 	text-generation
Endpoint Type	Revision
Protected	e7da7f221d5bf496a48136c0cd264e630fe9fcc8
Instance Type (CPU)	Provider
CPU • medium	AWS • us-east-1




Test your endpoint!

 Text Generation Examples 

My name is Lewis and I like to

Compute

This endpoint is running and ready for inference.

 Copy as cURL  Show JSON output Maximize 

我们可以用这样一段简单的代码来测试一下 GPT2 模型对应的效果。

 复制代码

```
1 API_URL = "https://abmlvcliaa98k9ct.us-east-1.aws.endpoints.huggingface.cloud"
2
3 text = "My name is Lewis and I like to"
4 data = query({"inputs" : text}, api_url=API_URL)
5
6 print(data)
```

输出结果：

复制代码

```
1 [{"generated_text": "My name is Lewis and I like to think I\'m a dog. It would me
```

有了这样一个部署在线上的模型，你就可以完全根据自己的需求随时调用 API 来完成自己的任务了，唯一的限制就是你使用的硬件资源有多少。

暂停、恢复以及删除 Endpoint

部署在 Endpoint 上的模型是按照在线的时长收费的。如果你暂时不用这个模型，可以选择**暂停** (Pause) 这个 Endpoint。等到想使用的时候，再重新**恢复** (Resume) 这个 Endpoint 就好了。暂停期间的模型不会计费，这个功能的选项就在模型 Overview 标签页的右上角。

点击Pause，Endpoint就会暂停，HuggingFace也就不会再向你收费

如果你彻底不需要使用这个模型了，你可以把对应的 Endpoint 删掉，你只需要在对应 Endpoint 的 Setting 页面里输入 Endpoint 的名称，然后选择删除就好了。

←

aws-gpt2-5509

Running

Pause endpoint

Overview

Analytics

Logs

Settings

Configuration

Instance Type

CPU [medium] · 2vCPU 4GB · Intel Ice Lake

▼

Replica autoscaling

Min

1

Max

1

Task

Text Generation

▼

Revision

e7da7f221d5bf496a48136c0cd264e630fe9fcc8

Update endpoint

Delete endpoint

This will permanently delete **aws-gpt2-5509** including all its deployments and logs.

Enter **aws-gpt2-5509** below to confirm.

Delete this endpoint

可以在Settings页面里面删除模型

HuggingFace 将部署一个开源模型到线上的成本基本降低到了 0。不过，目前它只支持海外的 AWS、Azure 以及 Google Cloud，并不支持阿里云或者腾讯云，对国内的用户算是一个小小的遗憾。

小结

好了，这一讲到这里也就结束了。

今天，我带着你了解了如何利用 HuggingFace 以及开源模型，来实现各类大模型应用的推理任务。最简单的方式，是使用 Transformers 这个 Python 开源库里面的 Pipeline 模块，只需要指定 Pipeline 里的 model 和 task，然后直接调用它们来处理我们给到的数据，就能拿到

结果。我们不需要关心模型背后的结构、分词器，以及数据的处理方式，也能快速上手使用这些开源模型。Pipeline 的任务，涵盖了常见的自然语言处理任务，同时也包括了音频和图像的功能。

而如果模型比较大，单个的 GPU 不足以加载这个模型，你可以尝试通过 HuggingFace 免费提供的 Inference API 来试用模型。只需要一个简单的 HTTP 请求，你就可以直接测试像 `flan-t5-xxl` 这样 110 亿参数的大模型。而如果你想要把这样的大模型应用到你的生产环境里，你就可以通过 Inference Endpoint 这个功能来把大模型部署到云端。当然，这需要花不少钱。

在了解了 Pipeline、Inference API 以及 Inference Endpoint 之后，相信你已经充分掌握利用 Huggingface 来完成各种常见的文本、音频任务的方法了。后面需要的就是多多实践。

思考题

最后，我给你留一道思考题。

你能试着使用一些 HuggingFace 的 `feature-extraction` 任务，通过开源大模型来做一下情感分析吗？你可以拿一些数据，看看 `flan-t5-xxl` 这样的模型的效果怎么样。

欢迎你把你实践后的结果分享出来，我们一起学习，共同进步，你也可以把这一讲分享给你身边感兴趣的朋友，邀他一起学习。我们下一讲再见！

推荐阅读

HuggingFace 的 [🔗 官方文档](#)里，给出了通过 Pipeline 完成各种任务的详细示例。你可以对照着自己的需求看一下这个文档，相信能解决你 90% 以上的问题。

© 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。



Oli张帆

2023-04-26 来自北京

HuggingFace是个好东西!



1



Toni

2023-05-06 来自瑞士

从下面这段代码看在 Endpoint 上部署自己需用的模型后, 得到一个"个人"的 API_URL 接口, 每次任务还是从用户端发起请求, 结果再从云端返回。单次请求任务也就完成了, 但如果涉及大量的运算, 一来一往会消耗大量时间在"路"上。可以将数据打包放在离"计算中心"近处, 完成计算后再一次性将结果打包返回吗? 还是有其它的解决方法? 请老师指点迷津。

```
API_URL = "https://abmlvcliaa98k9ct.us-east-1.aws.endpoints.huggingface.cloud"
```

```
text = "My name is Lewis and I like to"
```

```
data = query({"inputs" : text}, api_url=API_URL)
```

```
print(data)
```

作者回复: 还是到云端。

本地那就只有自己的GPU服务器搞私有化部署了, 如果真的要重度使用, 就需要用私有化部署了。



芋头

2023-05-04 来自北京

想复现graph-gpt <https://graphgpt.vercel.app/>, 即用纯文本, 通过模型生成知识图谱。想问问大大能教一下吗

作者回复: GraphGPT不是开源的么? 直接看源码就好了呀, 好像也是调用OpenAI的API吧

<https://github.com/varunshenoy/GraphGPT>



Meadery

2023-05-02 来自广东

DLL load failed while importing _imaging: 找不到指定的模块。是什么问题啊

作者回复: 重新安装一下 pillow?

pip install pillow



小神david

2023-05-01 来自北京

希望huggingface越办越好

