**LLM模型部署指南**

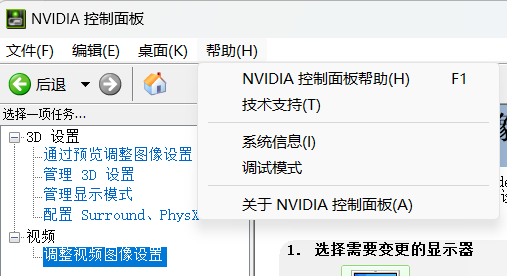
作者：杨宇琨

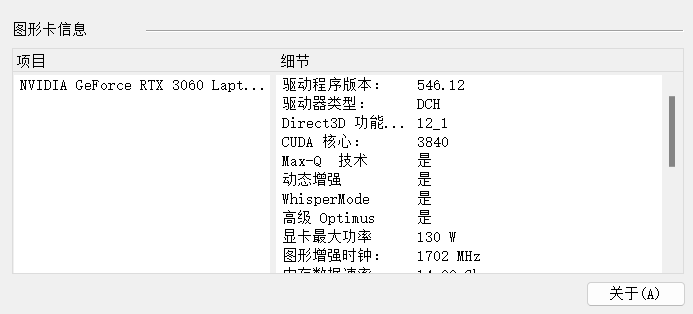
更新时间：2023/12/5

**硬件配置**

**笔记本—3060（8G显存）显卡及以上**

****

****

****

**台式机—2060（6G显存）显卡及以上**

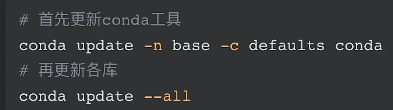
一般的游戏本都基本够用，台式机都基本够用

**环境配置**

**Python（3.8.xx版本）（必装）**

如果没有安装过Python考虑通过Anaconda安装python，Anaconda不仅仅是一个Python发行版，它还提供了一个包管理器，可以方便地安装、管理和更新各种数据科学和机器学习所需的库。

如果已经安装过Python记得更新conda库



**Jupyter编辑器（选装）**

Jupyter是一个基于Web的交互式计算环境，用于创建Jupyter Notebook文档。Jupyter Notebook文档是一个JSON文档，包含一个有序的输入/输出单元格列表，这些单元格可以包含代码、文本（使用Markdown语言）、数学、图表和富媒体（Rich media），通常以".ipynb"文件格式保存。

**Jupyter网页链接：**[**Project Jupyter | Home**](https://jupyter.org/)

**VS code（必装）**

VS Code具有强大的扩展性，用户可以根据自己的需要下载并安装各种扩展，例如针对不同编程语言的扩展、代码片段扩展、工具扩展等。这些扩展使得VS Code可以适应各种开发需求，从而提高开发效率。

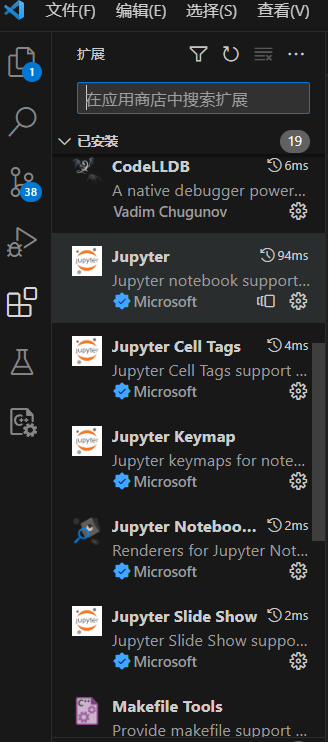
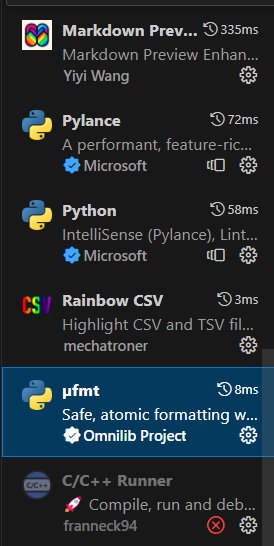
此外，VS Code还支持Jupyter Notebook，这意味着用户可以在VS Code中直接运行和调试Jupyter Notebook，从而实现代码编写、调试和可视化的一体化。

本文档之后的操作全部基于VS code中的Jupyter插件而非Jupyter官方编辑器

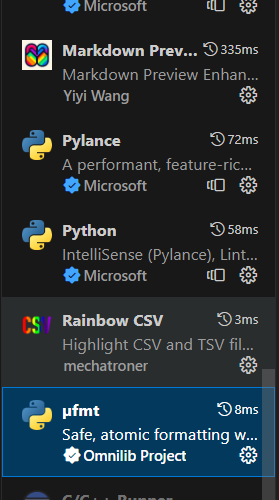
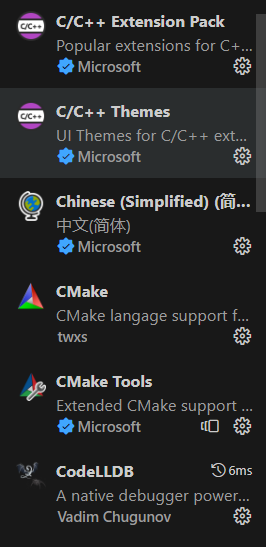
**VS code网站链接：**[**Visual Studio Code - Code Editing. Redefined**](https://code.visualstudio.com/)

**插件安装：**

**Jupyter：**  **Python：**

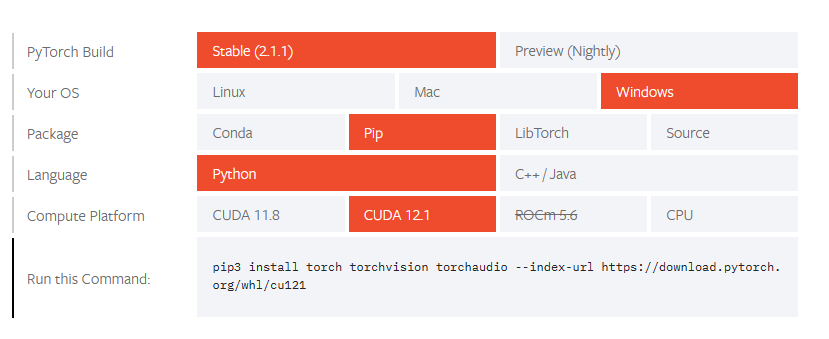
**汉化，彩虹标亮，Markdown预览（推荐安装）：**



**Pytorch（必装）**

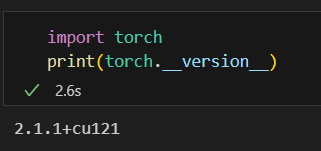
如果没有装过Pytorch，直接去官网下载

**网站链接：**[**Start Locally | PyTorch**](https://pytorch.org/get-started/locally/)

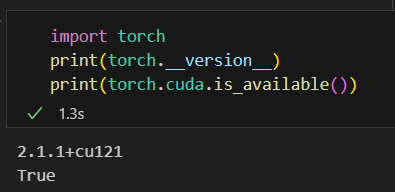
****

运行命令即可安装（装CUDA 12.1适配CUDA平台）

如果已经安装过Pytorch，创建一个文件来检查自己的版本是否符合



如果符合再检查cuda是否可用



如果以上两步有错误，更新Pytorch

现在目前最好的Pytorch更新方法是直接删了重装（我服

删除命令：

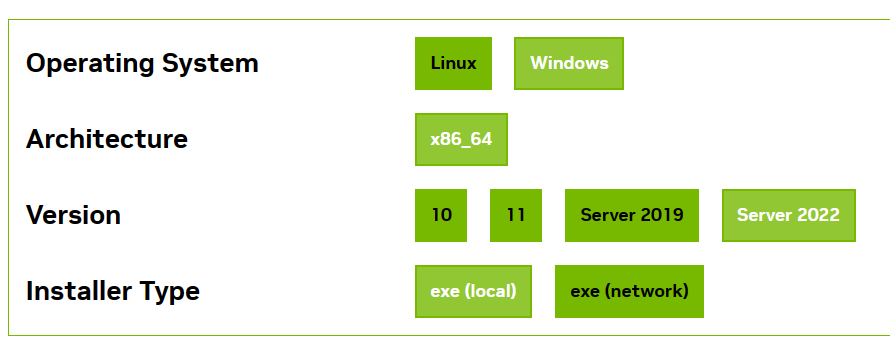
重装命令（其实就是上面安装命令）：



**CUDA**

直接上链接！！

[CUDA Toolkit 12.3 Update 1 Downloads | NVIDIA Developer](https://developer.nvidia.com/cuda-downloads)



选成我这样就行

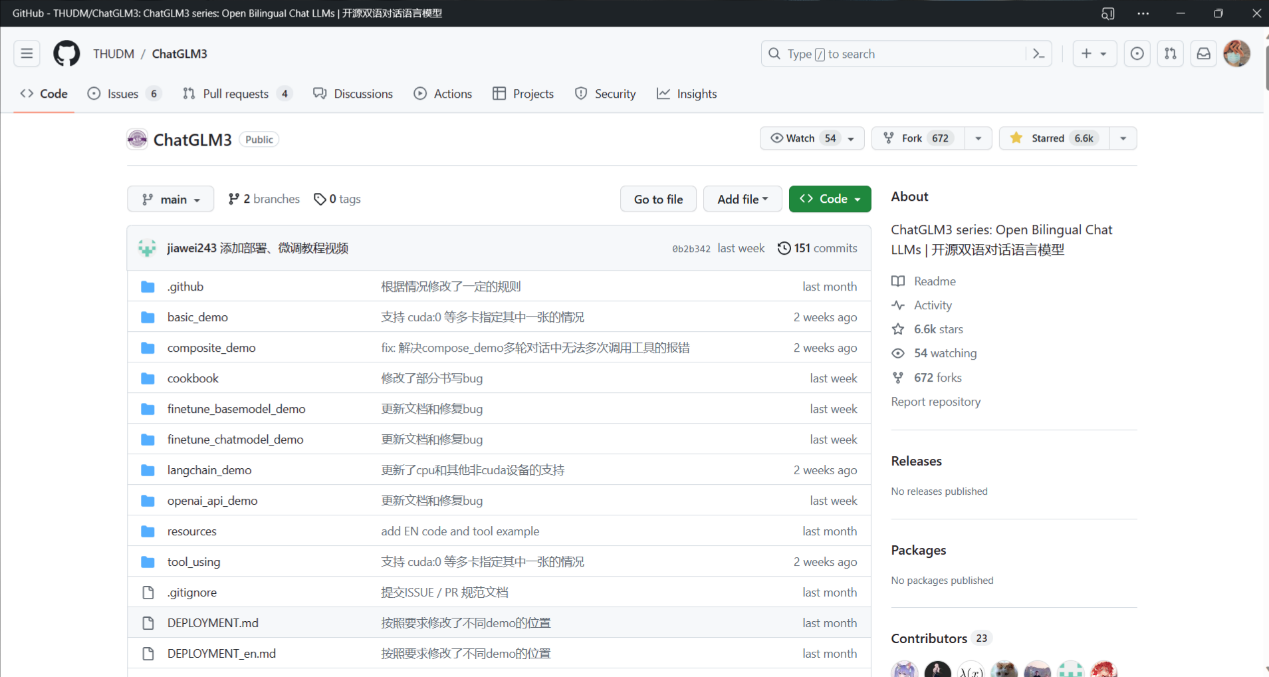
如果已经下过cuda，更新一下就行，相信大家都是没下过cuda的所以这里就不多加介绍了，放一个参考链接（没试过不保真）

[windows cuda更新教程\_cuda 12.0-CSDN博客](https://blog.csdn.net/weixin_51066144/article/details/128105318)

**还有一个很重要的东西就是Git，重要得有点不需要讲了，应该大家都装了，没装的话在网上找找方法**

**OK下面进入LLM部署正文！**

我们部署的模型是ChatGLM3，它是智谱AI和清华大学 KEG 实验室联合发布的新一代对话预训练模型。



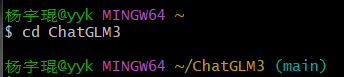
我们使用git命令克隆这个仓库到我们本地



$ git clone <https://github.com/THUDM/ChatGLM3>

$ cd ChatGLM3

下载完成后我们可以在c:/用户/（你的名字）目录下可以找打ChatGLM3这个文件夹，这样我们就克隆成功了



（可以看到我们已经进入到这个文件夹中了）

输入命令安装运行依赖：$ pip install -r requirements.txt



（这一步有可能报错比如segmentation fault，解决方法是把命令改为$ pip3 install -r requirements.txt）

**依赖安装好之后就可以运行模型了（试试以下代码）**

from transformers import AutoTokenizer, AutoModel

tokenizer = AutoTokenizer.from\_pretrained("THUDM/chatglm3-6b",

trust\_remote\_code=True)

model = AutoModel.from\_pretrained("THUDM/chatglm3-6b", trust\_remote\_code=True, device='cuda')

model = model.eval()

response, history = model.chat(tokenizer, "你好", history=[])

print(response)

response, history = model.chat(tokenizer, "晚上睡不着应该怎么办", history=history)

print(response)

这一步时间有点长

**如果说运行后发现输出不正常，可能是网络问题，因为模型加载需要跑到Huggingface平台，而访问这个平台需要魔法，对于没有魔法的同学我们可以采用本地下载的方式**

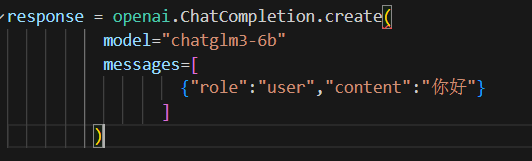
进入网站: [chatglm3-6b · 模型库 (modelscope.cn)](https://modelscope.cn/models/ZhipuAI/chatglm3-6b/summary)

里面有详细的命令教程，一步一步安装（需要花点时间）后就能完成本地模型的部署。

**Api调用**

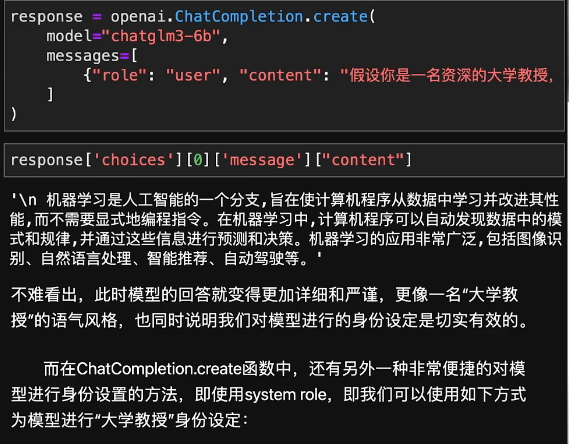
我们使用openai的调用方法，首先安装openai库

命令：pip install openai



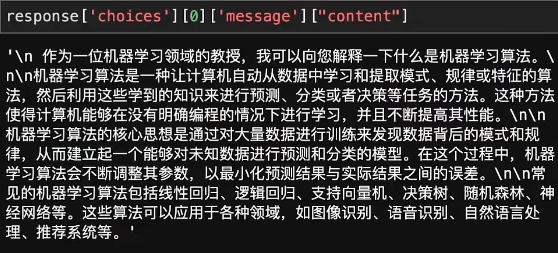
**Messages很重要！！！它包含了当前模型输入输出的信息的特征，有信息的角色来源，信息内容等！**

在这里，它是外部给大语言模型的信息

****

User直接通过对话告诉大语言模型信息，包括大预言模型自身需要假扮的角色，而在openai.ChatCompletion.create函数中可以使用systemrole方式设置大模型角色

****



效果更好

response['choices'][0]['message']["content"]

['choices']:大模型会给出几个回答

[0]:选择第一个回答，‘0’是一个index

['message']:message是一个对象{‘role’，‘content’，‘metadata’，‘tool’}role是角色，content是内容，metadata是元数据信息，tool是模型使用到的工具（比如function calling）

["content"]:大模型回复的内容

**Function Calling**

**Agent=LLM+Memory+Tool**

LLM：大模型 如chatgpt,chatglm3

Memory：本身的知识库，与外界互动后的记忆（history）

**Tool：重点，使大模型从一个泛化模型变成专业教授的关键部分，它可以是外部工具比如天气查询，Google爬虫，也可以是你自己写的一个库函数，比如算术系统，使得模型获得强大的数字推理能力**

一般function calling都会调用API 所以要先加载相关参数，这些参数的设置都会在使用的API网站上有说明

# 加载必要的参数

load\_dotenv()

serper\_api\_key=os.getenv("SERPER\_API\_KEY")

browserless\_api\_key=os.getenv("BROWSERLESS\_API\_KEY")

openai\_api\_key=os.getenv("OPENAI\_API\_KEY")

初始化一个agent

#初始化 agent

agent = initialize\_agent(

    tools, #配置工具集

    llm,   #配置大语言模型 负责决策

    agent=AgentType.OPENAI\_FUNCTIONS, #设置 agent 类型 https://python.langchain.com/docs/modules/agents/agent\_types/openai\_functions\_agent

    verbose=True,

    agent\_kwargs=agent\_kwargs, #设定 agent 角色

    memory=memory, #配置记忆模式

)

给出工具集

#初始化 agent 可使用的工具集合

tools = [

    Tool(

        name="Search",

        func=search,

        description="useful for when you need to answer questions about current events, data. You should ask targeted questions"

    ),

    ScrapeWebsiteTool(),

]

ScrapeWebsiteTool()类定义

class ScrapeWebsiteInput(BaseModel):

    """Inputs for scrape\_website"""

    target: str = Field(

        description="The objective & task that users give to the agent")

    url: str = Field(description="The url of the website to be scraped")

class ScrapeWebsiteTool(BaseTool):

    name = "scrape\_website"

    description = "useful when you need to get data from a website url, passing both url and objective to the function; DO NOT make up any url, the url should only be from the search results"

    args\_schema: Type[BaseModel] = ScrapeWebsiteInput

    def \_run(self, target: str, url: str):

        return scrape\_website(target, url)

ScrapeWebsiteInput类：这是一个BaseModel的子类，用于定义输入参数。它有两个字段：target和url。其中，target字段描述了用户给代理的任务目标，url字段描述了要抓取的网站的URL。

ScrapeWebsiteTool类：这是一个BaseTool的子类，用于定义工具的名称、描述和参数。该类的名称和描述分别为"scrape\_website"和"useful when you need to get data from a website url, passing both url and objective to the function; DO NOT make up any url, the url should only be from the search results"，即抓取网站数据时，需要传入网站URL和目标任务。该类的参数解析模型为ScrapeWebsiteInput。

\_run方法：这是一个私有方法，用于执行抓取网站的操作。该方法接收两个参数：target和url，然后调用scrape\_website函数进行数据抓取，并返回抓取结果。

定义Google search函数

#调用 Google search by Serper

def search(query):

    serper\_google\_url = os.getenv("SERPER\_GOOGLE\_URL")

    print(f"Serper Google Search URL: {serper\_google\_url}")

    payload = json.dumps({

        "q": query

    })

    headers = {

        'X-API-KEY': serper\_api\_key,

        'Content-Type': 'application/json'

    }

    response = requests.request("POST", serper\_google\_url, headers=headers, data=payload)

    print(f'Google 搜索结果: \n {response.text}')

    return response.text

函数接收一个参数query，这个参数是你想要搜索的关键词。

函数首先从环境变量SERPER\_GOOGLE\_URL中获取Google搜索API的URL，并打印出来。

然后，函数构建了一个JSON格式的payload，其中包含搜索查询关键词。

接下来，函数设置了HTTP请求的头部信息，包括API密钥和内容类型。

使用requests库向Google搜索API发送POST请求，并将payload作为请求体发送。

函数接收响应，并打印出Google搜索结果。

最后，函数返回搜索结果的文本内容。

以上是一个加入了google爬虫的Agent ，具体函数实现逻辑涉及到文本分隔，文本总结等内容，这里不过多阐述。

**Github链接：**[**https://github.com/parallel75/AI\_Agent.git**](https://github.com/parallel75/AI_Agent.git)