**GraphRAG实战**

**用LangChain+Neo4j开发知识图谱增强的RAG智能体应用**

**叶健峰 著**

**前言**

自从1950年艾伦·图灵提出人工智能的图灵测试，1956年麦卡锡在达特茅斯学院举办的历史上第一次人工智能研讨会上提出人工智能（AI, Artificial Intelligence）的概念以来，AI经历了漫长的几个发展阶段，有早期60、70年代符号主义与专家系统的探索，也有80年代至今由机器学习与深度学习推动的复兴，更有国际象棋上深蓝击败卡斯帕罗夫与围棋上Alpha Go击败李世石的突破，有繁荣也有寒冬，几起几落，然而始终未能走进千行百业，更未能走进千家万户，影响每个人的生活。

就像个人电脑与互联网走进千家万户，智能手机与移动互联网便利每个人的历程一样，AI的发展与普及，同样需要时与势的积累，水到渠成。早期的探索明确了AI发展的技术路径，个人电脑、互联网、智能手机、移动互联网的普及与各行各业的深度信息化则积累了海量的数据，在深度学习算法与GPU算力的冶炼下，终于迎来了大力出奇迹的质变时刻。2022年11月，ChatGPT横空出世，在全球掀起了新一轮AI的爆发式发展，各种大模型和落地应用如雨后春笋般不断涌现，迅速渗透并赋能各行各业。经过两年多的深入了解和使用体验，我相信这一次AI可以走出象牙塔，真正走进每个人的生活了。就像蒸汽机的工业革命解放了生产的人力，电力的二次工业革命解放了生产的能源，计算机与互联网的三次工业革命（信息技术革命）改变了生产的生产资料（数据成为了一种重要的生产要素）与组织方式一样，AI的普及将会解放人这个生产者本身，从而推动人类社会的生产力与生产关系，以及其上的经济基础与上层建筑的巨大变革。有些变化是人们现在就已经可以看得到的，比如工作任务的转移与工作方式的转变，创新发明方式的转变，分配关系的变化，媒体等公共话语权的变化，法律伦理规范的挑战等等。物理、化学、生理或医学等几个诺贝尔奖2024年授予了AI辅助研究的几个科学家，这足以说明AI对当今时代之变的巨大影响。以史为鉴，人类正在进入由AI引发新一轮工业革命的时代，正在身处百年未有之大变局。

国内的应变可谓迅速，从2023年下半年开始就陆续有各种大小模型发布测试，目前在工信部备案通过的已经有近二百个。虽然面对算力供应的封杀，但国内凭借庞大的经济体量、用户数量、数据体量，完整的工业门类和丰富的落地应用场景，在AI赋能千行百业，AI走进每个人生活的应用赛道上，大有后来居上之势，应该也能够后来居上，DeepSeek R1/V3的爆发就是一个很好的证明。反过来，应用的探索也可以极大地促进研发的进展，促进软硬件技术的迅速进步。2024年的世界互联网大会上，业界大佬们的共识是，大模型的下半场，卷的是应用。国内庞大的经济、数据、用户体量和丰富的应用场景足以让各种大模型八仙过海，各显神通，开展良性竞争，促进产业进步，增进人们福祉，正如宽阔的太平洋能容得下中美两个大国一样。

产业革命的价值在于大幅提高劳动生产率，催生新的需求新的产出以提高人们的生活质量，从而更快更好地创造新价值。AI要在各行各业中落地应用，赋能传统产业，首先要解决准确、高效地融入与改造传统产业的问题。大模型的性质，是一个在海量语料数据上训练的概率论统计学模型，在庞大的深度学习神经网络参数中压缩存储了从训练语料中获得的知识和模式，它工作的原理是根据输入的提示词不断地输出并积累下一个最可能的词。概率论统计学推理的性质决定了，它并不能和人类智能一样具有确定性的逻辑推理，虽然现在的算法水平已经可以让大模型大概率的输出合乎常识与逻辑的答复，但还是有一定的概率会一本正经的胡说八道，这是这个技术路线固有的缺憾。由此带来的第二个缺憾是，它不知道训练语料中没有的知识，即知识的缺失。大模型的训练语料大多是通过网上公开的渠道搜集，各种开源或授权的数据，所以训练出来的是通识型的通用模型。现实中由于知识产权、数据保密等原因，大部分组织的数据是私有的，并没有开源，所以通用模型在行业落地应用场景中的效果一般不会太好。第三个缺憾是知识截止，因为训练语料都有其搜集的截止日期，大模型无法回答截止日期以后训练语料中没有的问题，这对于一些有时效要求的应用场景比如新闻媒体、股票交易等是不能接受的。

这两年来业界为克服大模型上述的三个主要缺憾开发出了不同的方法，主要有训练行业专用模型、微调与RAG。训练行业专用模型需要收集和整理庞大的行业专有语料库，一般由有技术与算力资源的大厂与有庞大行业语料资源积累的龙头组织合作才能完成。现代工业分工比较细，任何一方都很难同时具备这两种资源，所以这是技术与资源的强强联合，投资大成本高周期长，普适性还不够强。然后大模型厂商们推出了微调的方案，让用户整理准备具体落地应用场景的语料库，在其服务平台通用模型的基础上自己进行微调训练。这样对技术的要求就降低了，但训练语料的收集、整理与标注仍然是一项成本高周期长的工作，中小型组织如果要实施，就要好好评估自身的能力与期望的成本收益。这两个方案组织私有的行业数据都要与大模型厂商进行分享，因此除了成本问题还涉及到数据保密与知识产权的问题，虽然有商业协议可以约束，但在保密部门或涉及商业机密的场景等就需要多加斟酌。当然也有美国的情报部门委托微软公司训练部署情报系统专用大模型这样的例子，这个级别的合作就要另当别论了。因此RAG（Retrieval Augmented Generation，检索增强生成）方案就应运而生并在2024年迅速成为了业界共同的选择，因为外挂的知识库部署在组织的本地，仅在推理时使用，不需要在训练时分享，较好保护了其数据的私密性与知识产权；也比较容易与组织的业务系统集成，赋能传统业务；实施起来又相对容易，较好地平衡了成本和收益。对于完全私密的情况，还可以本地部署端侧模型完全离线运行。

在生成质量上，RAG是通过在组织的私有知识库中检索出相对准确的相关语料，注入提示词中提交大模型去总结生成与问题比较匹配的答案，比起训练专用行业模型与微调来说，克服大模型的幻觉、知识缺失和知识截止固有缺憾的效果更好。所以RAG方案特别适合于金融、法律、教育、医疗等数据敏感、私密性强以及准确性要求高的行业落地应用场景。

普通RAG主要是通过向量嵌入与相似性比较来检索与问题直接相关的语料，它可以准确定位语义相关的文本块，但仍然有它的局限性，比如它不擅长回答答案分布在多个相关文档中或分布在文档分散段落中的问题；需要在多个原始材料中进行深度推理的问题；以及全局性的问题。业界为提高检索语料与用户问题的相关性进行了各种探索，GraphRAG就是其中效果比较好的一个解决方案：利用行业私有知识库中的结构化与非结构化文档建立反映行业知识与业务逻辑的全局知识图谱，然后通过向量检索定位知识图谱中的锚点，提取知识图谱中与用户问题高度相关的子图，组织所有与用户问题高度相关的材料，然后一起提交大模型总结回答。所以它可以有效克服普通RAG局部匹配的局限性。

知识图谱是反映现实世界中实体之间的关系及其运作的逻辑与流程最直接有效的模型，最早由谷歌公司提出并应用于其搜索引擎，有效改善了搜索的结果。作为它的算法基础，图数据科学从图论（离散数学）开始经过几十年的发展，已经非常成熟，产生了各种方便强大的图算法库与图数据库，非常擅长解决现实世界中实体关系分析的问题，所以GraphRAG是深度学习AI大模型与图数据科学学科交叉的结果，能够爆发出1+1远大于2的能量。

微软的GraphRAG论文在2024年7月一经发布，就在业界产生了巨大的影响，GraphRAG马上成为了当前RAG应用研究的热点，大家纷纷跟进，包括图数据库业界的翘楚Neo4j，它定义了图数据库事实上的工业标准。上述两大巨头的GraphRAG解决方案着重于借助大模型提高从非结构化文本文档中构建知识图谱的效率与质量，以及借助知识图谱提高大模型回答各类问题的质量。事实上，实际案例的测试表明从结构化文档中构建知识图谱的质量非常高，回答各类问题的效果也非常好，这是GraphRAG在非结构化文本外的另一种优势场景。

本书深入研究了业界领先的微软与Neo4j GraphRAG解决方案，结合国内国产大模型的应用条件，探索尝试了GraphRAG在国内各行业具体应用场景中落地实施的开源集成解决方案，具有较好的可操作性和较高的性价比，尤其适用于资源有限的各种中小型组织，快速地开发、测试、集成与部署GraphRAG应用，用大模型AI为传统业务赋能，助力组织在AI时代掌握先机，敏捷应变，准确应变，有效重组资源配置，重塑业务与流程，融入AI新时代，建立竞争与发展的新优势。本书也适用于对GraphRAG技术感兴趣的科教、技术人员和学生等个人，可以提供一个GraphRAG技术现状的概览和快速入门。

全书共分为9章，以开源的网络小说《悟空传》前7章为例，用大模型从中抽取实体与关系建立知识图谱，然后利用知识图谱回答与之相关的全局性问题或局部性问题，演示GraphRAG解决方案的效果与具体的实施方案。全书的代码均用Python开发。

第1章介绍了书中用到的深度学习开发环境的安装配置，包括在带GPU的个人电脑Windows+WSL2或云端虚拟主机上安装配置Ubuntu 22/24系统，安装配置书中用到的PyTorch、Jupyter Hub、HanLP、Neo4j图数据库软件。Rstudio Server、Shiny Server、Nginx及Shiny for Python的安装配置则放在第8章开发APP UI时介绍，有需要的读者可以先阅读并提前安装。

第2章介绍了微软GraphRAG的解决方案。安装跑起，用《悟空传》为例演示从非结构化的文本文档建立知识图谱，执行全局查询、局部查询与drift查询的效果，以及在本地Ollama上用端侧小参数模型跑起的效果。

第3章介绍了Neo4j GraphRAG的解决方案，用Docker跑起Neo4j KGBuilder，同样用《悟空传》为例演示从非结构化的文本文档建立知识图谱，执行全局查询与局部查询的效果，并介绍了在自己的Python应用中集成Neo4j KGBuilder的方法。

第4章介绍了融合微软与Neo4j GraphRAG各自的优点，开发自己的GraphRAG应用的开源解决方案，包括了实体关系提取，实体合并、嵌入与社区建立的实体工程，全局查询与局部查询的实现，同样以《悟空传》为例。用主流的开源大模型应用集成框架LangChain开发，用Neo4j图数据库存储和操作建立的知识图谱。Neo4j图数据库与向量数据库二合一比较方便，文本嵌入的向量存储为图结点的属性并可通过向量索引搜索匹配，Neo4j生态对此提供了很好的支持。

第5章介绍了智能体（Agent）的开发，用Agent贯通连接全局查询、局部查询和普通的问答，以支持多轮上下文相关的连续对话。用的是与LangChain配套的LangGraph Agent开发框架。Agent是当前大模型AI落地应用的主流技术方向，由大模型的函数调用功能进化而来，能够充分利用大模型的能力自主规划与决策，并调用外部工具执行特定的业务功能操作，有效连接落地应用场景的传统业务系统与AI大模型的能力，为传统业务赋能。

第6章测试了DeepSeek等目前已经接入LangChain的几个主流的国产大模型在GraphRAG中使用的效果。LangChain的优势之一是它的标准化和开放性，所有兼容OpenAI API的大模型都可以无缝接入。目前国内外主流的大模型都已接入LangChain，共有一百多个，其中国产的也有几十个。在国内落地大模型AI应用，国产大模型大概是硬性的要求，LangChain对此提供了很好的支持，既可以在研发时使用国外顶尖的大模型进行比较，也可以在部署时选择合适的国产大模型，统一的API让用户选择的范围比较广，切换非常方便。

第7章尝试了在本地部署的端侧小参数模型上测试GraphRAG的效果，包括了用Ollama跑起Qwen2.5，用vLLM跑起MiniCPM3，并探讨了在Llama.cpp及Huggingface上部署的可行性，适用于个人开发、测试、演示等场景。

第8章提供了GraphRAG与存量业务系统集成、赋能传统业务的高性价比解决方案，将智能体Agent通过FastAPI输出为RESTful API，便于为存量系统嵌入集成大模型的AI能力，有效盘活存量的软硬件与数据资产，助力组织优化资源配置，重塑业务与流程，提升AI时代的竞争力。同时也介绍了用Shiny for Python开发GraphRAG APP，完成应用交付用户的最后一步。

第9章介绍了GraphRAG应用生成质量评估的量化方法。非结构化文本质量评估是个非常有挑战性的问题，本章介绍的Ragas充分利用了大模型的文本处理能力，为生成的文本设计了一系列的定性判断问题，大模型对其作出定性判断后，统计这些定性判断就可以生成可信的量化评估指标。该解决方案提供了一条自动化评估和迭代改进GraphRAG应用的高性价比快速路径。

本书是作者GraphRAG系列专题研究文章的梳理和总结。这个课题及其它相关的研究从2024年4月下旬开始，到7月微软发布GraphRAG论文并开源其GraphRAG项目后全力投入，到10月下旬初步完成，元旦前后完成了书稿的整理，历时大半年，也算及时跟上了该领域应用研究当前的水平。当时DeepSeek尚未发布，所以先用OpenAI跑通，再适配国产大模型。国内GraphRAG方面的研究与应用正在蓬勃开展，但适配国产大模型应用的专著还有所欠缺，本书可以弥补这个空白，为有兴趣有需要的人们提供一些有用的参考。因为涉及大模型AI与图数据科学，国内具备这种交叉知识体系的人可能还不是很多，作者恰好就具备了，所以当微软发表了他们的论文后，决定予以一试。最后能够顺利走通，得益于长期知识积累的基础，也有赖于探索冒险的幸运。由于作者的能力水平所限，本书并没有作学术理论上的探讨，主要是具体应用中工程实践上的探索，所以定名为《GraphRAG实战》，这是要请读者诸君明鉴和包涵的。

作者是资深的IT从业人士，曾长期工作在信息系统开发运维的一线，主持开发过一些大型的互联网业务系统。近年来主要学习研究大数据分析与机器学习，曾翻译出版了R语言生态的[《精通Shiny》](http://www.oreilly.com.cn/index.php?func=book&isbn=978-7-5766-0656-0)一书（东南大学出版社，2023年7月，ISBN: 9787576606560）。ChatGPT发布后，两年多来持续跟踪学习大模型AI领域国内外的最新进展，对国内外主流的大模型及端侧的小参数模型都进行了全面和深入的测试，并开发了调用其API以及函数调用功能的落地应用实例APP，对大模型落地应用开发的技术栈有深入的了解，并积累了一定的经验。全书的开源集成解决方案，正是作者多年IT经验与知识厚积薄发的结果。

纵观历次工业革命的历程，革命性的变革会对每个人的学习、工作和生活都带来巨大的变化，会深刻调整社会乃至世界的利益关系与利益格局，因此不可避免会产生利益分配的摩擦与矛盾，阻力、反对甚至是对抗必然会伴随始终，冲突尖锐时甚至会引发社会的动荡与战争，比如英国工业革命的圈地运动，以及主要由利益冲突引发的世界大战。工业革命的燎原之火难免会伤及无辜，如果不在道德上执着于历史的正当性和合理性，只看历史的规律和结果，虽然每次工业革命都有社会群体作出了牺牲，但社会的文明确实是大步向前了，人们的生活确实是普遍改善了。按照中国人自古薪火相传的智慧与哲学，君子审时度势，趋吉避凶，看清历史与社会的大势，顺应时代的潮流，积极拥抱变化，才是在当下百年未有之大变局中正确的应变之道。所谓“大人虎变、君子豹变、小人革面”，这在DeepSeek冲击波中得到了充分的体现。当然，在讲求和谐的社会主义社会，科技以人为本，科技向善为要，AI工业革命的进程，要协调好所有相关主体的利益，努力增进所有人的福祉，这大概是政治的艺术。

大模型AI领域的技术进展日新月异，本书只是作者在学习体验过程中的若干思考和体会，一些内容也许很快就会过时，但总体的解决方案框架则相对稳定，希望读者们阅后都能有所收获。我相信，只要人们对新生事物保持着好奇心和探索勇气，前行的路上总会收获新知。行胜于言，路虽远，行则将至。

有关本书的问题反馈与交流探讨，请到本书的Github主页<https://github.com/icejean/GraphRAG>,也欢大家迎通过QQ或同名邮箱来信交流，QQ：[1793893070@qq.com](mailto:1793893070@qq.com)，谢谢大家！

作者

2025年2月27日于珠海

**目录**

[**前言**](#前言)

[**第1章 在WSL2上搭建GPU Linux Server深度学习环境**](#第1章在WSL2上搭建GPULinuxServer深度学习环境)

[**1.1 在Windows WSL2上安装Ubuntu 22**](#节11在WindowsWSL2上安装Ubuntu22)

[**1.2 安装Ubuntu 22 PyTorch深度学习开发环境**](#节12安装Ubuntu22PyTorch深度学习开发环境)

[**1.3 安装配置Jupyter Hub**](#节13安装配置JupyterHub)

[**1.4 在PyTorch上运行HanLP**](#节14在PyTorch上运行HanLP)

[**1.5 Neo4j Community安装配置**](#节15Neo4jCommunity安装配置)

[**第2章 微软GraphRAG**](#第2章微软GraphRAG)

[**2.1 微软GraphRAG测试**](#节21微软GraphRAG测试)

[**2.2 Ollama本地运行**](#节22Ollama本地运行)

[**第3章 Neo4j GraphRAG**](#第3章Neo4jGraphRAG)

[**3.1 安装Docker。**](#节31安装Docker)

[**3.2 Neo4j KGBuilder安装**](#节32Neo4jKGBuilder安装)

[**3.3 Neo4j KGBuilder测试**](#节33Neo4jKGBuilder测试)

[**3.4 在Python中调用KGBuilder**](#节34在Python中调用KGBuilder)

[**第4章 开发GraphRAG应用**](#第4章开发GraphRAG应用)

[**4.1 实体关系提取与导入**](#节41实体关系提取与导入)

[**4.2 实体工程：索引、合并与社区摘要**](#节42实体工程索引合并与社区摘要)

[**4.3 局部查询与全局查询**](#节43局部查询与全局查询)

[**第5章 Agent开发**](#第5章Agent开发)

[**5.1 实现多轮对话的GraphRAG局部查询**](#节51实现多轮对话的GraphRAG局部查询)

[**5.2 用LangSmith调试**](#节52用LangSmith调试)

[**5.3 用Agent实现局部查询**](#节53用Agent实现局部查询)

[**5.4 定制自己的Agent**](#节54定制自己的Agent)

[**5.5 为Agent增加全局查询**](#节55为Agent增加全局查询)

[**第6章 在GraphRAG中应用国产大模型**](#第6章在GraphRAG中应用国产大模型)

[**6.1 国产大模型接入LangChain**](#节61国产大模型接入LangChain)

[**6.2 用国产大模型构建知识图谱**](#节62用国产大模型构建知识图谱)

[**6.3 在Agent中应用国产大模型**](#节63在Agent中应用国产大模型)

[**第7章 本地部署LLM**](#第7章本地部署LLM)

[**7.1 Ollama本地部署LLM**](#节71Ollama本地部署LLM)

[**7.2 端侧小模型MiniCPM3测试**](#节72端侧小模型MiniCPM3测试)

[**7.3 其它端侧运行LLM的方法**](#节73其它端侧运行LLM的方法)

[**第8章 开发GraphRAG APP**](#第8章开发GraphRAGAPP)

[**8.1 用FastAPI输出Agent调用**](#节81用FastAPI输出Agent调用)

[**8.2 Shiny for Python开发环境安装配置**](#节82ShinyforPython开发环境安装配置)

[**8.3 Shiny for Python APP 开发**](#节83ShinyforPythonAPP开发)

[**第9章 GraphRAG应用评估**](#第9章GraphRAG应用评估)

[**9.1 用Ragas评估GraphRAG应用**](#节91用Ragas评估GraphRAG应用)

[**9.2 评估指标原理**](#节92评估指标的解释)

[**9.3 LangSmith查看LLM调用序列**](#节93LangSmith查看LLM调用序列)

**附录：本书引用网址列表**

[**https://github.com/icejean/GraphRAG/blob/main/ReferenceLinks.txt**](https://github.com/icejean/GraphRAG/blob/main/ReferenceLinks.txt)

**第1章 在WSL2上搭建GPU Linux Server深度学习环境**

前年写我的另一本书[《数里乾坤——用R与Python玩转大数据》](https://zhuanlan.zhihu.com/p/640437583)1（未出版）时，曾利用某大厂的推广优惠租用了一台GPU Linux服务器几个月，因为团队开发测试也好生产部署也好，真正落地应用GPU的深度学习环境，还是Linux服务器为主。该型号配备了4核CPU/8G RAM/100G SSD/Nvidia T4 GPU，学习研究刚好够用，但优惠期满后的价格有点贵，写完书不久就退租销毁了。现在一台最低配的8核CPU/32G RAM/T4 GPU的Linux虚拟主机，各大厂的月租要2千多到3千多（国内最优惠的中小厂[优刻得](https://www.ucloud.cn/)2是5百多），Nvidia T4的CUDA算力是7.5，我笔记本的Geforce RTX 2060 GPU算力也是7.5（可参阅在线文档[《Your GPU Compute Capability》](https://developer.nvidia.com/cuda-gpus)3），最新的联想[拯救者Y9000X](https://activity.lenovo.com.cn/xiaofei/zjz/hdy.html)4配备的Geforce RTX 4060/4070算力是8.9，1万多就有，还是16核/32G RAM，所以差不多算力的机器，个人租用超过5个月，就不如直接买一个。当然，如果需要一个公网IP对外提供服务或演示，还是要租大厂的GPU虚拟主机。个人学习或研究，买个高配的游戏本，Windows + WSL2 + Ubuntu双系统，比较合算。个人网站就租个不带GPU的虚拟主机，每年一两千不算多。学习大数据、AI专业的同学，入学时直接配个高配游戏本就可以。还有那么多存量带GPU的机器，也可以好好利用起来。所以先把WSL2 + Linux服务器配好，磨刀不误砍柴工。另外像PyTorch、Tensorflow等主流的深度学习框架，一般[Linux上是最新的](https://tensorflow.google.cn/install/source?hl=en#linux)5，[Windows上的支持就要慢一些](https://tensorflow.google.cn/install/source_windows?hl=en#gpu)6，只能用老一点的版本，所以一线的研究人员一般用Linux。

本书的各种模型，后端用的深度学习框架都是PyTorch，所以本书只涉及PyTorch。本书用的GPU是Nvidia Geforce RTX 2060、Tesla P40等，安装的是Nvidia的驱动程序及CUDA。

云端Ubuntu上的GPU虚拟主机，除了GPU驱动程序安装以外，与Windows + WSL2上的安装配置一样，本书也作了介绍，所以本书也适用于云端用Ubuntu 22或Ubuntu 24的GPU虚拟主机。

**1.1 在Windows WSL2上安装Ubuntu 22**

选择安装Ubuntu 22而不是目前最新的Ubuntu 24，是因为Ubuntu 22对各种GPU的兼容性比较好，各种深度学习及大模型应用的运行更稳定，稍早的CUDA版本只支持到Ubuntu 22。最新的CUDA版本已经支持Ubuntu 24，需要更新的GPU驱动程序。本书也适用于Ubuntu 24，除了Nvidia驱动程序的安装更新外，其它基本一样，书中也作了介绍。WSL2的另一个好处是可以和GPU共享CPU内存，可以加载比GPU显存大的模型，这对学习研究来说是很方便的。

**1.1.1 安装Hyper-V及WSL2**

Windows上（本书是Win 10）运行Linux要先安装Hyper-V及WSL（Windows Subsystem for Linux）。如果用云端的GPU Linux主机，可以跳过本小节及第2小节，直接从第3小节配置root用户及SSH服务开始。

使用Hyper-V功能的前提是，CPU必须支持虚拟化及二级地址转换功能(Intel-VT或AMD-V)。只要CPU型号不是特别老，一般都没问题。

1）确保Windows虚拟化功能已打开

参考《[Docker安装运行报错WSL问题排查方案](https://www.cnblogs.com/bokemoqi/p/17926296.html)7》一帖，先确保电脑开启了虚拟化功能，以便可以在Windows上运行WSL。首先，用快捷键Ctrl + Shift + Esc打开打开【任务管理器】，然后切换到【性能】视图，如图1-1所示：

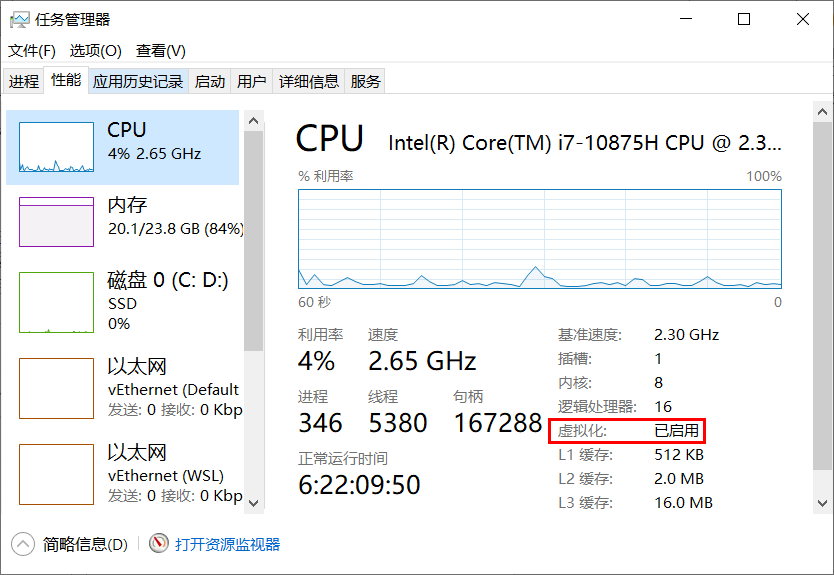


图1-1 打开Windows任务管理器确认虚拟化功能已启用

若没有开启，具体开启方法如下。

2）安装Hyper-V

一般笔记本电脑使用的Windows家庭版默认没有这个功能，参考[《Windows10/11家庭版开启Hyper-V虚拟机功能详解》](https://zhuanlan.zhihu.com/p/667571538)8一贴安装。建立一个代码如下的批处理文件Hyper-V.bat，并以管理员身份运行它来安装Hyper-V。

pushd "%~dp0"

dir /b %SystemRoot%\servicing\Packages\\*Hyper-V\*.mum >hv.txt

for /f %%i in ('findstr /i . hv.txt 2^>nul') do dism /online /norestart /add-package:"%SystemRoot%\servicing\Packages\%%i"

del hv.txt

Dism /online /enable-feature /featurename:Microsoft-Hyper-V -All /LimitAccess /ALL

Pause

之后，就会自动跳出Hyper-V的安装界面，整个安装过程都是在命令行中完成。组件数量比较多，安装过程需要几分钟时间，请耐心等待，不要强行中断。在看到“操作成功完成，重新启动Windows以完成该操作”的提示之后，在光标处输入字母“Y”，然后回车确认，重启电脑。

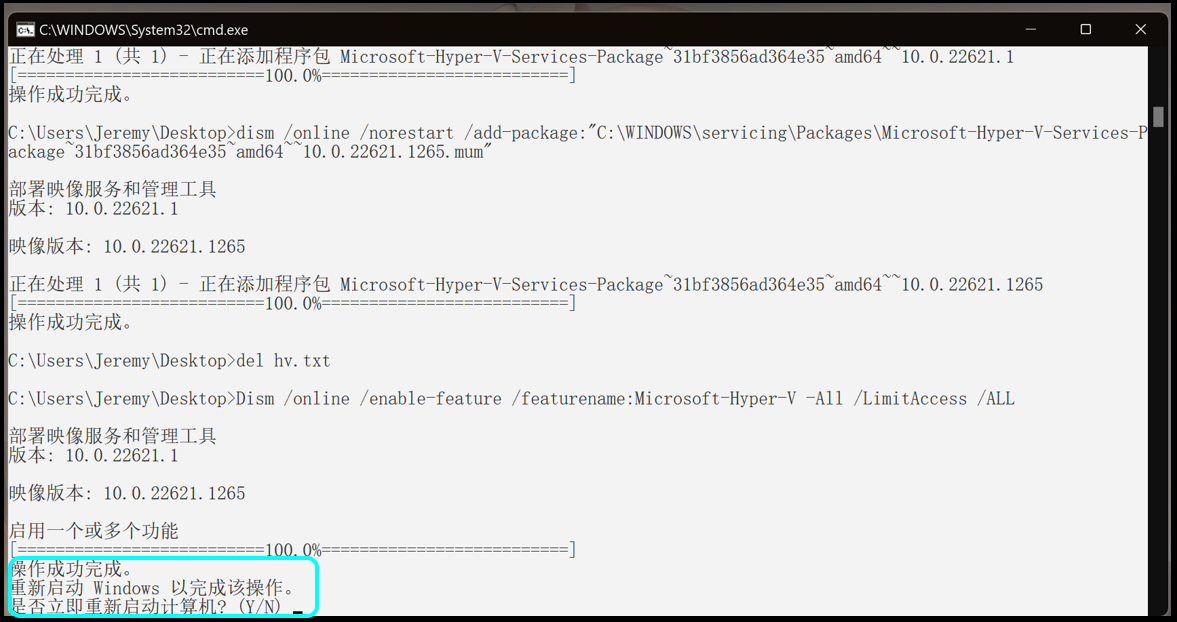


图1-2 Windows上安装Hyper-V功能

3）安装WSL2

可以从WSL的Github主页<https://github.com/microsoft/WSL>9下载安装。主页上持续更新，有最新的版本，在releases中选择最新的版本，在Assets中选择与CPU体系对应的msi安装文件，比如[wsl.2.4.11.0.x64.msi](https://github.com/microsoft/WSL/releases/download/2.4.11/wsl.2.4.11.0.x64.msi)10安装即可。

接下来，打开【Windows PowerShell】，将WSL2设置为默认使用版本，CUDA要求用WSL2，如图1-3所示：

wsl --set-default-version 2

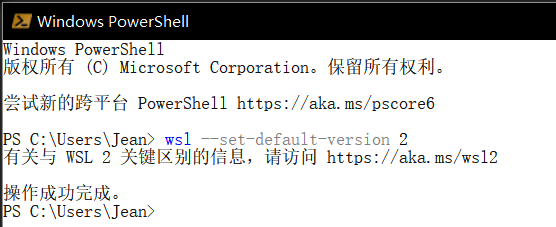


图1-3 设置默认使用WSL2

具体使用管理可以参阅[《WSL 使用教程》](https://www.jianshu.com/p/39a5b2e002b6)11。

**1.1.2 安装Ubuntu 22**

安装Ubuntu比较简单，打开Microsoft Store，搜索“ubuntu”，选择较新的Ubuntu 24.04.1 LTS或Ubuntu 22.04.5 LTS下载安装即可，这是Nvidia CUDA 12.x支持的版本（注：CUDA 12.6+ 已经支持Ubuntu 24，之前的版本只支持Ubuntu 22），本书用的是Ubuntu 22，Ubuntu 24上的操作是一样的。

注意Microsoft Store不支持代理，要先退出VPN代理。

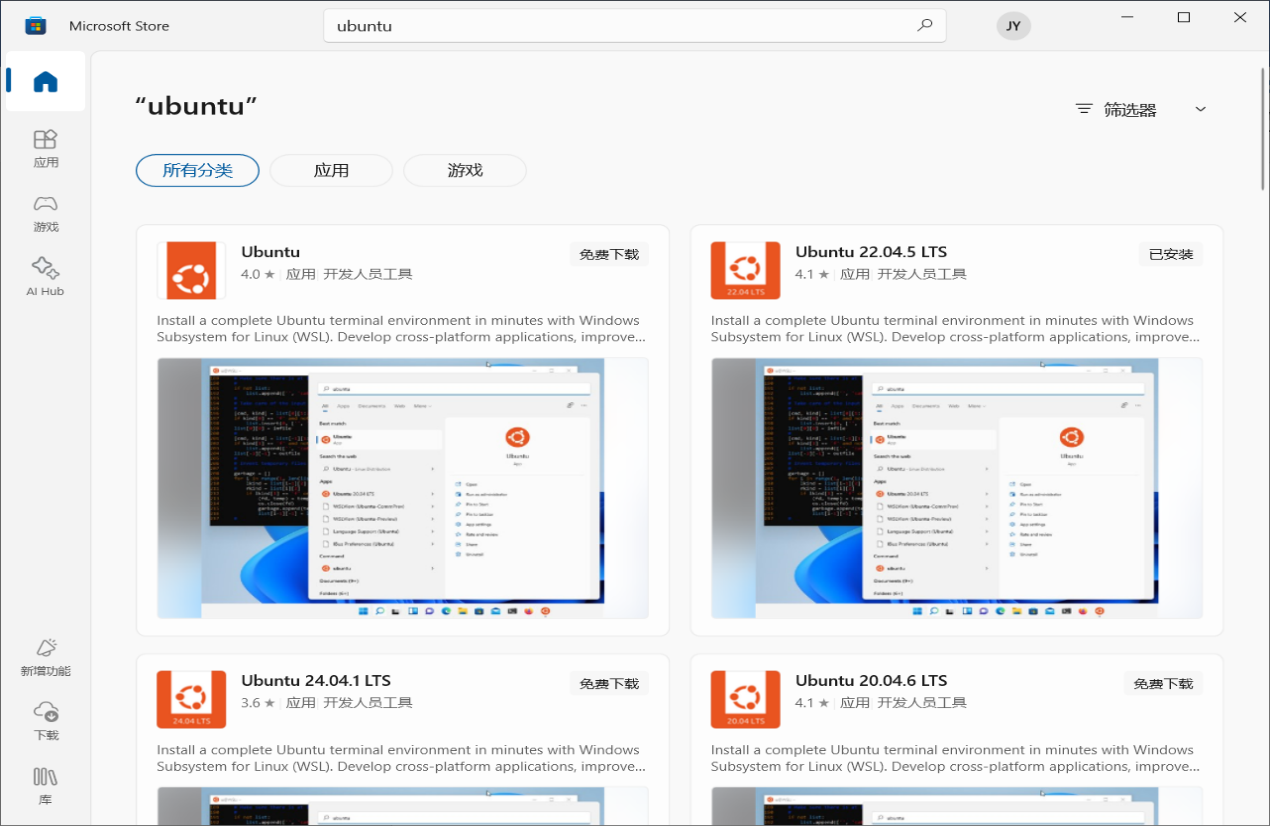


图1-4 Ubuntu 24.04是Nvidia CUDA 12.6+支持的最新版本

然后输入Ubuntu 22的初始用户名/口令，各大厂云端的镜像一般是用ubuntu这个用户名。

**注意：本文以下所有的Linux安装命令，如果没有特别说明都是用root用户执行。**

**1.1.3 安装配置ssh服务**

WSL2 Ubuntu 22安装好后就会自动启动，并打开一个Windows PowerShell命令行窗口。新开一个PowerShell窗口，用下面的命令可以列出所有的WSL虚拟机及其运行状态。

PS C:\Users\Jean> wsl -l -v

NAME STATE VERSION

\* Ubuntu-22.04 Running 2

用下面的命令可以停止某个特定虚拟机。

PS C:\Users\Jean> wsl -t Ubuntu-22.04

操作成功完成。

下面的命令可以启动某个特定虚拟机，并进入该虚拟机的终端窗口界面。

PS C:\Users\Jean> wsl -d Ubuntu-22.04

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.146.1-microsoft-standard-WSL2 x86\_64)

……

(base) root@Jean-Y9000X:/mnt/c/Users/Jean#

不过我还是习惯了用SecureCRT，Ubuntu 22默认是没有安装ssh服务的，要给它装上。

1）先设置root的口令

登录ubuntu激活root并以root登录，Ubuntu默认是不能用root的，包括远程登录。

$sudo passwd root

$su root

安装并启动ssh服务，apt update更新所有可安装软件包的元数据很重要，需要一点时间。

apt update

apt install openssh-server

systemctl status ssh

2）修改/etc/ssh/sshd\_config允许root远程登录

vi /etc/ssh/sshd\_config

改成这样：

PermitRootLogin yes

StrictModes yes

PasswordAuthentication yes

重启ssh:

systemctl restart ssh

这样在SecureCRT中就可以用root远程登录了。

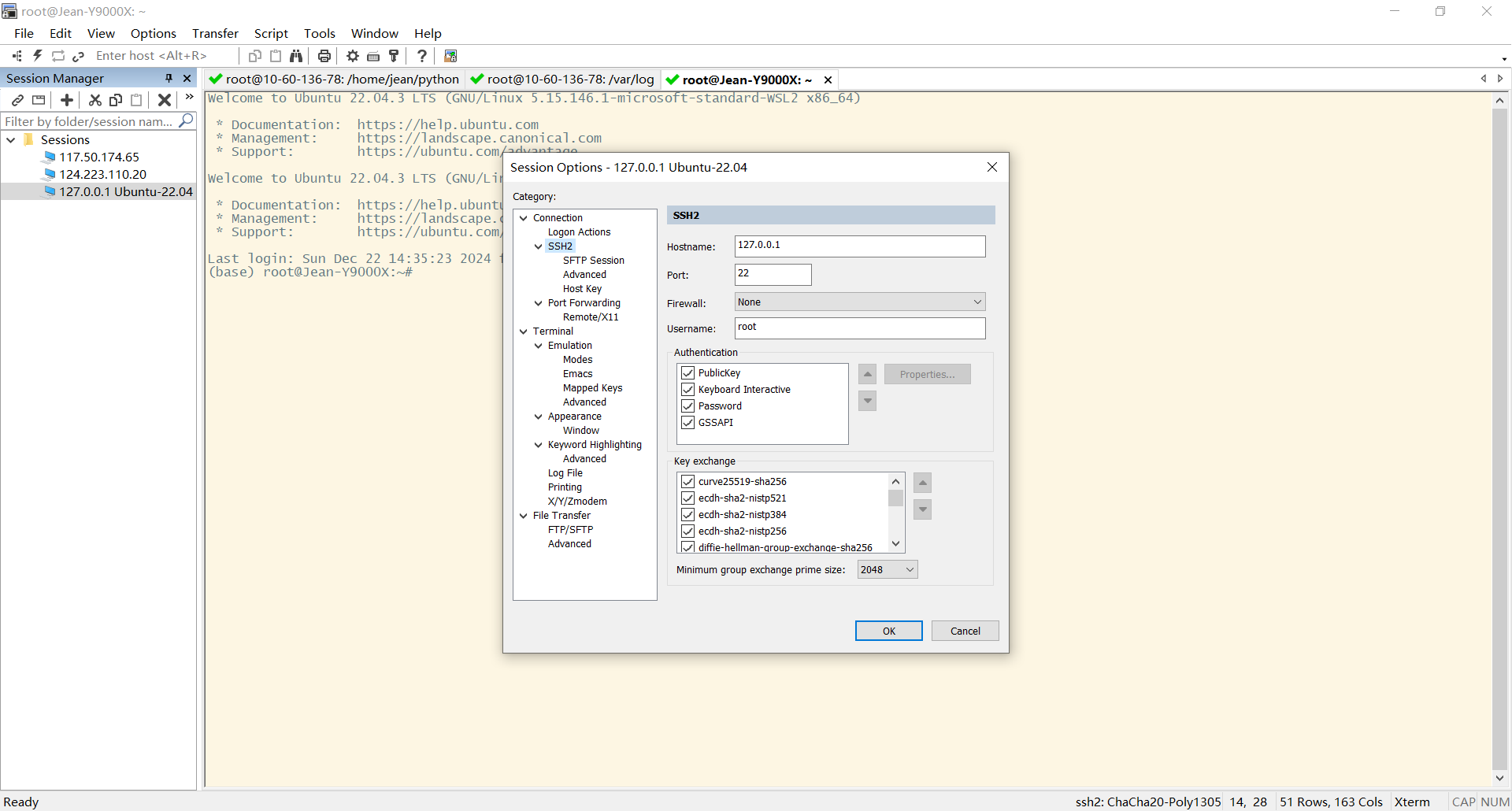


图1-5 用SecureCRT连接到Linux服务器

**1.1.4 把运行映像迁移到D盘**

WSL2 Ubuntu 22（虚拟硬盘映像）默认安装在C:/Users/<UserName>/AppData/Local/Packages/目录下，空白系统并不大，只有1G多，但装完本文的深度学习环境超过30G，所以一定要搬到D盘。在PowerShell命令行窗口中依次运行下面的命令即可。

wsl -l -v

wsl -t Ubuntu-22.04

wsl --export Ubuntu-22.04 d:\WSL\Ubuntu-22.04.tar

wsl --unregister Ubuntu-22.04

wsl --import Ubuntu-22.04 D:\WSL\Ubuntu-22.04 d:\WSL\Ubuntu-22.04.tar --version 2

重新启动Ubuntu 22。

wsl -d Ubuntu-22.04

**1.1.5 确认Nvidia驱动程序工作正常**

可以参阅文档[《Getting Started with CUDA on WSL 2》](https://docs.nvidia.com/cuda/wsl-user-guide/index.html#getting-started-with-cuda-on-wsl-2)12，只需下载安装Nvidia最新的显卡Windows驱动程序，就可以同时支持Windows及WSL，Ubuntu 22中不需要再安装Linux驱动程序。经过测试，更新Nvidia显卡的驱动程序不会影响已经在Windows上安装的CUDA及cuDNN各版本软件的运行。英文原文描述如下：

3. CUDA Support for WSL 2  
The latest NVIDIA Windows GPU Driver will fully support WSL 2. With CUDA support in the driver, existing applications (compiled elsewhere on a Linux system for the same target GPU) can run unmodified within the WSL environment.  
To compile new CUDA applications, a CUDA Toolkit for Linux x86 is needed. CUDA Toolkit support for WSL is still in preview stage as developer tools such as profilers are not available yet. However, CUDA application development is fully supported in the WSL2 environment, as a result, users should be able to compile new CUDA Linux applications with the latest CUDA Toolkit for x86 Linux.  
Once a Windows NVIDIA GPU driver is installed on the system, CUDA becomes available within WSL 2. The CUDA driver installed on Windows host will be stubbed inside the WSL 2 as libcuda.so, therefore users must not install any NVIDIA GPU Linux driver within WSL 2. One has to be very careful here as the default CUDA Toolkit comes packaged with a driver, and it is easy to overwrite the WSL 2 NVIDIA driver with the default installation. We recommend developers to use a separate CUDA Toolkit for WSL 2 (Ubuntu) available from the CUDA Toolkit Downloads page to avoid this overwriting. This WSL-Ubuntu CUDA toolkit installer will not overwrite the NVIDIA driver that was already mapped into the WSL 2 environment. To learn how to compile CUDA applications, please read the CUDA documentation for Linux.

如果是用云端GPU虚拟主机，可以参考这篇文档来安装Nvidia驱动程序：[《How to Install CUDA on Ubuntu 22.04 | Step-by-Step》](https://www.cherryservers.com/blog/install-cuda-ubuntu)13。更新显卡驱动程序以支持更新版本的CUDA可以参考[《How to Install NVIDIA Drivers on Ubuntu 24.04》](https://linuxconfig.org/how-to-install-nvidia-drivers-on-ubuntu-24-04)14里《Automatic Install using PPA repository》与《Manual Install using the Official Nvidia.com Driver》一节，以便安装Ubuntu 24。

1）确认Windows上驱动程序可以正常工作

这是写作时最新版的552.22，支持到最新的CUDA 12.4。

PS C:\Users\Jean> nvidia-smi

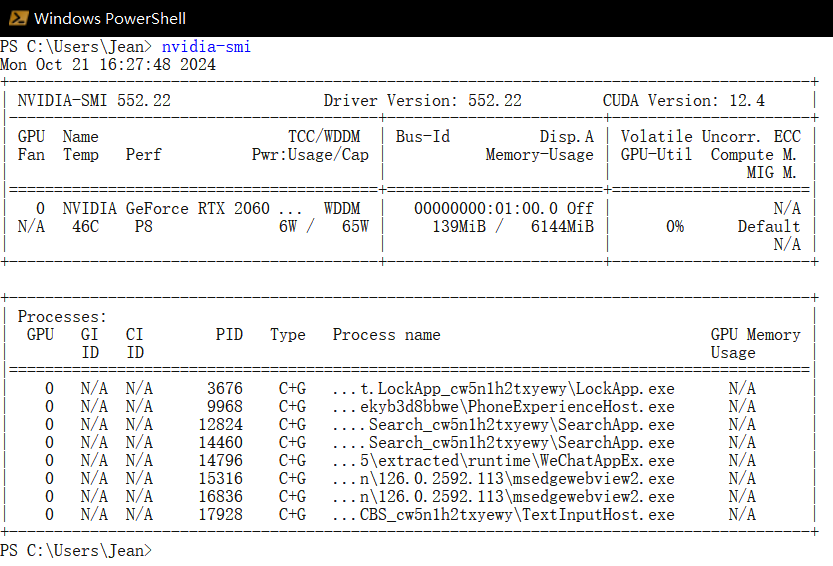


图1-6 确认Windows上Nvidia驱动程序工作正常

2）确认Ubuntu 22中驱动程序工作正常

先找到驱动程序被映射到WSL Ubuntu 22的路径，要排除Windows上的/mnt目录。

(base) root@Jean-Y9000X:~# find / -path "/mnt" -prune -o -name "nvidia-smi"

/usr/lib/wsl/lib/nvidia-smi

建立一个软连接：

ln -s /usr/lib/wsl/lib/nvidia-smi /usr/bin/nvidia-smi

测试。

(base) root@Jean-Y9000X:~# nvidia-smi

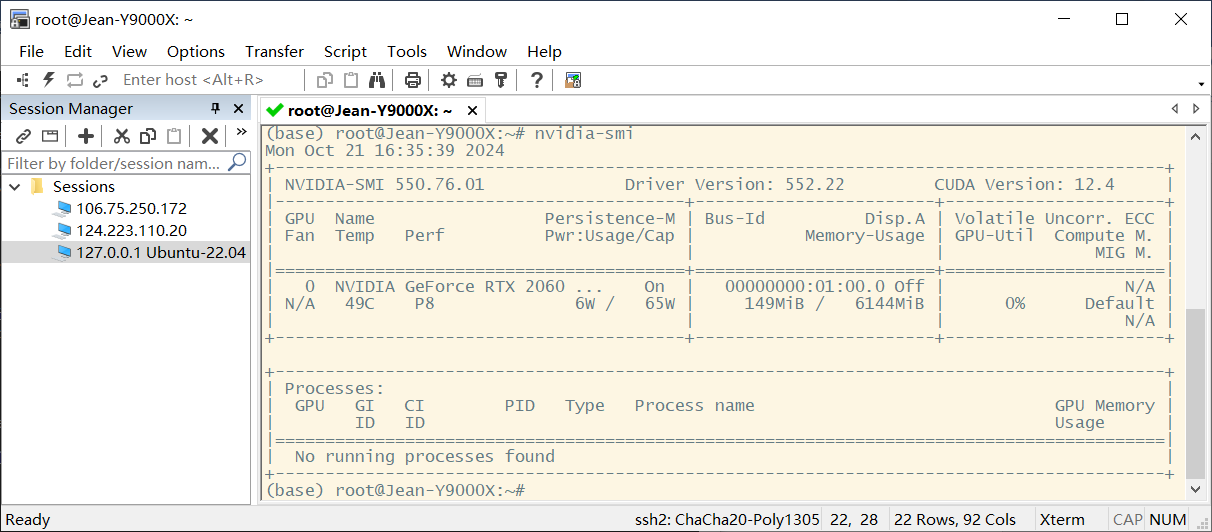


图1-7 确认Ubuntu 22上Nvidia驱动程序工作正常

**1.2 安装Ubuntu 22 PyTorch深度学习开发环境**

最新版的PyTorch会自动安装所依赖的CUDA runtime，不过这里还是介绍一下自己手动安装CUDA及cuDNN，以便读者有需要时安装Tensorflow等其它深度学习环境，并且PyTorch自动安装的CUDA runtime是不带nvcc编译器等完整CUDA开发环境工具的，后面第7章《本地部署LLM》中从源码编译安装vLLM服务器时要用到，所以还是建议手动安装。

**1.2.1 安装cuda-toolkit**

访问https://pytorch.org/，查看最新稳定版的PyTorch，以及与之匹配的CUDA版本，当前最新的稳定版是2.5,它可以匹配CUDA 12.1以上的版本。

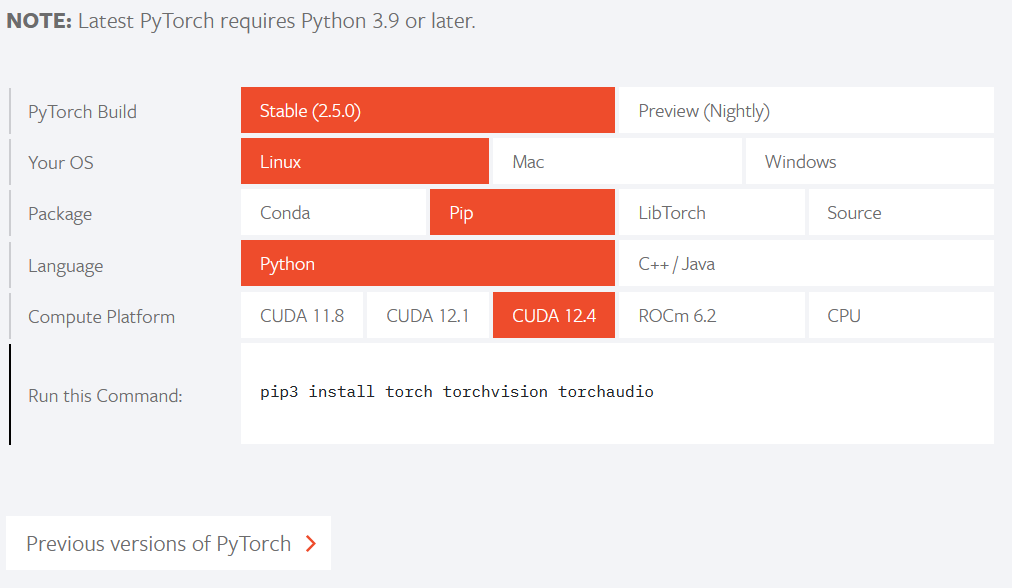


图1-8 确认当前PyTorch稳定版本支持的CUDA版本

访问[CUDA 12.3下载地址15](https://developer.nvidia.com/cuda-12-3-2-download-archive?target_os=Linux&target_arch=x86_64&Distribution=WSL-Ubuntu&target_version=2.0&target_type=deb_local)，（Tensorflow对CUDA版本的跟随要慢一点，这是Tensorflow目前支持的最高版本，具体可以参阅[Tensorflow的兼容列表](https://tensorflow.google.cn/install/source?hl=en#linux)16）选择Linux->x86-64->WSL-Ubuntu->2.0->deb(network)。根据前面Nvidia的[参考文档](https://docs.nvidia.com/cuda/wsl-user-guide/index.html#getting-started-with-cuda-on-wsl-2)12，WSL-Ubuntu版的cuda-toolkit是不带驱动程序的，不要选错。

如果是云端的GPU虚拟主机，这里选Ubuntu->22.04即可。

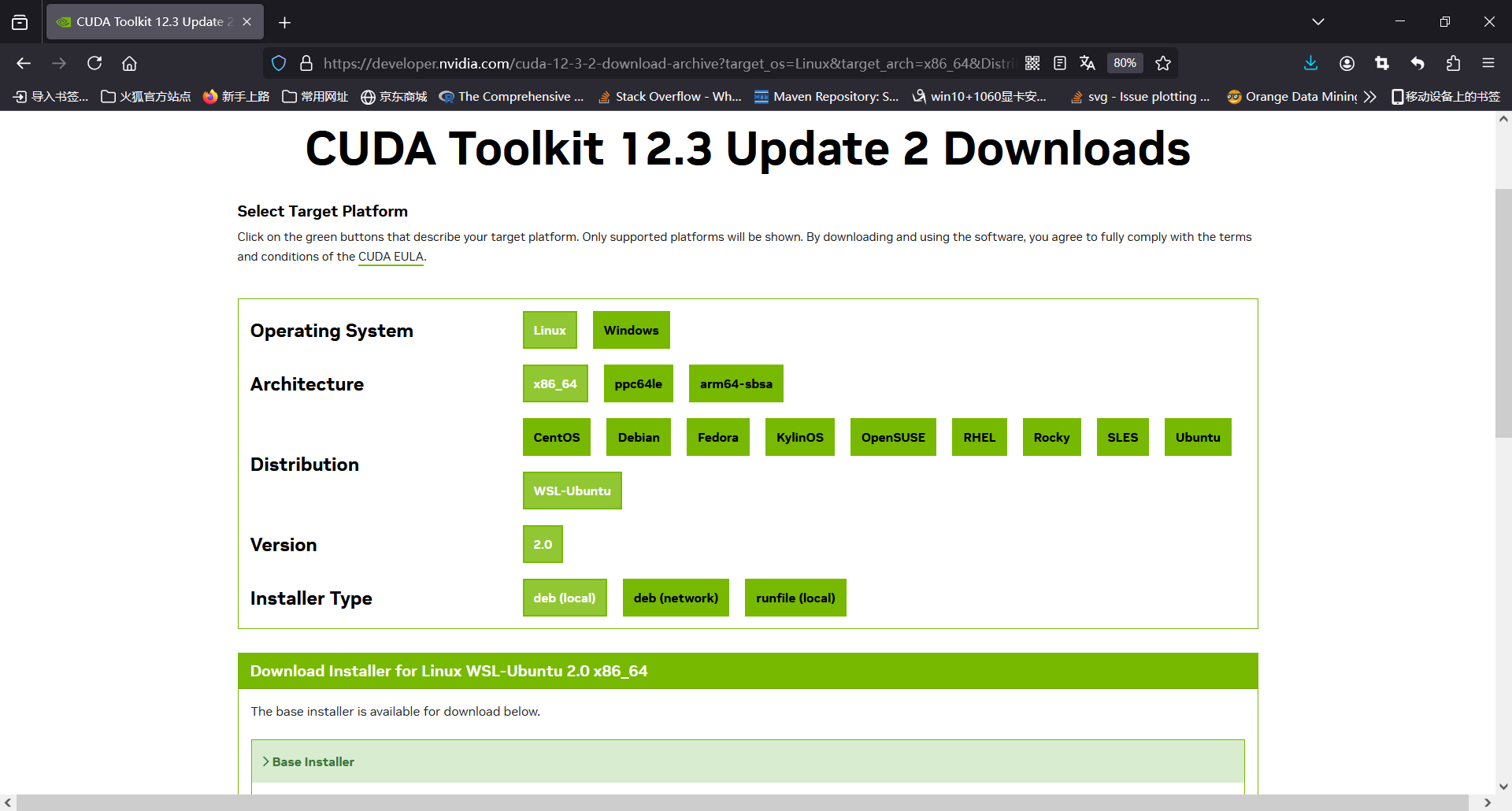


图1-9 在线安装安装cuda-toolkit-12-3

然后就会显示下面的下载及在线安装命令，依次执行即可。在线安装的好处是不会在Ubuntu的虚拟硬盘中占用额外的安装包存储空间，有几个G。

wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86\_64/cuda-keyring\_1.1-1\_all.deb

sudo dpkg -i cuda-keyring\_1.1-1\_all.deb

sudo apt-get update

sudo apt-get -y install cuda-toolkit-12-3

装完后找到nvcc的位置：

(base) root@Jean-Y9000X:~# find / -path "/mnt" -prune -o -name "nvcc"

/usr/local/cuda-12.3/bin/nvcc

可以看到/usr/local中已经通过/etc/alternatives/cuda建立了到cuda-12.3的软连接。

(base) root@Jean-Y9000X:/usr/local# ls -l

total 36

drwxr-xr-x 2 root root 4096 Apr 27 10:43 bin

lrwxrwxrwx 1 root root 22 Apr 27 10:43 cuda -> /etc/alternatives/cuda

lrwxrwxrwx 1 root root 25 Apr 27 10:43 cuda-12 -> /etc/alternatives/cuda-12

drwxr-xr-x 15 root root 4096 Apr 27 10:43 cuda-12.3

建立nvcc的软连接：

ln -s /usr/local/cuda/bin/nvcc /usr/bin/nvcc

验证安装：

root@Jean-Y9000X:~# nvcc -V

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2023 NVIDIA Corporation

Built on Wed\_Nov\_22\_10:17:15\_PST\_2023

Cuda compilation tools, release 12.3, V12.3.107

Build cuda\_12.3.r12.3/compiler.33567101\_0

**1.2.2 安装cuDNN9.x**

cuDNN是CUDA平台上用于深度神经网络的GPU加速库，很多深度学习模型都要用到，它对CUDA的兼容列表可以参阅它的[**Support Matrix**](https://docs.nvidia.com/deeplearning/cudnn/latest/reference/support-matrix.html#support-matrix)**17**。

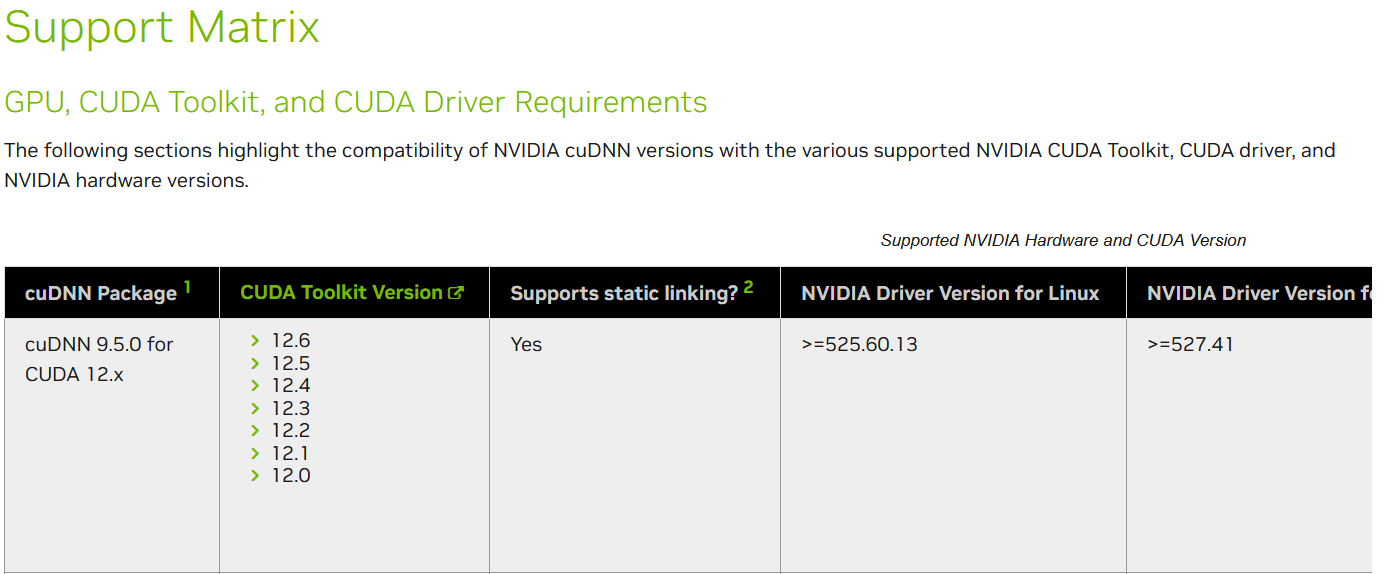
****

图1-10 cuDNN对CUDA的兼容列表

cuDNN最新的9.x版本提供了在线安装，目前Tensorflow支持的最高版本8.9版要下载安装包到本地安装，比较复杂，可以参考[《Ubuntu 20.04(linux) cuda(12)+cudnn的deb方式安装以及验证》](https://blog.csdn.net/qq_32033383/article/details/135015041)18一贴。在线安装9.X版则要方便很多，本书不使用Tensorflow，直接用在线安装即可。访问[cuDNN9.4下载地址](https://developer.nvidia.com/cudnn-9-4-0-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=22.04&target_type=deb_network)16，选择Linux->x86-64->Ubuntu->22.04->deb(network)：

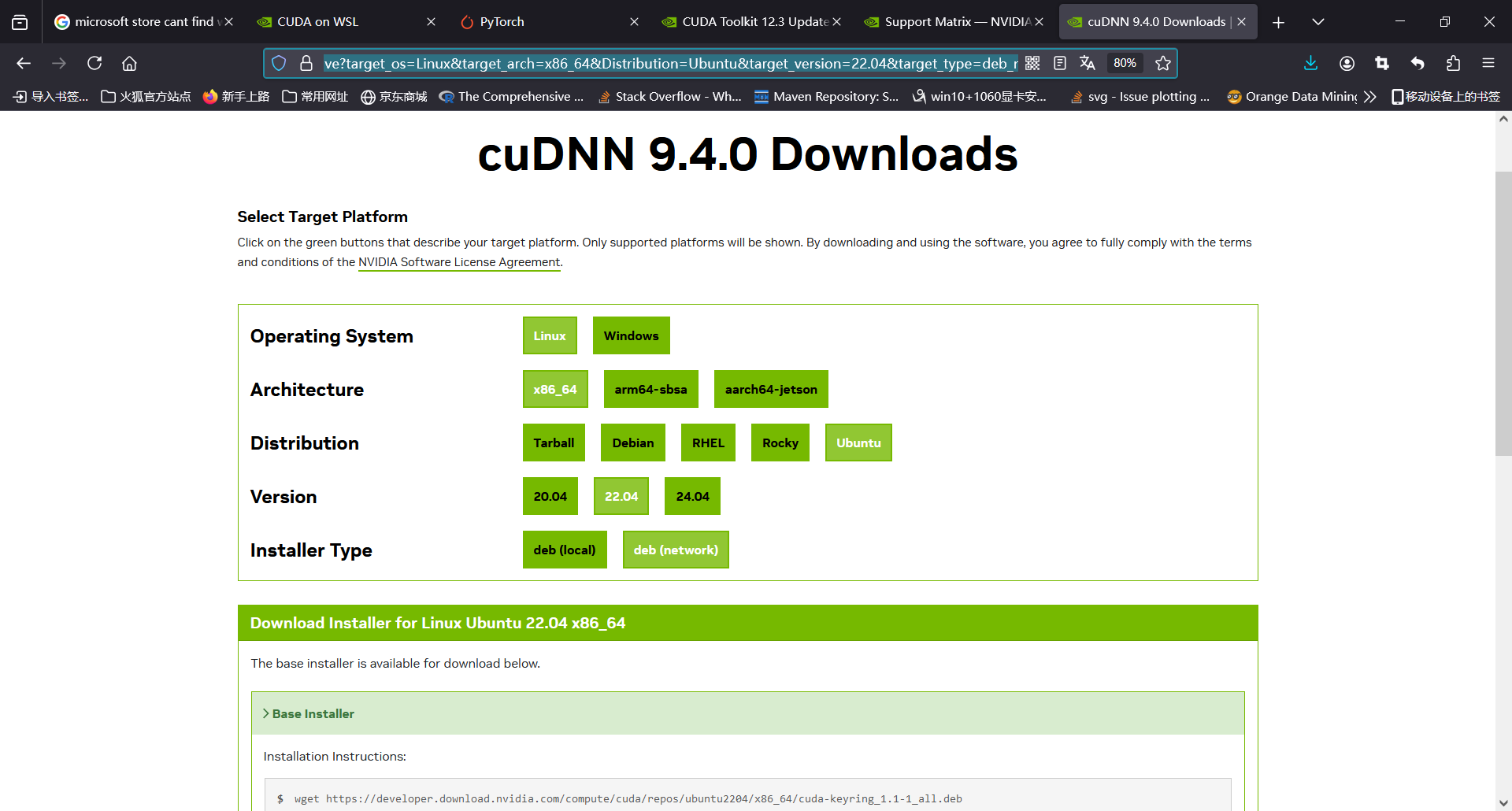


图1-11 在线安装cuDNN9.4.0

然后就会显示下面的下载及在线安装命令，依次执行即可。

wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86\_64/cuda-keyring\_1.1-1\_all.deb

sudo dpkg -i cuda-keyring\_1.1-1\_all.deb

sudo apt-get update

sudo apt-get -y install cudnn-cuda-12

验证安装：

(base) root@10-23-6-18:~# cat /usr/include/cudnn\_version.h | grep CUDNN\_MAJOR -A 2

#define CUDNN\_MAJOR 9

#define CUDNN\_MINOR 4

#define CUDNN\_PATCHLEVEL 0

……

**1.2.3 将CUDA目录加入全局环境变量**

vi /etc/profile

加入下面的内容：

export PATH=/usr/local/cuda/bin:$PATH

export LD\_LIBRARY\_PATH=/usr/local/cuda/lib64:$LD\_LIBRARY\_PATH

export CUDA\_HOME=/usr/local/cuda

**1.2.4 安装Anaconda3**

Anaconda是一个开源的Python和R语言的发行版本，用于计算科学（数据科学、机器学习、大数据处理和预测分析），Anaconda致力于简化软件包管理系统和部署。它透过conda进行软件包管理，并拥有许多适用于Windows、Linux和MacOS的数据科学软件包。

下载[Linux版64-Bit (x86) Installer](https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-x86_64.sh)20，可以在Windows上用迅雷下载到D:\迅雷下载，这样快很多，然后安装：

cd /mnt/d/迅雷下载

bash Anaconda3-2024.02-1-Linux-x86\_64.sh

我安装到/usr/lib64/anaconda3目录下，然后在/etc/profile中加入到PATH：

export PATH=/usr/lib64/anaconda3/bin:$PATH

执行conda init会更改用户的.bashrc，自动加入下面的一段代码，用户登录时就会自动激活base环境：

# >>> conda initialize >>>

# !! Contents within this block are managed by 'conda init' !!

\_\_conda\_setup="$('/usr/lib64/anaconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"

if [ $? -eq 0 ]; then

eval "$\_\_conda\_setup"

else

if [ -f "/usr/lib64/anaconda3/etc/profile.d/conda.sh" ]; then

. "/usr/lib64/anaconda3/etc/profile.d/conda.sh"

else

export PATH="/usr/lib64/anaconda3/bin:$PATH"

fi

fi

unset \_\_conda\_setup

# <<< conda initialize <<<

更改conda配置，使其指向国内清华大学的镜像：

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/

conda config --set show\_channel\_urls yes

conda config --show channels

**1.2.5 安装PyTorch**

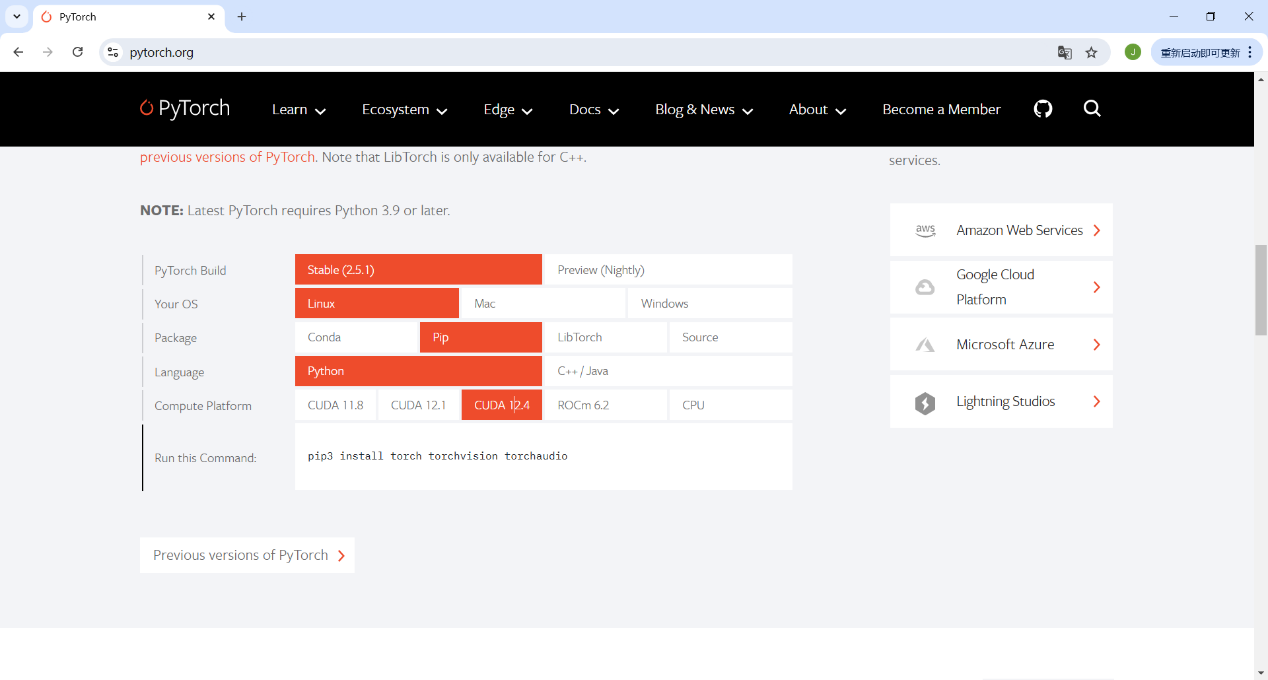


图1-12 PyTorch在线安装更为简单

[PyTorch](https://pytorch.org/)21主页上提供了目前稳定版本的安装选项，会自动生成在线安装脚本。目前稳定的版本是PyTorch 2.5.0，匹配CUDA 12.1及12.4，不过根据网上大家的测试，CUDA 12.x都是可以的，没有Tensorflow那么严格，所以与Tensorflow一起用CUDA 12.3没有问题。

创建一个独立的虚拟环境pytorch并在虚拟环境中安装PyTorch：

conda create --name pytorch python=3.11

conda activate pytorch

conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia

最新的PyTorch安装程序会自动安装所依赖的CUDA运行环境，与前面安装的CUDA开发环境会有部分重复。但它不包括完整CUDA开发环境的一些工具，比如nvcc编译器等。

(pytorch) root@10-23-6-18:~# conda list nvidia

……

nvidia-cuda-runtime-cu12 12.4.99 pypi\_0 pypi

nvidia-cudnn-cu12 9.1.0.70 pypi\_0 pypi

……

验证GPU调用成功。

(pytorch) root@Jean-Y9000X:~# python

Python 3.11.9 (main, Apr 19 2024, 16:48:06) [GCC 11.2.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> import torch

>>> torch.\_\_version\_\_

'2.3.0'

>>> print(torch.cuda.is\_available())

True

>>> device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')

>>> device

device(type='cuda', index=0)

>>> print(torch.cuda.get\_device\_name(0))

NVIDIA GeForce RTX 2060 with Max-Q Design

清理Ubuntu 22虚拟硬盘。

conda clean --all

pip cache purge

**1.2.6 设置用户ubuntu的conda运行环境**

主要就是su -l ubuntu执行conda init，这样用户ubuntu登录后就会自动激活conda base环境：

ubuntu@Jean-Y9000X:~$ conda init

……

modified /home/ubuntu/.bashrc

……

**1.3 安装配置Jupyter Hub**

前面用Python命令行测试了对GPU的调用，对真正开发不是很方便。conda base环境已经预装了Jupyter Lab，它是单用户的，Jupyter Hub则支持Linux服务器多用户环境中每个用户分别使用自己的Jupyter Lab，在团队开发环境中就比较方便，这里讲讲它的安装配置。

**1.3.1 安装Jupyter Hub**

(base) ubuntu@Jean-Y9000X:~$ conda list jupyter

……

jupyterlab 4.0.11 py311h06a4308\_0

……

旧版的configurable-http-proxy需要先安装npm和node.js：

apt install -y npm nodejs

检查一下版本：

(base) root@Jean-Y9000X:~# node -v

v12.22.9

(base) root@Jean-Y9000X:~# npm -v

8.5.1

最新版的configurable-http-proxy改用纯Python实现，直接安装即可。

conda install configurable-http-proxy

conda install jupyterhub

**1.3.2 配置Jupyter Hub**

新建目录/etc/jupyterhub，在该目录下新建一个配置文件，编辑文件。

(gpu) root@VM-0-14-ubuntu:~# mkdir /etc/jupyterhub

(gpu) root@VM-0-14-ubuntu:~# cd /etc/jupyterhub

(gpu) root@VM-0-14-ubuntu:/etc/jupyterhub# jupyterhub --generate-config

Writing default config to: jupyterhub\_config.py

(gpu) root@VM-0-14-ubuntu:/etc/jupyterhub# vi jupyterhub\_config.py

内容如下：

c.Authenticator.whitelist = {'ubuntu'} # 允许使用Jupyter Hub的用户列表，逗号分隔。

c.Authenticator.admin\_users = {'ubuntu'} #Jupyter Hub的管理员用户列表

c.Spawner.notebook\_dir = '/home/{username}' #浏览器登录后进入用户的主目录

c.Spawner.default\_url = '/lab' # 使用Jupyter Lab而不是Notebook

c.JupyterHub.extra\_log\_file = '/var/log/jupyterhub.log' #Jupyter Hub额外的日志文件

**1.3.3 配置Jupyter Hub为开机自启动服务**

先看看conda base环境的PATH设置：

(base) root@Jean-Y9000X:~# echo $PATH

/usr/lib64/anaconda3/bin:/usr/lib64/anaconda3/condabin:/usr/local/cuda/bin:/usr/lib64/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

然后新建一个系统守护进程的配置文件：

vi /etc/systemd/system/jupyterhub.service

内容如下，几个要点。

1）以root运行。

2）设定PATH路径，因为开机启动进程没有登录的过程，不会执行/etc/profile等设置环境变量，需要把上面的PATH拷进去。

3）用全路径引用执行jupyterhub。

4）引用上面创建的Jupyter Hub配置文件。

[Unit]

Description=Jupyterhub service

After=syslog.target network.target

[Service]

User=root

Environment="PATH=/usr/lib64/anaconda3/bin:/usr/lib64/anaconda3/condabin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

ExecStart=/usr/lib64/anaconda3/bin/jupyterhub -f /etc/jupyterhub/jupyterhub\_config.py

[Install]

WantedBy=multi-user.target

守护进程的配置文件更改后要先运行下面的命令来使它生效：

Systemctl enable jupyterhub.service

systemctl daemon-reload

然后可以用以下的命令来管理它：

systemctl start/stop/restart/status jupyterhub

现在可以用上面配置文件中指定的Ubuntu系统用户名及口令登录进Jupyter Hub打开自己的Jupyter Lab了。访问http://localhost:8000：

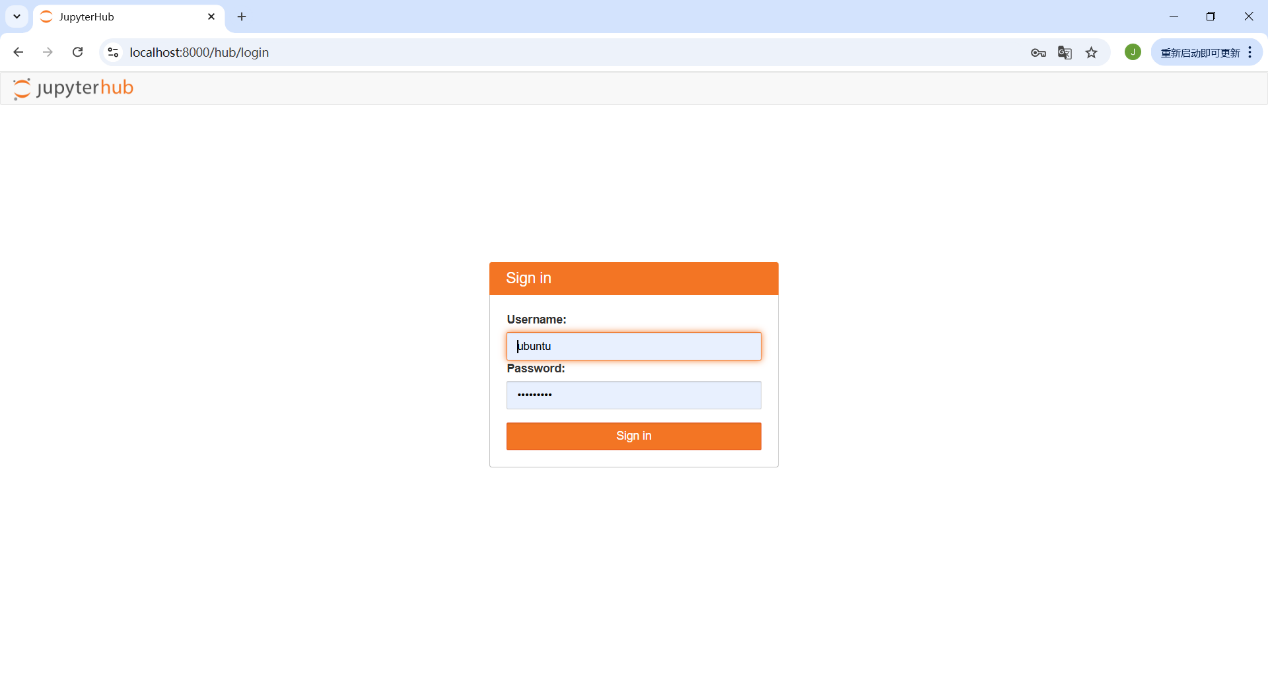


图1-13 Jupyter Hub用Linux用户名口令登录

新建Notebook时，默认的Python kernel是conda base环境的ipykernel。

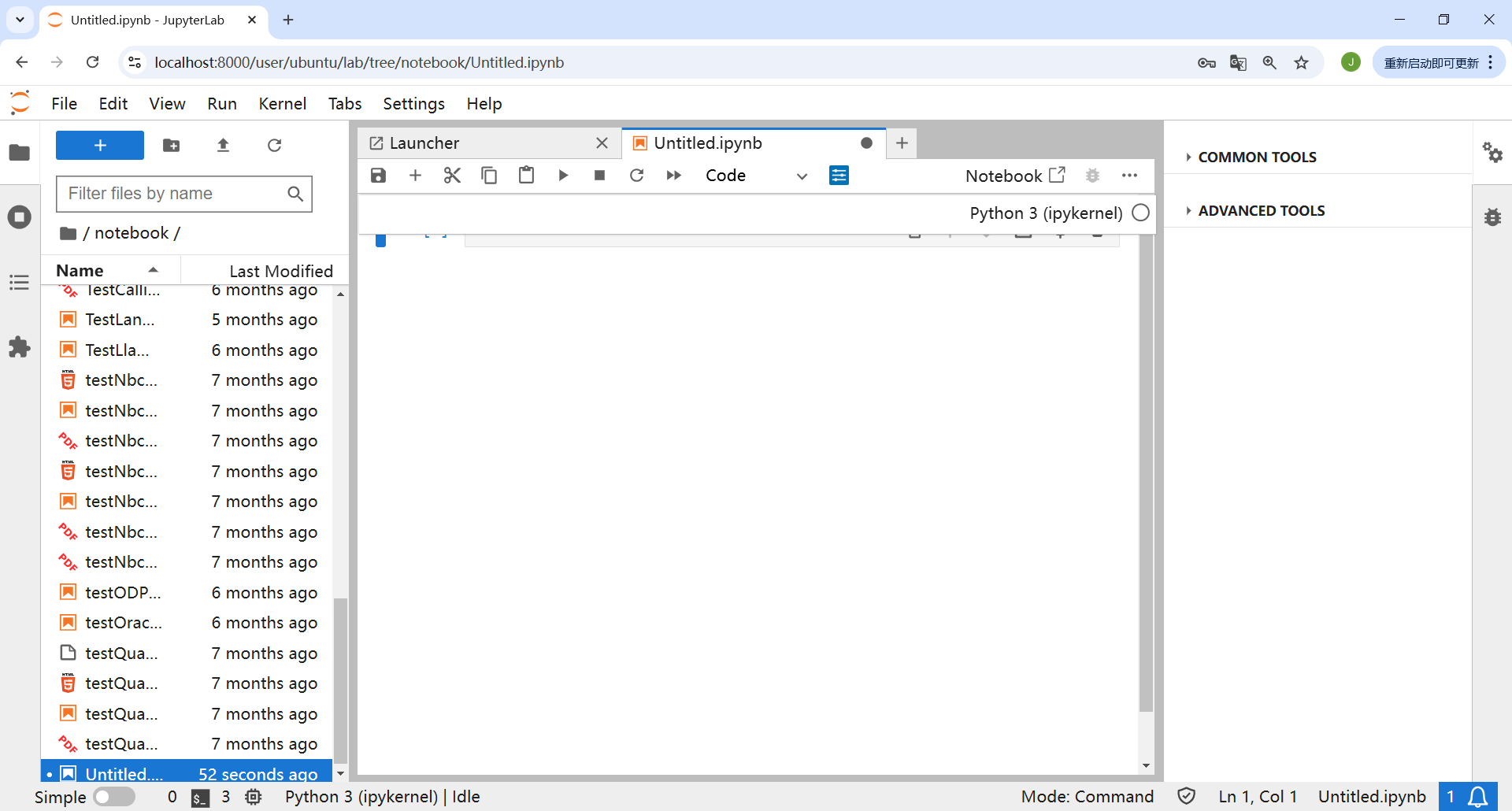


图1-14 Jupyter Lab默认只有conda base环境一个kernel

**1.3.4 为Jupyter Lab添加虚拟环境**

参考网贴[《Jupyter Lab设置切换虚拟环境》](https://blog.csdn.net/CUFEECR/article/details/123987150)22。前面已经创建了虚拟环境pytorch，现在修改配置以便在Jupyter Lab中引用它们。添加虚拟环境pytorch：

conda activate pytorch

conda install ipykernel

python -m ipykernel install --name pytorch --display-name pytorch

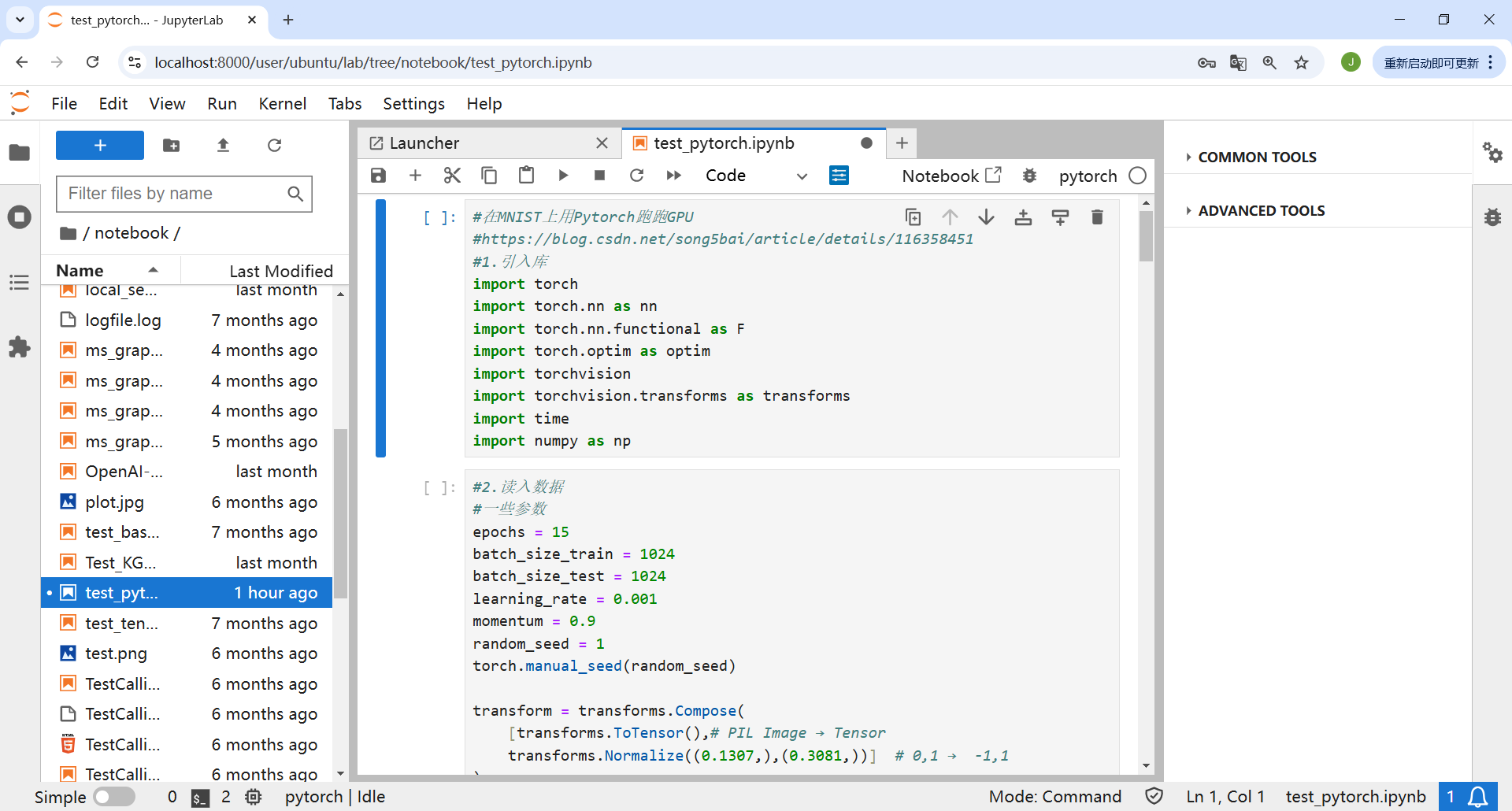


图1-15 Jupyter Lab使用虚拟环境pytorch

然后重启一下系统就可以了（上面两个截图都是重启服务后的），或者关闭所有已经打开的Jupyter Hub服务器进程，重启一下Jupyter Hub服务也可以。

systemctl restart jupyterhub.service

**1.3.5 PyTorch GPU测试例子**

现在用PyTorch来实现经典的MNIST手写体数字识别例子，详见[《在MNIST上用Pytorch跑跑GPU》](https://blog.csdn.net/song5bai/article/details/116358451)23一贴，具体源码如下：

#https://blog.csdn.net/song5bai/article/details/116358451

#1.引入库

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

#2.读入数据

#一些参数

batch\_size\_train = 200

batch\_size\_test = 1000

learning\_rate = 0.0073564225

momentum = 0.9

random\_seed = 1

torch.manual\_seed(random\_seed)

transform = transforms.Compose(

[transforms.ToTensor(),# PIL Image → Tensor

transforms.Normalize((0.1307,),(0.3081,))] # 0,1 → -1,1

)

trainset = torchvision.datasets.MNIST(download=True,root='./data',train=True,transform=transform)

trainloader = torch.utils.data.DataLoader(trainset,batch\_size=batch\_size\_train,shuffle=True,num\_workers=2)

testset = torchvision.datasets.MNIST(download=True,root='./data',train=False,transform=transform)

testloader = torch.utils.data.DataLoader(testset,batch\_size=batch\_size\_test,shuffle=True,num\_workers=2)

# explore testing data

examples = enumerate(trainloader)

batch\_idx, (example\_data, example\_targets) = next(examples)

print(example\_targets)

print(example\_data.shape)

#展示数据（示例）：

import matplotlib.pyplot as plt

fig = plt.figure()

for i in range(8):

plt.subplot(2,4,i+1)

plt.tight\_layout()

plt.imshow(example\_data[i][0], cmap='gray', interpolation='none')

plt.title("Ground Truth: {}".format(example\_targets[i]))

plt.xticks([])

plt.yticks([])

plt.show()

#3.定义卷积神经网络

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net,self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1,6,5)

self.pool = nn.MaxPool2d(2,2)

self.conv2 = nn.Conv2d(6,16,3)

self.fc1 = nn.Linear(16\*5\*5,120) #3-dim (0,1,2)

self.fc2 = nn.Linear(120,84)

self.fc3 = nn.Linear(84,10)

def forward(self,x):

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))#

#print(x.size()) # 4\*16\*5\*5

x = x.view(-1,16\*5\*5)

x = F.relu(self.fc1(x))

x = F.relu(self.fc2(x))

x = self.fc3(x)

return x

net = Net()

#GPU（示例）：

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

print(device)

net = net.to(device)

#优化设置（示例）：

optimizer = optim.SGD(net.parameters(),lr=learning\_rate,momentum=momentum)

# 损失函数

criterion = nn.CrossEntropyLoss()

#4.训练

epochs=15

for epoch in range(epochs):

running\_loss = 0.0

for i,data in enumerate(trainloader,0):

images,labels = data

images = images.to(device)

labels = labels.to(device)

optimizer.zero\_grad()

outputs = net(images).to(device)

loss = criterion(outputs,labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

if i%200 == 199:

print(f'{epoch+1}, {i+1}; loss:{running\_loss/200}')

running\_loss = 0.0

print('Finished Traing')

#5.在测试集上测试模型

correct = 0

total = 0

with torch.no\_grad():

for data in testloader:

images,labels = data

images = images.to(device)

labels = labels.to(device)

outputs = net(images)

\_,predicted = torch.max(outputs.data,1)

total += labels.size(0)

correct += (predicted==labels).sum().item()

print(f'Accuracy :{100\*correct/total} %')

#各类的精确度（示例）：

classes = (0,1,2,3,4,5,6,7,8,9)

class\_correct = [0 for \_ in range(10)]

class\_total = [0 for \_ in range(10)]

with torch.no\_grad():

for data in testloader:

images,labels = data

images = images.to(device)

labels = labels.to(device)

outputs = net(images)

\_,predicted = torch.max(outputs.data,1)

c = (predicted==labels).squeeze() # 1\*4 → 4

for i in range(4):

label = labels[i]

class\_correct[label] += c[i].item()

class\_total[label] += 1

for i in range(10):

print('Accuracy of %3s : %2d %%'%(classes[i],100\*class\_correct[i]/class\_total[i]))

#保存和加载（示例）：

PATH = "./data/MNIST.model"

#快速保存我们训练过的模型

torch.save(net.state\_dict(), PATH)

#重新加载保存的模型

net.load\_state\_dict(torch.load(PATH))

先安装一下作图用到的matplotlib：

conda activate pytorch

conda install matplotlib

然后浏览器登录Jupyter Lab，建立一个使用pytorch kernel的Notebook，上面的每节程序可以作为Notebook的一个代码块，然后运行Notebook看看。

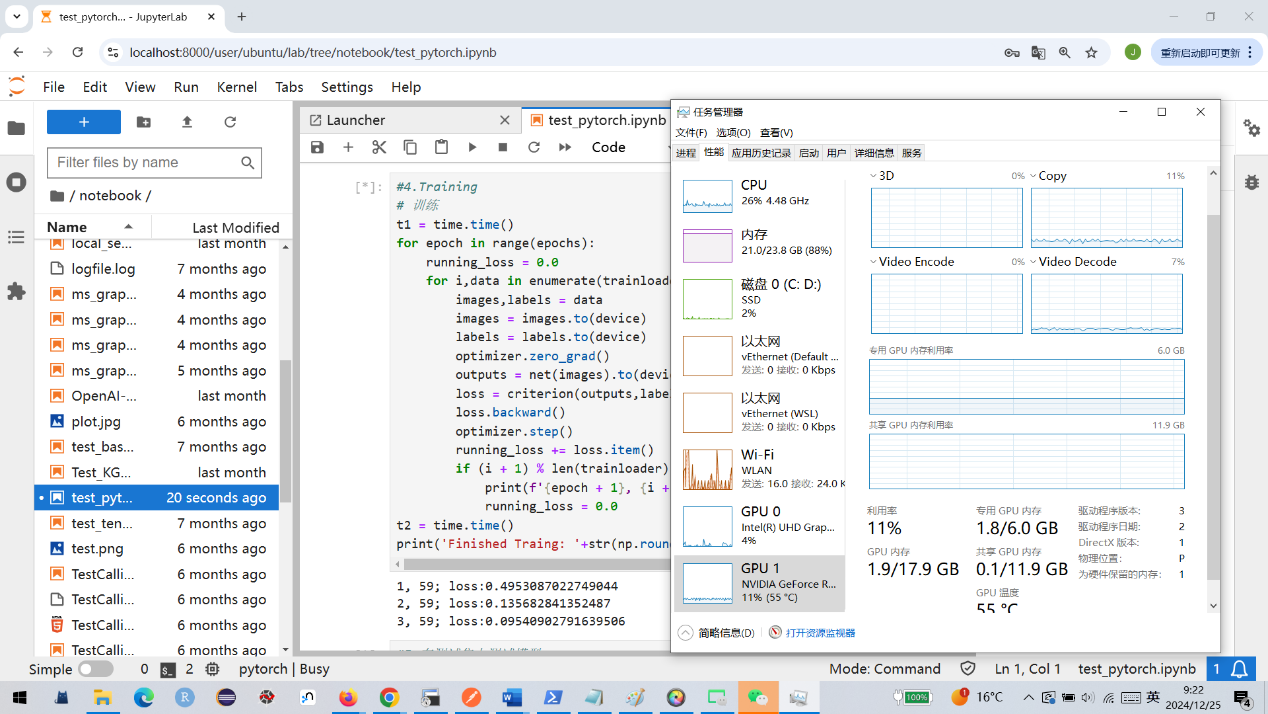


图1-16 测试例子运行时Nvidia GPU1的负荷与显存使用情况

**Jupyter Hub主要用于开发调试“.ipynb”后缀的Jupyter Notebook程序。本书的大部分源码都是“.py”后缀的Python程序，有很多IDE可以开发调试，读者可能都有自己熟悉的IDE，在此不再展开介绍。作者用的是Rstudio Server，一个浏览器WEB界面的IDE，非常方便好用，因为它同时也用于R语言的开发调试，Python与R都是数据科学工作中使用的主要语言。因为本章篇幅与全书结构的关系，Rstudio Server开发环境的安装配置放在第8章《开发GraphRAG APP》的第8.2节《Shiny for Python开发环境安装配置》中，感兴趣的读者可以先跳到该节把Rstudio Server与Shiny Server及Shiny for Python开发环境装好，以便可以在它们上面跑起本书后面章节中“.py”后缀的Python程序。**

**1.4 在PyTorch上运行HanLP**

[HanLP](https://github.com/hankcs/HanLP)24是Python生态中领先的基于深度学习预训练模型的开源NLP工具包，支持中文等一百多种语言和十几种自然语言处理任务，由[何晗](https://github.com/hankcs)25开发，山东青岛的[自然语义科技有限公司](https://www.hanlp.com/)26运营。它有云端的RESTful API和本地部署的Native API，后者可以单机部署，离线处理，适合在内网处理敏感的文本数据，它提供了一系列目前大模型还难以覆盖的NLP处理任务支持，可以有效补充大模型的自然语言处理能力，本书将使用它的分词模型把文档分块，分词是大模型工作的基础。

**1.4.1 安装HanLP 2.1 Native API**

参阅[《HanLP 安装文档》](https://hanlp.hankcs.com/docs/install.html)27，默认安装用的是PyTorch后端。

conda activate pytorch

pip install hanlp

**1.4.2 测试**

具体可以[参阅各种demo](https://github.com/hankcs/HanLP/tree/doc-zh/plugins/hanlp_demo/hanlp_demo/zh)28及[API文档](https://hanlp.hankcs.com/docs/)29。Native API的输入单位为句子，需使用[多语种分句模型](https://github.com/hankcs/HanLP/blob/master/plugins/hanlp_demo/hanlp_demo/sent_split.py)30或[基于规则的分句函数](https://github.com/hankcs/HanLP/blob/master/hanlp/utils/rules.py#L19)31先行分句，这里仅演示单任务模型分词，输入已经是句子。

# 1、加载HanLP

import hanlp

import os, time

# 2、单任务模型， 分词。

tok = hanlp.load(hanlp.pretrained.tok.COARSE\_ELECTRA\_SMALL\_ZH)

toks = tok(['商品和服务。', '阿婆主来到北京立方庭参观自然语义科技公司。'])

print(toks)

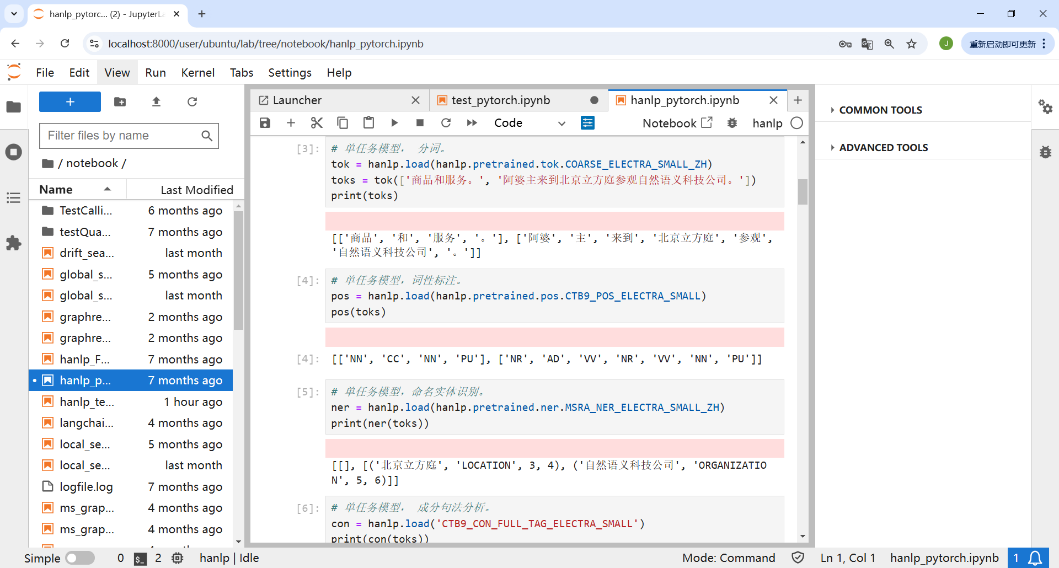


图1-17 HanLP2.1可以在PyTorch后端上顺利执行。

性能监控：

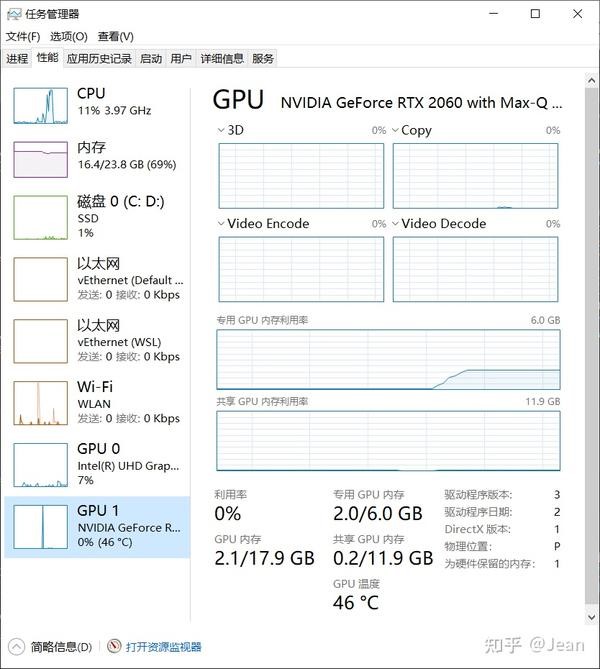


图1-18 Nvidia GPU1及其显存的消耗都不算大

加载上面的各种模型消耗了2G显存，GPU的负荷并不大。通常GPU显存是这类深度学习预训练NLP模型落地应用的瓶颈，而不是GPU的算力，[GeForce RTX 2060](https://www.nvidia.cn/geforce/graphics-cards/rtx-2060/)32有6G显存，当然跟[A100](https://images.nvidia.cn/aem-dam/en-zz/Solutions/data-center/a100/nvidia-a100-datasheet-nvidia-a4-2188504-r5-zhCN.pdf)33（40/80G显存）之类高端GPU无法比，不过也不比[Tesla T4](https://www.nvidia.cn/content/dam/en-zz/zh_cn/Solutions/Data-Center/tesla-t4/nvidia-t4-datasheet-a4-nvidia-772234-r14-lr-cn.pdf)34（数据中心入门级，16G显存，单价2万多）差太多，HanLP跑起来并不吃力，个人学习研究是够用的。当然，各种GPU除了显存的差别，算力、带宽等其它技术指标的差异是非常大的，不能仅比较显存，一分钱一份货，这里只是大致看一下。

**1.5 Neo4j Community安装配置**

Neo4j是目前全球领先的开源图数据库，在关系网络的分析中特别有用,它支持的Neo4j Graph Data Science则是非常强大好用的图算法库。Neo4j GDS最新发布的2.11.0只支持到Neo4j 5.24.0，具体可以参阅[GDS主页](https://github.com/neo4j/graph-data-science)35，所以本书将安装Neo4j Community 5.24.0及GDS 2.11.0。在下一章中，就要用到Neo4j。

**1.5.1 升级JDK**

Noe4j 5.X要求JDK 17以上，可以升级至最新的Open JDK21。

# apt-get install openjdk-21-jdk

设置环境变量。

# vi /etc/profile

增加JAVA的环境变量。

export JAVA\_HOME=/usr/lib/jvm/java-21-openjdk-amd64

export JRE\_HOME=/usr/lib/jvm/java-21-openjdk-amd64

export PATH=$PATH:$JAVA\_HOME/bin

**1.5.2 下载解压Neo4j Community**

微云数聚是Neo4j 中国官方代理、战略合作伙伴及咨询服务提供商，他们提供的Neo4j汉化社区版对Neo4j社区版作了一些扩充。

# wget --no-check-certificate https://we-yun.com/doc/neo4j-chs/5.24.0/neo4j-chs-community-5.24.0-unix.tar.gz

# tar xzvf neo4j-chs-community-5.24.0-unix.tar.gz

# mv neo4j-chs-community-5.24.0-unix /opt/neo4j-chs-community-5.24.0-unix

**1.5.3 启动Neo4j一次，创建服务器的目录结构**

增加Neo4j环境变量。

# vi /etc/profile

增加下面的环境变量。

export NEO4J\_HOME=/opt/neo4j-chs-community-5.24.0-unix

export NEO4J\_CONF=$NEO4J\_HOME/conf

启动Neo4j，第一次启动会创建服务器的目录结构。

(base) root@10-23-6-18:/opt/neo4j-chs-community-5.24.0-unix/bin# ./neo4j start

Directories in use:

……

Starting Neo4j.

Started neo4j (pid:551876). It is available at http://localhost:7474

There may be a short delay until the server is ready.

(base) root@10-23-6-18:/opt/neo4j-chs-community-5.24.0-unix/bin# ./neo4j stop

Stopping Neo4j............ stopped.

**1.5.4 拷贝空白数据库neo4j备份**

社区版不支持多个并发数据库，以后新建数据库时拷贝该备份并重新命名，然后切换至新的数据库即可。

# cd /opt/neo4j-chs-community-5.24.0-unix/data/databases

# cp -R neo4j blank

**1.5.5 修改配置**

打开网络访问地址等，默认只侦听loopback地址 127.0.0.1。这里把从CSV导入数据的许可目录限制在/home/ubuntu/data。

# cd /opt/neo4j-chs-community-5.24.0-unix/conf

# vi neo4j.conf

参考配置：

# 打开网络访问地址

server.default\_listen\_address=0.0.0.0

#启动时打开的数据库，社区版只能在线打开一个数据库，通过改变下面的名字切换。# Change to the database you want

initial.dbms.default\_database=neo4j

# 切换时重建事务日志，否则不能启动，因为数据库变了，不匹配。

# Create a new transaction log when change to a new database

db.recovery.fail\_on\_missing\_files=false

# 配置允许CSV数据文件导入，Linux上必须配置，

# 限制从其它目录导入数据，以堵塞安全漏洞。

# This setting constrains all `LOAD CSV` import files to be under the `import` directory.

server.directories.import=/home/ubuntu/data

# Determines if Cypher will allow using file URLs when loading data using

# `LOAD CSV`. Setting this value to `false` will cause Neo4j to fail `LOAD CSV`

# clauses that load data from the file system.

dbms.security.allow\_csv\_import\_from\_file\_urls=true

#取消对APOC及GDS过程的安全限制。

# A comma separated list of procedures and user defined functions that

# are allowed full access to the database through unsupported/insecure internal APIs.

dbms.security.procedures.unrestricted=apoc.\*,gds.\*,rs.\*,example.\*

# A comma separated list of procedures to be loaded by default.

# Leaving this unconfigured will load all procedures found.

dbms.security.procedures.allowlist=apoc.\*,gds.\*,rs.\*,example.\*

#打开用户验证，注释掉后默认是打开的。

# Whether requests to Neo4j are authenticated.

# To disable authentication, uncomment this line

#dbms.security.auth\_enabled=false

**1.5.6 重启Neo4j**

访问http://localhost:7474作为测试。默认用户名/口令是 neo4j/neo4j，第一次登录要改密码。

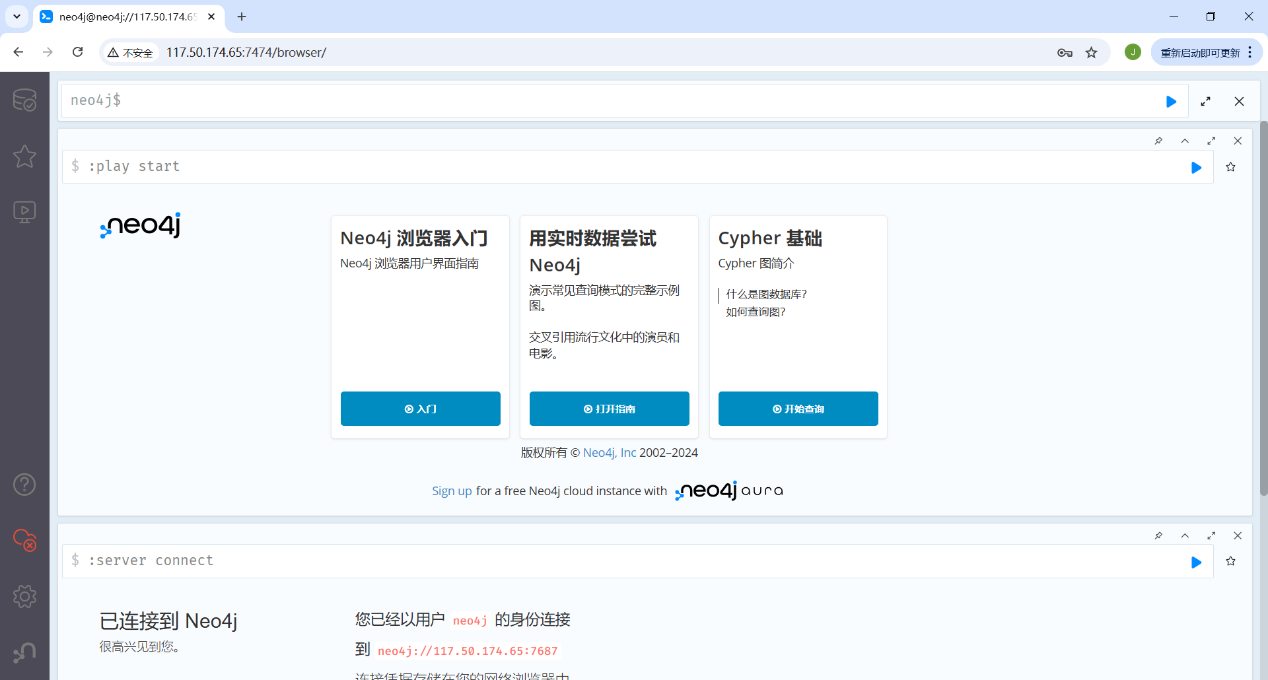


图1-19 登录Noe4j Community Browser

**1.5.7 安装APOC、GDS等plug-in**

默认安装已经有APOC 5.24.0，只需下载GDS 2.11.0。

# cd /opt/neo4j-chs-community-5.24.0-unix/plugins

# wget --no-check-certificate https://we-yun.com/doc/neo4j-gds/2.11.0/neo4j-graph-data-science-2.11.0.jar

下载好后可以检查一下。

(base) root@10-23-6-18:/opt/neo4j-chs-community-5.24.0-unix/plugins# ls

README.txt apoc-5.24.0-core.jar neo4j-graph-data-science-2.11.0.jar rs-5.11.0.jar rs-license.Trial.20241231.txt

重启Neo4j，在浏览器中执行下面的Cypher语句测试GDS及APOC的安装。

return apoc.version()

return gds.version()

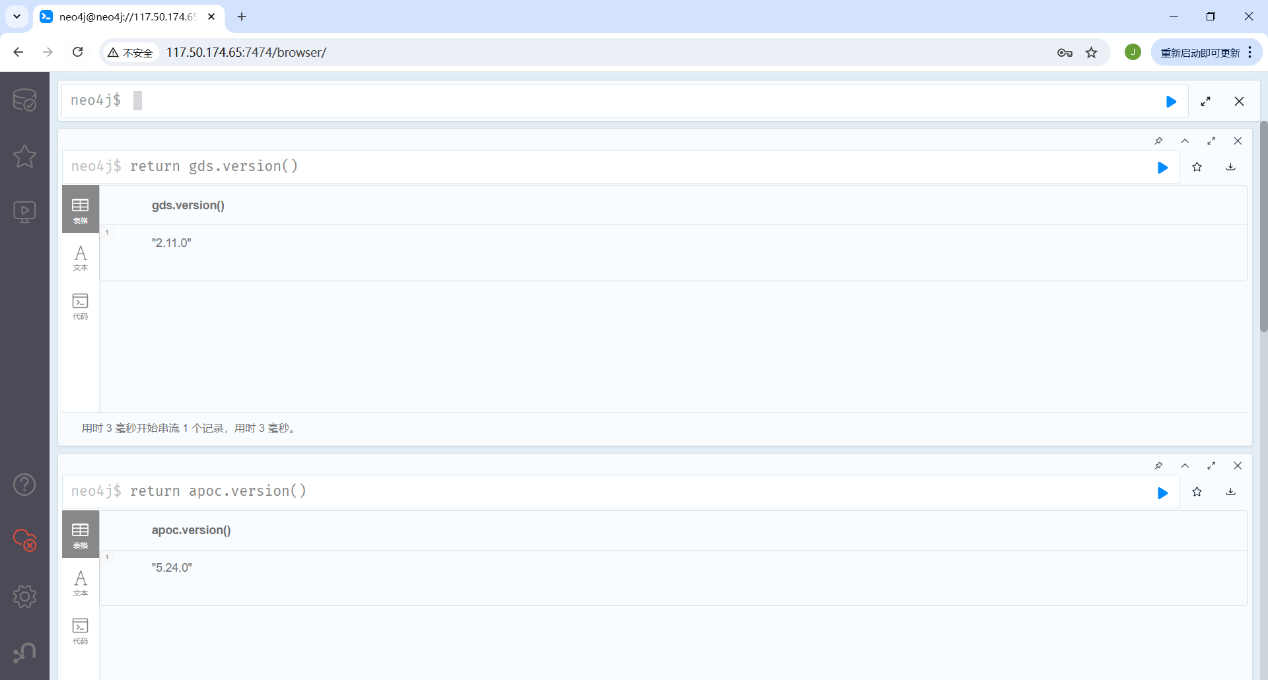


图1-20 测试APOC及GDS的安装

**1.5.8 配置开机启动Neo4j**

参考Neo4j Community 的文档 [installation-linux-tarball-service](https://neo4j.com/docs/operations-manual/current/installation/linux/tarball/#installation-linux-tarball-service)36，从tar文件安装Neo4j时，可以创建一个系统服务文件，让系统重启时自动启动Neo4j。

# cd /lib/systemd/system

# vi neo4j.service

内容如下：

[Unit]

Description=Neo4j Graph Database

After=network-online.target

Wants=network-online.target

[Service]

User=root

Environment="JAVA\_HOME=/usr/lib/jvm/java-21-openjdk-amd64"

Environment="JRE\_HOME=/usr/lib/jvm/java-21-openjdk-amd64"

Environment="PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/jvm/java-21-openjdk-amd64/bin"

Environment="NEO4J\_HOME=/opt/neo4j-chs-community-5.24.0-unix"

Environment="NEO4J\_CONF=/opt/neo4j-chs-community-5.24.0-unix/conf"

ExecStart=/opt/neo4j-chs-community-5.24.0-unix/bin/neo4j console

[Install]

WantedBy=multi-user.target

然后激活它。

systemctl daemon-reload

systemctl enable neo4j

现在系统重启后它会自动启动，也可以用下面的命令来管理它。

# systemctl start/stop/restart/status neo4j

到这里，WSL2 Ubuntu 22上的Python GPU Linux Server深度学习环境就初步搭好。