

深度学习环境配置与使用方法教程

张庚

上海工程技术大学

电子电气工程学院 7719 实验室

版本: version 1.1

日期: 2023 年 12 月 24 日

目录

| | | |
|----------|--------------------------------------|-----------|
| 1 | 前言 | 2 |
| 1.1 | 操作系统的选择与安装过程 | 2 |
| 1.2 | Linux 系统安装过程 | 2 |
| 1.2.1 | 安装系统之后的操作 | 4 |
| 1.2.2 | 安装 python3 | 4 |
| 1.3 | Windows10 安装过程 | 5 |
| 2 | SSH 远程登录等配置 | 5 |
| 2.1 | SSH 远程登录 | 5 |
| 2.2 | STFP 文件传输工具 (选看) | 6 |
| 2.3 | XRDP 桌面远程登录 (选看) | 8 |
| 2.4 | Samba 文件共享 (选看) | 9 |
| 3 | 深度学习环境配置 | 9 |
| 3.1 | NVIDIA 显卡深度学习环境配置 | 9 |
| 3.1.1 | 从源上安装显卡驱动 | 10 |
| 3.1.2 | 官网驱动安装 | 10 |
| 3.1.3 | 检查驱动是否安装成功 | 11 |
| 3.1.4 | CUDA 安装 | 11 |
| 3.1.5 | CUDNN 安装 | 14 |
| 3.2 | AMD 显卡深度学习环境配置 (选看) | 14 |
| 3.3 | 华为昇腾 Altas300 系列 AI 加速卡深度学习环境配置 (选看) | 15 |
| 3.4 | 寒武纪 S370 系列 AI 加速卡深度学习环境配置 (选看) | 15 |
| 3.5 | 燧原 T20、T21 加速卡深度学习环境配置 (选看) | 15 |
| 3.6 | 摩尔线程 S70、S80AI 加速卡深度学习环境配置 (选看) | 15 |
| 4 | 深度学习框架搭建环境安装 | 16 |
| 4.1 | Python 深度学习环境配置 | 16 |
| 4.1.1 | Miniconda3 安装配置 | 16 |
| 4.1.2 | Python 虚拟环境的设置 | 16 |
| 4.1.3 | Pytorch 环境安装 | 18 |

| | | |
|----------|---------------------------|-----------|
| 4.1.4 | Tensorflow 环境安装 | 19 |
| 4.1.5 | Jax 环境安装 | 21 |
| 4.1.6 | 其他一些包环境 | 21 |
| 4.1.7 | Paddle 环境安装（选看） | 21 |
| 4.1.8 | MindSpore 环境安装（选看） | 21 |
| 4.1.9 | OneFlow 环境安装（选看） | 22 |
| 4.1.10 | MXNet 环境安装（选看） | 22 |
| 4.1.11 | Chainer 环境安装（选看） | 23 |
| 4.1.12 | Theano 环境安装（选看） | 23 |
| 4.2 | Julia 深度学习环境配置（选看） | 25 |
| 4.2.1 | Julia 环境安装 | 26 |
| 4.2.2 | Flux ML 安装 | 26 |
| 4.2.3 | 其他一些包环境 | 26 |
| 4.3 | Rust 深度学习环境配置（选看） | 26 |
| 4.3.1 | Rust 环境安装 | 26 |
| 4.3.2 | Candle 深度学习环境安装 | 26 |
| 4.3.3 | Burn 深度学习环境安装 | 26 |
| 4.3.4 | Tch-rs 深度学习环境安装 | 26 |
| 4.3.5 | Neuronika 深度学习环境 | 26 |
| 4.3.6 | 其他一些包环境 | 26 |
| 4.4 | Go 深度学习环境配置（选看） | 26 |
| 4.4.1 | Go 语言环境安装 | 26 |
| 4.4.2 | Gorgonia 深度学习环境安装 | 26 |
| 4.5 | lua 深度学习环境配置（选看） | 26 |
| 4.5.1 | torch7 深度学习环境安装 | 26 |
| 4.5.2 | 其他一些包环境 | 28 |
| 4.6 | Java 深度学习环境配置（选看） | 28 |
| 4.6.1 | Java 以及 Maven 环境安装 | 28 |
| 4.6.2 | TensorFlow 在 Android 上的使用 | 28 |
| 4.6.3 | Pytorch 在 Android 上的使用 | 28 |
| 4.6.4 | DeepLearning4j 深度学习环境安装 | 28 |
| 4.6.5 | 其他一些环境 | 28 |
| 5 | 远程登录工具使用 | 28 |
| 5.1 | 公网远程连接桌面工具 | 28 |
| 5.2 | 文件传输工具 | 30 |
| 5.3 | VSCode 配置 SSH 远程连接 | 31 |
| 5.4 | PyCharm 配置 SSH 远程连接 | 33 |
| 5.5 | 内网穿透（选看） | 35 |
| 6 | Docker 使用方法 | 35 |
| 6.1 | Docker 安装 | 36 |
| 6.2 | Docker 使用 | 38 |

| | | |
|----------|----------------------------------|-----------|
| 7 | WSL2 使用方法 | 42 |
| 7.1 | WSL2 安装 | 42 |
| 7.2 | WSL2 使用方法 | 42 |
| 8 | LaTeX 安装教程 | 43 |
| 8.1 | LaTeX 安装与配置 | 43 |
| 8.2 | 在线 LaTeX 编辑器 | 43 |
| 8.3 | Typst 安装与配置（选看） | 43 |
| 9 | 大模型部署方法 | 43 |
| 9.1 | 普通 CPU 部署方法（全平台） | 43 |
| 9.2 | 使用 Intel 处理器部署（Intel 10 代以上 CPU） | 43 |
| 9.3 | 使用 NVIDIA 处理器部署（显存至少 16GB 以上） | 43 |
| 9.4 | 使用华为 Atlas300I 推理卡部署（选看） | 43 |
| 9.5 | 使用摩尔线程 S80 部署（选看） | 43 |

1 前言

本文档介绍的是实验室对已有的硬件设备进行系统、软件安装和使用的教程，根据日常实验室使用过程中出现的问题以及使用习惯，以及本人在实验经验、经历总结出以下的系统使用过程，以方便组内的同学使用。

1.1 操作系统的选择与安装过程

Linux 操作系统有很多选择，一般分为以下几个派别的：Debian 系（例如 Debian、Ubuntu、Deepin、Kali 等）、Fedora 系（RedHat、Fedora、CentOS 等）、Arch 系、Gentoo 系、SUSE 系等。介于生态、可维护性、开源免费性考虑，我们不选择基于滚动版本的操作系统（例如 ArchLinux、CentOS Stream 等），商业版本操作系统（RedHat、Oracle Linux 等等）、自配置源码编译系统（Gentoo、Funtoo 甚至 LFS 操作系统等等）、包括其他系列的较难使用的操作系统（例如 NixOS 等），我们首要考虑的是兼容性、广泛性和实用性，所以在日常实验中，大家可以选择以下系列的操作系统：

- 基于 Debian 的系统：Linux Mint、Ubuntu、Ubuntu 衍生版；
- 基于 Fedora 的系统：Fedora、Rocky、Alma；

以上的操作系统均具有较好的开源性、稳定性，并且社区教程和资源比较多，所以选择以上的操作系统作为自己的开发系统。在大模型部署的容器中，有些教程选择较为轻量级的 Alpine Linux 作为 Docker 容器内部的操作系统环境，这也是可以的，目的就是尽一切可能简化 Docker 容器内的操作系统配置文件大小。Ubuntu 建议选择 16.04,18.04,20.04,22.04 或者 24.04 这几个版本。在本服务器上我们选择了 Ubuntu 衍生版的操作系统，国产 Ubuntu Kylin22.04，对 Windows 应用具有较好的兼容性，Windows 味道的桌面，特别是对于小白友好。

Windows 操作系统选择 Windows10 或者是 Windows Server 2019 服务器版操作系统。

1.2 Linux 系统安装过程

Debian 系列操作系统首选 Ubuntu22.04，也可以选择基于这个系列的衍生版，在官网或者各大镜像站下载，例如清华大学镜像站、中科大镜像站、阿里镜像站等等，例如清华大学镜像站下载 Ubuntu 22.04 操作系统，注意选择正确的架构，基于自己笔记本和服务器的操作系统一般都是 x86 系列的，这里选择 64 位操作的 x86 操作系统 amd64 架构，如图所示：

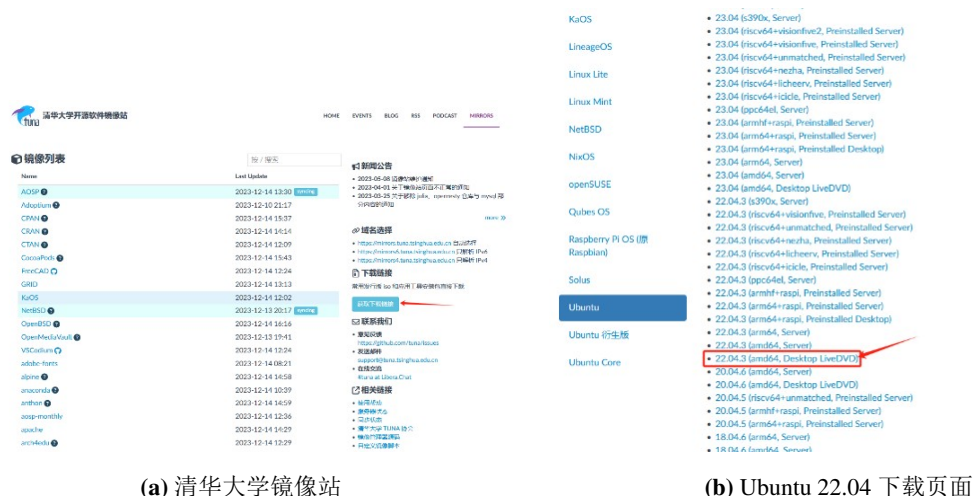


图 1: Ubuntu 下载页面

在 Windows 下使用 UltraISO 工具或者 BalenaEtcher 镜像刻录工具（如图所示），将其写入到 U 盘中，得到系统的启动盘，U 盘刻录需要花费大概 10-20 分钟或者更少的时间。

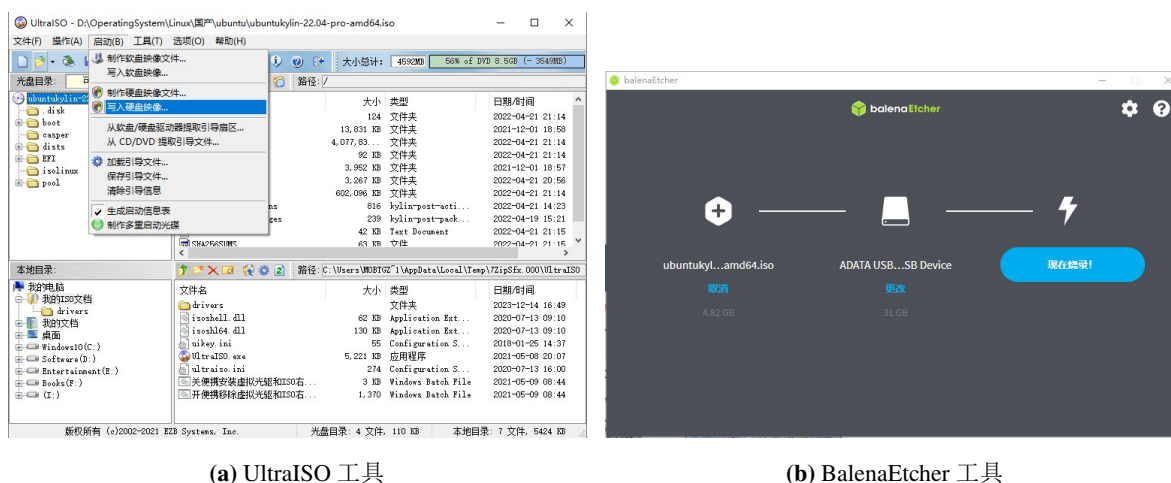


图 2: 刻录工具介绍

当然也可以选择功能强大的 Ventoy 工具，如图3所示安装，将 Ventoy 工具安装在 U 盘之后，直接将下载好 ISO 文件丢到 Ventoy 分区当中即可，具体安装步骤可以参考 Ventoy 官方网站。

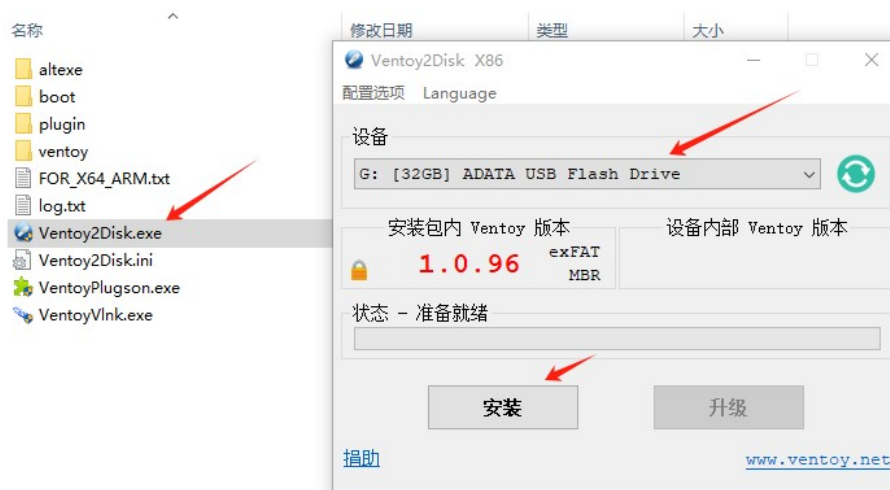


图 3: Ventoy 工具安装

刻录完成之后，将 U 盘插入到电脑上，开机按 Del 按键（具体按照主板型号）进入 BIOS 界面，关闭 Secure Boot 选项，并将启动方式改为 UEFI 启动方式，第一启动项改为 U 盘启动，重启工作站即可进入到 U 盘的系统。当然有些设备按 F12 或 ESC 按键（具体参照主板型号）可以直接进入 U 盘系统。

进入 U 盘中的 Ubuntu 预安装系统，按照步骤即可。需要注意的是，选择英文作为系统默认的语言，免去日后开发过程中的一些不必要的麻烦；选择最小化安装的方式，这样可以出去一些不必要的软件；安装过程中选择将一些软件包更新 skip 掉，或者断开网线进行安装操作系统，这些软件包更新实际上在系统安装完成之后联网更新也是可以的；交换分区按照实际大小来进行设置，也可以不进行设置处理。如若不设置的话，系统会在根据目录下设置一个相当于交换分区的文件用于和内存之间的交换。一般情况下，内存足够大不需要设置交换分区的，交换分区相当于在 Windows 系统中的虚拟内存大小的设置。

1.2.1 安装系统之后的操作

首先备份操作系统原来的源

```
cp /etc/apt/sources.list /etc/apt/sources.list.bak
```

然后将源的内容更换为国内镜像源，这里我们可以选择中科大镜像源、清华镜像源或者是阿里云镜像源，我们这里选择中科大镜像源：

```
sudo nano /etc/apt/sources.list
```

更改为以下内容

```
# 默认注释了源码仓库，如有需要可自行取消注释
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy main restricted universe multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-security main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-security main restricted universe
multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-updates main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-updates main restricted universe
multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-backports main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-backports main restricted universe
multiverse

# 预发布软件源，不建议启用
# deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-proposed main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-proposed main restricted universe
multiverse
```

使用国内镜像站的源作为系统源的目的是，在下载源软件能够更快更好，更新系统也较为方便，以防止网络延迟。然后更新源：

```
sudo apt update # 更新源
sudo apt upgrade # 升级软件包
```

1.2.2 安装 python3

注意这里安装的是系统中 python3 和 pip 环境，与后面 conda 中 python 环境是不一致的，若不需要系统环境下的 python 环境可以跳过这个步骤。

```
sudo apt install python3 python3-pip python3-dev python3-virtualenv
```

更改 pip 配置，将安装好的 python 中 pip 源的配置更换为以下的设置，显得 pip 安装速度会更快。下面是用户文件夹配置 pip 源

```
cd ~
mkdir .pip
# 编辑文件 pip.conf
sudo nano ~/.pip/pip.conf
```

更换为以下内容

```
[global]
index-url = https://mirrors.aliyun.com/pypi/simple/
[install]
trusted-host=mirrors.aliyun.com
```

下面是 root 用户配置 pip 源

```
sudo mkdir /root/.pip
# 编辑文件pip.conf
sudo nano /root/.pip/pip.conf
```

创建软链接（非必须）

```
# 把原来的python软链接删掉
sudo rm /usr/bin/python
# 新建一个软链接
sudo ln -s /usr/bin/python3 /usr/bin/python
sudo ln -s /usr/bin/pip3 /usr/bin/pip
```

当然上述创建软链接也是非必须的，在从系统源上安装 python3 时候就已经将其软链接建立好了。

1.3 Windows10 安装过程

在 MSDN 上可以下载 Windows10 或者 Windows Server2019 原版镜像，同时再下载一个 WinPE ISO 镜像，并将 WinPE 用 UltraISO 或者 BalenaEtcher 工具将 ISO 文件刻录到 U 盘上，然后进入 U 盘系统之后，使用 WinNTSetup 工具对操作系统进行安装。由于在 Windows 系统安装过程极为简便这里就不再过多进行叙述。

2 SSH 远程登录等配置

2.1 SSH 远程登录

一般情况下，Ubuntu 系统中会默认安装好 SSH 环境，如果没有 SSH，则通过以下的方式进行安装

```
sudo apt install openssh-server # 安装openssh服务器端
sudo apt install openssh-client # 安装openssh客户端
```

可以用以下的命令启动 ssh，或者是开机启动 SSH：

```
sudo /etc/init.d/sshd restart #重启SSH服务
sudo /etc/init.d/sshd stop #停止SSH服务
sudo /etc/init.d/sshd start #开启SSH服务
```

当然现在大多数 Linux 为 systemd 初始化的操作系统，openRC 方式较少，也可以通过以下的方式进行重启服务：

```
sudo systemctl restart ssh#重启SSH服务
sudo systemctl stop ssh #停止SSH服务
sudo systemctl start ssh #开启SSH服务
sudo systemctl enable ssh #开机启动SSH服务
sudo systemctl disable ssh #禁用SSH服务
```

可以通过以下的命令查看 SSH 服务是否启动

```
sudo systemctl status ssh # 查看ssh服务状态
```

使用 SSH 登录用以下的命令即可以登录：

```
ssh -p port_number username@ipaddress
```

其中-p 表示的是指向的端口号名称，默认为 22；username 为远程主机的用户名，ipaddress 为远程主机的 IP 地址。连接上之后，就可以当做 Linux 用户主机使用。

2.2 SFTP 文件传输工具（选看）

在计算机领域，SSH 文件传输协议（英语：SSH File Transfer Protocol，也称 Secure File Transfer Protocol，中文：安全文件传送协议，英文：Secure FTP 或字母缩写：SFTP）是一数据流连线，提供文件存取、传输和管理功能的网络传输协议。与 FTP 协议相比，在几乎所有情况下，SFTP 都比 FTP 更可靠，因为它具有潜在的安全功能并且能够搭载 SSH 连接。FTP 是一种不安全的协议，只能在有限的情况下或您信任的网络上使用。

一般远程客户端配置好了 SSH，就可以使用 SFTP 协议传输文件。如果使用了自定义的 SSH 端口号（默认为 22），打开 SFTP 会话时可以设定端口号

```
sftp -oPort=custom_port username@ ipaddress
```

几个常见的使用方法如下

- **获取帮助：**可以通过以下的命令查看 SFTP 参数使用方法

```
sftp> help  
或  
sftp> ?
```

输出结果如下所示

```
sftp> help  
Available commands:  
bye                               Quit sftp  
cd path                           Change remote directory to 'path'  
chgrp [-h] grp path               Change group of file 'path' to 'grp'  
chmod [-h] mode path              Change permissions of file 'path' to 'mode'  
chown [-h] own path               Change owner of file 'path' to 'own'  
df [-hi] [path]                   Display statistics for current directory or  
                                  filesystem containing 'path'  
exit                               Quit sftp  
get [-afpR] remote [local]        Download file  
help                               Display this help text  
lcd path                           Change local directory to 'path'  
lls [ls-options] [path]            Display local directory listing  
lmkdir path                       Create local directory  
ln [-s] oldpath newpath            Link remote file (-s for symlink)  
lpwd                               Print local working directory  
ls [-lafhlNrSt] [path]             Display remote directory listing  
lumask umask                       Set local umask to 'umask'  
mkdir path                         Create remote directory  
progress                           Toggle display of progress meter  
put [-afpR] local [remote]         Upload file  
pwd                                Display remote working directory
```


| | |
|-----------------------------|----------------------------------|
| quit | Quit sftp |
| reget [-fpR] remote [local] | Resume download file |
| rename oldpath newpath | Rename remote file |
| reput [-fpR] local [remote] | Resume upload file |
| rm path | Delete remote file |
| rmdir path | Remove remote directory |
| symlink oldpath newpath | Symlink remote file |
| version | Show SFTP version |
| !command | Execute 'command' in local shell |
| ! | Escape to local shell |
| ? | Synonym for help |

- **遍历远程文件系统：**我们可以使用一些功能类似于 shell 命令的命令来浏览远程系统的文件层次结构。但是 SFTP 并不完全是 shell 的使用方法，从命令上就可以看出并不一样。

首先，通过找出我们当前在远程系统上的哪个目录来定位自己

```
sftp>pwd
Remote working directory: /home/sesame
```

查看远程系统当前目录的内容

```
sftp>ls
Cognata

    Desktop
Documents

    Downloads
nohup.out

pangweisong

    qiancj
set_iptables.sh

opt
```

SFTP 其实也实现了一些更重要的可选标志，例如将 -la 添加到 ls 以查看更多文件元数据和权限：

```
sftp>ls -la
outputdrwxr-xr-x    5 remote_money  remote_money      4096 Aug 13 15:11 .
drwxr-xr-x    3 root      root      4096 Aug 13 15:02 ..
-rw-----    1 remote_money  remote_money      5 Aug 13 15:04 .bash_history
-rw-r--r--    1 remote_money  remote_money     220 Aug 13 15:02 .bash_logout
-rw-r--r--    1 remote_money  remote_money    3486 Aug 13 15:02 .bashrc
drwx-----    2 remote_money  remote_money     4096 Aug 13 15:04 .cache
-rw-r--r--    1 remote_money  remote_money     675 Aug 13 15:02 .profile
. . .
```

进入到其他目录

```
sftp>cd testDirectory
sftp> cd qiancj
```

- **访问本地的文件系统：**通过在命令前面加上一个 l (local) 来将命令指向本地文件系统，例如 pwd, cd 等等。

```
sftp>lpwd
```

```
Local working directory: /home/qiancj  
sftp> lcd /home/qiancj/codes/sanjie
```

- 从远程服务器下载文件到本地服务器：从远程系统上下载文件到本地

```
sftp> get remoteFile  
OutputFetching /home/remote_money/remoteFile to remoteFile  
/home/remote_money/remoteFile      100%   37KB  36.8KB/s   00:01
```

下载到本地并重命名

```
sftp> get remoteFile localFile
```

get 命令可以增加可选项，比如要拷贝目录及其下所有文件，可以使用-r 选项

```
sftp> get -r someDirectory
```

使用 -P 或 -p 标志维护适当的权限和访问时间

```
sftp> get -Pr someDirectory
```

- 传输本地文件到远程服务器：使用 put 命令:

```
sftp> put localFile
```

传输到远程服务器并重命名

```
sftp> put localFile remoteFile
```

传输目录及其下所有文件

```
put -r localDirectory
```

2.3 XRDP 桌面远程登录（选看）

xrdp 是一个微软远程桌面协议（RDP）的开源实现，允许我们通过 Xorg 协议的图形界面远程控制操作系统。这里使用 RDP 而不是 VNC 作为远程桌面，这是因为 Windows 自带的远程桌面连接软件就可以连接很方便，同时也可以直接实现在主机和远程主机之间的复制粘贴等等操作，比较方便。

安装步骤如下所示：

```
sudo apt install xrdp # 安装xrdp
```

安装完成之后 xrdp 就会自动启动，可以使用以下的命令查看它的状态信息

```
sudo systemctl status xrdp
```

设置为开机启动

```
sudo systemctl enable xrdp
```

默认情况下，xrdp 使用/etc/ssl/private/ssl-cert-snakeoil.key，它仅仅对 ssl-cert 用户组成语可读，所以需要运行下面的命令，将 xrdp 用户添加到这个用户组：

```
sudo adduser xrdp ssl-cert  
sudo systemctl restart xrdp
```

2.4 Samba 文件共享（选看）

我们这里选择的的是 SAMBA 文件共享服务进行安装，可以通过建立局域网 SAMBA 服务来实现：安装 SAMBA 服务：

```
sudo apt-get install samba samba-common-bin
```

配置/etc/samba/smb.conf 文件

```
suod nano /etc/samba/smb.conf
```

在文件的最后添加以下内容

```
# 共享文件夹显示的名称
[home]
# 说明信息
comment = WorkStation Storage
# 可以访问的用户
valid users = mobtgzhang,root
# 共享文件的路径
path = /home/mobtgzhang/
# 可被其他人看到资源名称（非内容）
browseable = yes
# 可写
writable = yes
# 新建文件的权限为 664
create mask = 0664
# 新建目录的权限为 775
directory mask = 0775
```

可以把配置文件中你不需要的分享名称删除，例如 [homes], [printers] 等。运行这个命令测试一下配置文件是否有错误，根据提示做相应修改：testparm

添加登陆账户并创建密码，必须是 linux 已存在的用户：

```
sudo smbpasswd -a mobtgzhang
```

重启 samba 服务

```
sudo systemctl restart smbd
```

3 深度学习环境配置

深度学习环境的配置首要的问题是深度学习显卡加速驱动的安装，对于 NVIDIA 显卡驱动来说是 CUDA 深度学习加速库，对于 AMD 显卡来说是 ROCm 深度学习加速库，其他的计算卡通过对应的教程可以进行安装。然后在这些驱动和加速库的基础之上可以进一步进行深度学习环境的配置和安装，这里列举了几种显卡/计算卡的配置方法以供参考。

3.1 NVIDIA 显卡深度学习环境配置

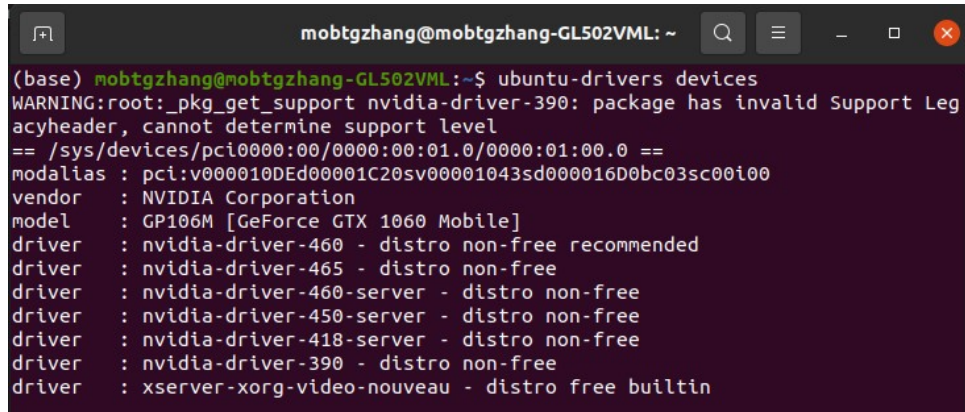
最为常见的深度学习显卡为 NVIDIA 显卡，这里一共有两种方法安装 NVIDIA 驱动，下面分为两个章节介绍这两种方法。

3.1.1 从源上安装显卡驱动

首先我们列举出可用显卡信息

```
sudo apt-get update #更新一下源
ubuntu-drivers devices
```

它会列举出显卡驱动的信息，选择你想要安装的显卡驱动信息：



```
(base) mobtgzhang@mobtgzhang-GL502VML:~$ ubuntu-drivers devices
WARNING:root:_pkg_get_support nvidia-driver-390: package has invalid Support Leg
acyheader, cannot determine support level
== /sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0 ==
modalias : pci:v000010DEd00001C20sv00001043sd000016D0bc03sc00i00
vendor    : NVIDIA Corporation
model     : GP106M [GeForce GTX 1060 Mobile]
driver    : nvidia-driver-460 - distro non-free recommended
driver    : nvidia-driver-465 - distro non-free
driver    : nvidia-driver-460-server - distro non-free
driver    : nvidia-driver-450-server - distro non-free
driver    : nvidia-driver-418-server - distro non-free
driver    : nvidia-driver-390 - distro non-free
driver    : xserver-xorg-video-nouveau - distro free builtin
```

图 4: Ubuntu 系统下显示的设备信息

例如安装以下显卡驱动信息：

```
sudo apt-get install nvidia-driver-470
```

或者安装对于 ubuntu 系统的显卡驱动，自动安装驱动程序：

```
sudo ubuntu-drivers autoinstall
```

重启电脑，即可以安装成功。

3.1.2 官网驱动安装

(1) 禁用 nouveau 驱动：

nouveau，是一个自由及开放源代码显卡驱动程序，是为 Nvidia 的显示卡所编写，也可用于属于系统芯片的 NVIDIA Tegra 系列，此驱动程序是由一群独立的软件工程师所编写。但是 nouveau 开源驱动基本上是不能正常使用的，性能极低。

首先编辑文件 blacklist.conf：

```
sudo nano /etc/modprobe.d/blacklist.conf
```

移动光标，在最后一行添加以下代码：

```
blacklist nouveau
options nouveau modeset=0
```

然后 ctrl + O 保存文件。

(2) 更新内核文件：

注意备份文件，还需要注意自己系统的内核文件，不同版本的 Linux 内核更新不一样

```
# 备份内核文件
sudo cp /boot/initrd.img-$(uname -r) /boot/initrd.img-$(uname -r).bak
# 更新内核文件
```

```

sudo update-initramfs -u
# 更新GRUB
sudo update-grub2 # 或者是 sudo grub2-mkconfig -o /boot/grub/grub.cfg
# 重启电脑
sudo shutdown -r now

```

- (3) 进入 BIOS 设置，关闭 Secure Boot 设置：进入 Secure Boot Menu，将 Secure Boot Control 设置为 disabled。由于 Secure boot 带来的一些内核签名验证问题，比较麻烦，所以为简易起见关闭了 Secure boot 设置。

NVIDIA 驱动程序下载

在下方的下拉列表中进行选择，针对您的 NVIDIA 产品确定合适的驱动。

| | | |
|-------|-------------------------------|-----|
| 产品类型: | GeForce | ▼ |
| 产品系列: | GeForce 10 Series (Notebooks) | ▼ |
| 产品家族: | GeForce GTX 1060 | ▼ |
| 操作系统: | Linux 64-bit | ▼ |
| 下载类型: | 生产分支 | ▼ ? |
| 语言: | Chinese (Simplified) | ▼ |

搜索

图 5: NVIDIA 官方网址

从官网下载驱动程序。以我自己的笔记本为例，选择对应的版本即可。
然后进行安装

```

sudo bash NVIDIA-Linux-x86_64-{你的NVIDIA驱动版本号}.run --no-x-check --no-nouveau-check --no-opengl-files

```

参数的含义如下：

- **--no-x-check**: 不检查 X 服务是否运行；
- **--no-nouveau-check**: 不检查 nouveau 驱动是否运行；
- **--no-opengl-files**: 只安装驱动文件，不安装 OpenGL 文件

这样再 reboot，就不会出现循环登录的问题。

3.1.3 检查驱动是否安装成功

检查 NVIDIA 显卡驱动程序是否安装成功，可启动 nvidia-smi 进行查看（如图6所示）：
也可以启动 nvidia-settings 进行查看（如图7所示）：

3.1.4 CUDA 安装

因为编译 cuda 程序需要 GCC 版本低一点的编译器，否则会有以下的一些错误：

```

Error: unsupported compiler: 8.2.0. Use --override to override this check.

```

对于 Ubuntu22.04 来说，源上最低版本的 GCC 工具为 GCC-7.5.0，所以我们这里使用 GCC7 版本的编译器，安装对应的 GCC 编译器。

```

sudo apt install gcc-7 g++-7

```

```
suesai7719@suesai7719-PC: ~/Desktop
File Edit View Search Terminal Help
(base) suesai7719@suesai7719-PC:~/Desktop$ nvidia-smi
Sun Dec 24 15:48:34 2023
+-----+
| NVIDIA-SMI 525.147.05   Driver Version: 525.147.05   CUDA Version: 12.0   |
+-----+-----+-----+-----+-----+-----+
| GPU  Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|====+=====+=====+=====+=====+=====+
|  0  NVIDIA GeForce ...   Off   | 00000000:01:00.0 On  |           7%      N/A |
|  0%   56C    P8     41W / 350W | 251MiB / 12288MiB |           Default    |
|                               |                      | MIG M.     N/A      |
+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                                                       GPU Memory |
|  GPU   GI    CI          PID    Type   Process name                          Usage  |
|====+=====+=====+=====+=====+=====+
|  0   N/A   N/A         1444      G   /usr/lib/xorg/Xorg                     171MiB |
|  0   N/A   N/A         4485      G   ..._31673.log --shared-files           4MiB |
|  0   N/A   N/A         5914      G   /usr/bin/ukui-kwin_x11                 66MiB |
|  0   N/A   N/A         62387     G   .../bin/kylin-kmre-appstream           6MiB |
+-----+-----+-----+-----+-----+-----+
+-----+
```

图 6: nvidia-smi 查看显卡驱动信息

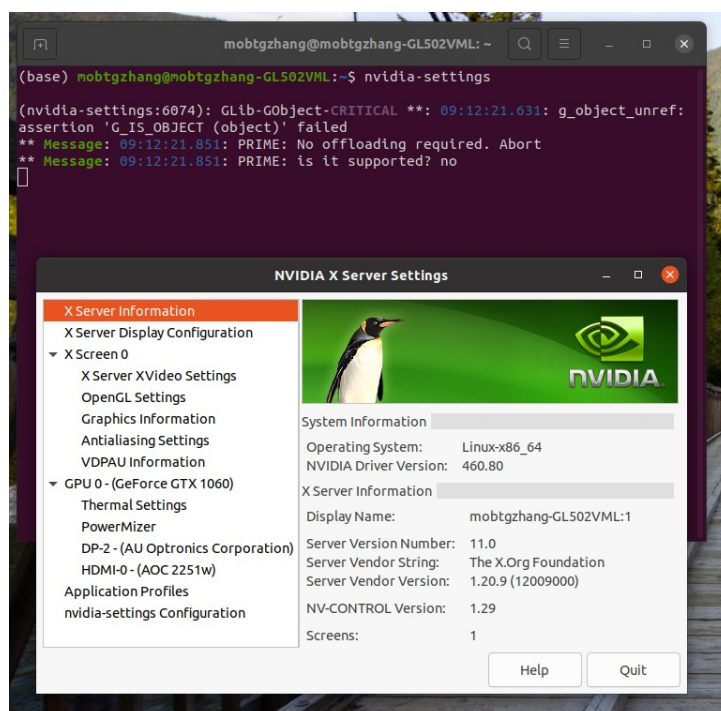


图 7: nvidia-settings 设置查看显卡驱动信息

查看是否安装成功

```
cd /usr/bin
ls -l gcc* # 显示gcc编译器
ls -l g++* # 显示g++编译器
gcc -v # 查看gcc是否安装成功
g++ -v # 查看g++是否安装成功
```

如果查看 gcc, g++ 命令当前的版本并非 gcc-7 和 g++-7, 那么就需要创建软链接

```
sudo mv /usr/bin/gcc /usr/bin/gcc.bak
sudo ln -s /usr/bin/gcc-7 /usr/bin/gcc
sudo mv /usr/bin/g++ /usr/bin/g++.bak
sudo ln -s /usr/bin/g++-7 /usr/bin/g++
```

对于 Ubuntu22.04 来说, 源上最低版本的 GCC 工具为 GCC-7.5.0, 所以我们这里使用 GCC7 版本的编译器, 安装对应的 GCC 编译器。

下载 CUDA run 文件见官网, 为兼容方便我们这里安装了 cuda-11.2。

```
wget https://developer.download.nvidia.com/compute/cuda/11.2.0/local_installers/cuda_11.2.0_460.27.04_linux.run
sudo sh cuda_11.2.0_460.27.04_linux.run
```

注意!!!! 安装时候会提示安装它提示安装的显卡驱动, 一定要选择 no, 否则会出现显卡驱动冲突, 无法进入桌面环境。

若在安装过程中缺少一些库文件, 例如 libGL.so 等等文件, 可能会出现如下错误:

```
Missing recommended library: libGLU.so
Missing recommended library: libX11.so
Missing recommended library: libXi.so
Missing recommended library: libXmu.so
```

可以通过以下的命令安装对应的库文件:

```
sudo apt-get install libglu1-mesa libxi-dev libxmu-dev libglu1-mesa-dev
```

配置 CUDA 对应的环境变量, 编辑文件 bashrc

```
sudo nano ~/.bashrc
```

注. ~/.bashrc 文件普通用户变量环境文件, 如果是 root 用户, 需要修改为/root/.bashrc 文件。另外, 不要改/etc/profile 文件, 这里是整个系统环境变量文件, 如果非得需要添加系统环境变量, 可以直接在文件夹/etc/profile.d/下添加对应的.sh 文件, 这样就可以添加系统环境变量了。

并添加以下内容

```
export PATH=/usr/local/cuda-11.2/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda-11.2/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

注. 这里的 CUDA 安装路径为/usr/local/cuda-11.2, 如果安装的是其他版本的 CUDA, 需要修改为对应的版本号。

使得文件生效

```
source ~/.bashrc
```

查看 cuda 是否安装成功以及是否添加到环境变量当中


```
nvcc -V
```

显示以下的信息

```
(base) suesai7719@suesai7719-PC:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_14_21:12:58_PST_2021
Cuda compilation tools, release 11.2, V11.2.152
Build cuda_11.2.r11.2/compiler.29618528_0
(base) suesai7719@suesai7719-PC:~$
```

图 8: NVCC 命令显示的内容

另外一种方法是直接在源上安装 cuda 即可, 对于 Ubuntu22.04 来说, 源上的 CUDA 版本为 11.5 版本。

```
sudo apt install nvidia-cuda-toolkit
```

3.1.5 CUDNN 安装

官网下载 cudnn 文件, 并解压, 注意对应版本的 cuda 的 cudnn, 这里是 cuda11.2 对应的文件

```
tar -xzf cudnn-11.2-linux-x64-v8.4.15.tgz
sudo cp cuda/include/cudnn* /usr/local/cuda/include
sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

查看 cuda 是否安装成功以及是否添加到环境变量当中, cudnn 是否安装成功

```
cat /usr/local/cuda/version.json
cat /usr/local/cuda/include/cudnn_version.h | grep CUDNN_MAJOR -A 2
```

至此, CUDNN 即可安装到系统当中。当然也可以从源上安装 CUDNN, Ubuntu22.04 源上的版本为 8.2.4.15。

```
sudo apt install nvidia-cudnn
```

3.2 AMD 显卡深度学习环境配置（选看）

AMD 显卡也是可以进行深度学习环境搭建的, 对于 RX5700, RX6600, RX6750xt, RX7900XTX 等这些系列的显卡均可以进行深度学习环境搭建。这里以最新版本 ROCm5.7.1 为例子。首先, 下载并转换包签名密钥

```
sudo mkdir --parents --mode=0755 /etc/apt/keyrings
wget https://repo.radeon.com/rocm/rocm.gpg.key -O - | \
    gpg --dearmor | sudo tee /etc/apt/keyrings/rocm.gpg > /dev/null
```

对于 Ubuntu22.04, 在 apt 额外源库中添加 ROCm 源:

```
# jammy 版本的 Kernel 驱动仓库源
sudo tee /etc/apt/sources.list.d/amdgpu.list <<'EOF'
deb [arch=amd64 signed-by=/etc/apt/keyrings/rocm.gpg] https://repo.radeon.com/amdgpu/5.7.1/
ubuntu jammy main
```



```
EOF
# jammy 版本的 ROCm 驱动仓库源
sudo tee /etc/apt/sources.list.d/rocm.list <<'EOF'
deb [arch=amd64 signed-by=/etc/apt/keyrings/rocm.gpg] https://repo.radeon.com/rocm/apt/debian
jammy main
EOF
# 与系统包相比，更好是来自 rocm 存储库的包
echo -e 'Package: *\nPin: release o=repo.radeon.com\nPin-Priority: 600' | sudo tee /etc/apt/
preferences.d/rocm-pin-600
```

然后更新源

```
sudo apt update
```

安装 rocm-hip-libraries 元包。这包含大多数常见 ROCm 应用程序的依赖项，然后重启操作系统。

```
sudo apt install rocm-hip-libraries
sudo reboot
```

这样 ROCm 机器学习加速库安装完成。可以通过 rocm-smi 命令查看安装和 ROCm 运行的情况：



```
(webui-py38-rocm) linux@linux:~/桌面/stable-diffusion-webui$ rocm-smi
===== ROCm System Management Interface =====
===== Concise Info =====
GPU  Temp  AvgPwr  SCLK  MCLK  Fan  Perf  PwrCap  VRAM%  GPU%
0     62.0c   38.0W   500Mhz 1000Mhz 0%   auto  264.0W   6%     0%
===== End of ROCm SMI Log =====
```

图 9: rocm-smi 命令显示的内容

详细的安装教程可以参考其官网¹。

3.3 华为昇腾 Atlas300 系列 AI 加速卡深度学习环境配置（选看）

（待补充）

3.4 寒武纪 S370 系列 AI 加速卡深度学习环境配置（选看）

（待补充）

3.5 燧原 T20、T21 加速卡深度学习环境配置（选看）

（待补充）

3.6 摩尔线程 S70、S80AI 加速卡深度学习环境配置（选看）

（待补充）

¹https://rocm.docs.amd.com/en/latest/Installation_Guide/Installation-Guide.html

4 深度学习框架搭建环境安装

深度学习是一个非常大的领域，有很多的深度学习框架，不同的语言中也有不同的安装方式，包括 Python、Julia、lua 等等。这里我们主要介绍 Python、Julia、rust、go 等几种语言的深度学习框架的安装方法。

4.1 Python 深度学习环境配置

深度学习框架有很多种，这里我们主要介绍几种常见的深度学习框架的安装方法，包括 Pytorch、Tensorflow、Jax、Mindspore、MXNet、Theano 等等。这里我们主要介绍 Pytorch 和 Tensorflow 的安装方法，其他的框架的安装详细方法可以参考官网的安装教程。

4.1.1 Miniconda3 安装配置

Miniconda 是一个 Anaconda 的轻量级替代，默认只包含了 python 和 conda，但是可以通过 conda 命令来安装所需要的包。Miniconda 的安装非常简单，只需要下载对应的安装包，然后运行安装即可。下载地址可以选择国内的镜像站地址，文件目录位置在 anaconda/miniconda3/ 这个文件目录下，例如在清华镜像站地址。

| | | |
|--------------------------------------|-----------|------------------|
| Miniconda3-latest-Linux-aarch64.sh | 93.1 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-Linux-armv7l.sh | 29.9 MiB | 2017-01-31 01:54 |
| Miniconda3-latest-Linux-ppc64le.sh | 94.9 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-Linux-s390x.sh | 110.7 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-Linux-x86.sh | 62.7 MiB | 2019-01-03 00:11 |
| Miniconda3-latest-Linux-x86_64.sh | 115.4 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-MacOSX-arm64.pkg | 82.7 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-MacOSX-arm64.sh | 83.2 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-MacOSX-x86.sh | 26.0 MiB | 2017-01-31 01:55 |
| Miniconda3-latest-MacOSX-x86_64.pkg | 84.9 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-MacOSX-x86_64.sh | 85.3 MiB | 2023-11-17 03:59 |
| Miniconda3-latest-Windows-x86.exe | 67.8 MiB | 2022-05-17 04:01 |
| Miniconda3-latest-Windows-x86_64.exe | 74.0 MiB | 2023-11-17 03:59 |

图 10: 清华镜像站 Miniconda 下载

下载完成之后，直接使用 bash 安装即可

```
chmod +x Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
```

注意将对应的 miniconda 安装到的文件目录即可。最后一个安装步骤是初始化 conda，这里选择 yes，然后重启终端，即可完成 miniconda 的安装。

4.1.2 Python 虚拟环境的设置

这里分为系统虚拟环境使用方法和 conda 虚拟环境使用方法。

- 系统虚拟环境使用方法：

当然可以在系统环境中安装对应的虚拟环境包，用以下的命令安装虚拟环境

```
pip3 install virtualenv
```

可以通过以下的命令创建一个虚拟环境

```
python3 -m venv -p /usr/bin/python3
# 或者以下命令
virtualenv -p /bin/python3 venv
```

这样就会建立一个名字为 `venv` 的虚拟环境。然后激活虚拟环境

```
source path/to/virtualenv/bin/activate
```

停止虚拟环境、退出虚拟环境

```
deactivate
```

为了更加有效方便管理和安装虚拟环境，所以这里我们可以使用管理工具对虚拟环境进行管理和操作。

安装虚拟环境管理工具

```
pip3 install virtualenvwrapper
```

通常 `virtualenvwrapper` 有以下的命令进行操作和使用

```
mkvirtualenv venv # 新建虚拟环境
rmvirtualenv venv # 删除虚拟环境
workon # 查看安装的所有虚拟环境
workon venv # 进入或切换虚拟环境
cdsitepackages # 进入虚拟环境的site-packages目录
lssitepackages # 列出site-packages目录的所有软件包
```

- **conda 虚拟环境使用方法：**

conda 虚拟环境的使用方法和系统虚拟环境的使用方法类似，conda 默认为 `base` 虚拟环境，可以通过以下的命令查看当前的虚拟环境

```
conda info --envs
```

可以通过以下的命令创建一个虚拟环境

```
conda create -n venv python=3.8
```

这样就会建立一个名字为 `venv` 的虚拟环境。其中 `-n` 指的是虚拟环境指定虚拟环境名称，`python` 可以指定其中的版本号。然后激活虚拟环境

```
conda activate venv
```

停止虚拟环境、退出虚拟环境

```
conda deactivate
```

由于 conda 默认为官方的镜像源站，所以这里可以将其中的镜像源站修改为国内的镜像源站，例如清华镜像站，这样可以加快下载的速度。

```
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/main/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-
forge
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
conda config --set show_channel_urls yes
```

或者是在 `.condarc` 文件中添加以下内容

```
channels:
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/free/
- https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/main/
```

```
- conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda
  -forge
- conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2
  /
show_channel_urls: true
```

这样就可以将镜像源站修改为清华镜像站了。

更新 conda 使用以下的命令

```
conda update conda
```

更新 conda 中 base 环境中的所有包

```
conda update --all
```

4.1.3 Pytorch 环境安装

目前 Pytorch 分为 1.x 和 2.x 两个版本，相较于 1.x 版本，2.x 版本的 Pytorch 更加的轻量级，2.x 版本进一步增强了 Python 的效率。pytorch 深度学习环境配置一般通过以下的方式进行安装和配置，为了兼容所有深度学习环境，我们这里使用的是 cuda11.2 版本的配置，首先创建一个 pytorch 的虚拟环境，按照以下的方式进行安装：

```
conda create -n pytorch python=3.8
conda activate pytorch
```

然后安装 pytorch，这里我们选择的是 pytorch2.0 版本，安装命令如下所示：

```
conda install pytorch torchvision torchaudio cudatoolkit=11.2 -c pytorch -c nvidia
```

也可以通过 pip 安装 pytorch，安装命令如下所示：

```
pip install torch==2.1.0 torchvision==0.16.0 torchaudio==2.1.0 --index-url https://download.
pytorch.org/whl/cu118
```

安装完成之后，可以通过以下的命令查看 pytorch 的版本信息

```
python -c "import torch; print(torch.__version__)"
```

如果加速卡使用的是 AMD 显卡，对于基于 ROCm5.6 的深度环境，使用以下的命令进行安装：

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm5
.6
```

在项目的处理过程中，我们也使用到了以下的一些基于 pytorch 的深度学习环境，如下所示

- torchvision: 计算机图像处理包²;
- torchaudio: 音频处理包³;
- pytorch_geometric: 图神经网络库 (基于 networkx 图库)⁴;

这个安装过程稍有点复杂，需要安装以下的依赖包：

²<https://pytorch.org/vision/stable/index.html>

³<https://pytorch.org/audio/stable/index.html>

⁴<https://pytorch-geometric.readthedocs.io/en/latest/>

```
pip install --verbose --no-cache-dir torch-scatter
pip install --verbose --no-cache-dir torch-sparse
pip install --verbose --no-cache-dir torch-cluster
pip install --verbose --no-cache-dir torch-spline-conv #(optional)
pip install torch-geometric
```

- DGL: 图神经网络库 (基于 networkx 图库)⁵;
- allennlp: 神经网络库⁶;
- Transformers: 自然语言处理中的预训练模型⁷;
- 哈工大 ltp 自然语言处理包⁸;
- tensorboardX: 神经网络可视化;
- skorch: sklearn 的 pytorch 封装;
- gpytorch: 高斯过程推理库;
- botorch: 贝叶斯概率推断库;
- Pyro: 概率推断库;

安装过程中注意对应 CUDA 的版本号安装对应的包文件, 这里我们使用的是 CUDA11.2 版本的深度学习环境, 所以安装的包文件也是对应的 CUDA11.2 版本的包文件。对于不同的 Pytorch 库, 可以参考对应的 Pytorch 生态系统⁹。

测试 pytorch CUDA 环境是否可以使用: 可以使用以下的简单代码测试 pytorch 的深度学习加速环境是否可以使用

```
import torch
print(torch.cuda.is_available())
```

若显示 True 那么就可以使用 cuda 深度学习加速库。

4.1.4 Tensorflow 环境安装

首先要注意对应的 TensorFlow 的 GPU 版本, 如下图11所示。

这里我们选择的是 TensorFlow2.6 版本, 对应的 CUDA 版本为 11.2, cuDNN 版本为 8.1, Python 版本为 3.8。只需要以下的命令进行安装

```
pip install tensorflow-gpu==2.6.0 # 安装TensorFlow GPU版本
pip install tensorboard # 神经网络可视化图
```

测试 GPU 是否能使用的方法:

```
import tensorflow as tf
device = tf.config.list_physical_devices('GPU')
print(device)
```

如果出现了 GPU 的列表, 就说明 GPU 深度学习环境已经配置完成。

⁵<https://docs.dgl.ai/index.html>

⁶<https://docs.allennlp.org/main/>

⁷<https://huggingface.co/docs/transformers/index>

⁸<http://ltp.ai/docs/index.html>

⁹<https://pytorch.org/ecosystem/>

GPU

| Version | Python version | Compiler | Build tools | cuDNN | CUDA |
|-----------------------|----------------|--------------|--------------|-------|------|
| tensorflow-2.15.0 | 3.9-3.11 | Clang 16.0.0 | Bazel 6.1.0 | 8.9 | 12.2 |
| tensorflow-2.14.0 | 3.9-3.11 | Clang 16.0.0 | Bazel 6.1.0 | 8.7 | 11.8 |
| tensorflow-2.13.0 | 3.8-3.11 | Clang 16.0.0 | Bazel 5.3.0 | 8.6 | 11.8 |
| tensorflow-2.12.0 | 3.8-3.11 | GCC 9.3.1 | Bazel 5.3.0 | 8.6 | 11.8 |
| tensorflow-2.11.0 | 3.7-3.10 | GCC 9.3.1 | Bazel 5.3.0 | 8.1 | 11.2 |
| tensorflow-2.10.0 | 3.7-3.10 | GCC 9.3.1 | Bazel 5.1.1 | 8.1 | 11.2 |
| tensorflow-2.9.0 | 3.7-3.10 | GCC 9.3.1 | Bazel 5.0.0 | 8.1 | 11.2 |
| tensorflow-2.8.0 | 3.7-3.10 | GCC 7.3.1 | Bazel 4.2.1 | 8.1 | 11.2 |
| tensorflow-2.7.0 | 3.7-3.9 | GCC 7.3.1 | Bazel 3.7.2 | 8.1 | 11.2 |
| tensorflow-2.6.0 | 3.6-3.9 | GCC 7.3.1 | Bazel 3.7.2 | 8.1 | 11.2 |
| tensorflow-2.5.0 | 3.6-3.9 | GCC 7.3.1 | Bazel 3.7.2 | 8.1 | 11.2 |
| tensorflow-2.4.0 | 3.6-3.8 | GCC 7.3.1 | Bazel 3.1.0 | 8.0 | 11.0 |
| tensorflow-2.3.0 | 3.5-3.8 | GCC 7.3.1 | Bazel 3.1.0 | 7.6 | 10.1 |
| tensorflow-2.2.0 | 3.5-3.8 | GCC 7.3.1 | Bazel 2.0.0 | 7.6 | 10.1 |
| tensorflow-2.1.0 | 2.7, 3.5-3.7 | GCC 7.3.1 | Bazel 0.27.1 | 7.6 | 10.1 |
| tensorflow-2.0.0 | 2.7, 3.3-3.7 | GCC 7.3.1 | Bazel 0.26.1 | 7.4 | 10.0 |
| tensorflow_gpu-1.15.0 | 2.7, 3.3-3.7 | GCC 7.3.1 | Bazel 0.26.1 | 7.4 | 10.0 |
| tensorflow_gpu-1.14.0 | 2.7, 3.3-3.7 | GCC 4.8 | Bazel 0.24.1 | 7.4 | 10.0 |
| tensorflow_gpu-1.13.1 | 2.7, 3.3-3.7 | GCC 4.8 | Bazel 0.19.2 | 7.4 | 10.0 |
| tensorflow_gpu-1.12.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.15.0 | 7 | 9 |
| tensorflow_gpu-1.11.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.15.0 | 7 | 9 |
| tensorflow_gpu-1.10.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.15.0 | 7 | 9 |
| tensorflow_gpu-1.9.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.11.0 | 7 | 9 |
| tensorflow_gpu-1.8.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.10.0 | 7 | 9 |
| tensorflow_gpu-1.7.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.9.0 | 7 | 9 |
| tensorflow_gpu-1.6.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.9.0 | 7 | 9 |
| tensorflow_gpu-1.5.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.8.0 | 7 | 9 |
| tensorflow_gpu-1.4.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.5.4 | 6 | 8 |
| tensorflow_gpu-1.3.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.4.5 | 6 | 8 |
| tensorflow_gpu-1.2.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.4.5 | 5.1 | 8 |
| tensorflow_gpu-1.1.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.4.2 | 5.1 | 8 |
| tensorflow_gpu-1.0.0 | 2.7, 3.3-3.6 | GCC 4.8 | Bazel 0.4.2 | 5.1 | 8 |

图 11: TensorFlow GPU 版本对应的 CUDA、cuDNN、Python 版本

4.1.5 Jax 环境安装

Jax 深度学习框架¹⁰是 Google 开发的一个深度学习框架，它的特点是可以将 numpy 的代码转换为 GPU 代码，从而可以加速 numpy 的运算。Jax 的安装非常简单，只需要以下的命令即可：

```
conda create -n jax_env python=3.8
pip install --upgrade pip
pip install --upgrade "jax[cuda11_pip]" -f https://storage.googleapis.com/jax-releases/
jax_cuda_releases.html
```

剩下具体教程看官方教程即可，比较详细，这里不再赘述。

4.1.6 其他一些包环境

Python 库可以说是继 C/C++ 之后最为丰富的语言库，包含有很多的开发框架、实用工具、科学计算库、机器学习库、深度学习库等等。这里列举一些需要安装的一些常用的库，如下列举的一些库：

- **numpy、scipy**：科学计算库；
- **pandas**：数据处理库；
- **matplotlib、seaborn**：数据可视化库；
- **scikit-learn**：机器学习库；

以上是常见的一些 Python 库，当然还有很多其他的 Python 库，需要的库可以参考 Python 官方库¹¹。有些同学在编辑 Python 文件常用到 Jupyter Notebook，在本机上搭建方式如下所示：

```
conda install -c conda-forge notebook # 使用conda 安装notebook
```

4.1.7 Paddle 环境安装（选看）

Paddle¹²是百度开发的一个深度学习框架，它的特点是可以将 numpy 的代码转换为 GPU 代码，从而可以加速 numpy 的运算。同时 Paddle 也是一个非常好的深度学习框架，它的安装也非常简单，只需要以下的命令即可：

```
conda create -n paddle_env python=3.8
python -m pip install paddlepaddle-gpu==2.5.2.post112 -f https://www.paddlepaddle.org.cn/whl/
linux/mkl/avx/stable.html
```

Paddle 官方库中也有很多的深度学习库，其中大部分库使用的构建神经网络语法均与 Pytorch 类似，所以很多 Pytorch 的代码可以直接使用 Paddle 的库进行替换并且修改，从而可以使用 GPU 进行加速。

4.1.8 MindSpore 环境安装（选看）

昇思 MindSpore¹³是华为开发的一个深度学习框架，旨在为开发者提供高效、易用且可靠的深度学习工具。其设计理念包括灵活、可扩展、安全可靠和高性能。对于有单 CPU、GPU-CUDA10.1、GPU-CUDA11.1、GPU-CUDA11.6、Ascend910、Ascend310 版本：

这里我们在虚拟环境中创建 MindSpore 环境，如下所示：

¹⁰<https://jax.readthedocs.io/en/latest/>

¹¹<https://pypi.org/>

¹²<https://www.paddlepaddle.org.cn/>

¹³<https://www.mindspore.cn/>



图 12: MindSpore 版本对应 GPU、CPU、NPU 的 Python 版本

```
conda create -n mindspore_env python=3.8
conda activate mindspore_env
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/2.2.10/MindSpore/unified/
x86_64/mindspore-2.2.10-cp38-cp38-linux_x86_64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple
```

其它的一些 MindSpore 教程以及环境搭建可以参考 MindSpore 官网¹⁴。

4.1.9 OneFlow 环境安装（选看）

OneFlow¹⁵是开源的、采用全新架构设计,世界领先的工业级通用深度学习框架。OneFlow 可以进行分布式训练、多机多卡如单机单卡一样简单;完美契合一站式平台 (k8s + docker)、原生支持超大模型;近零运行时开销、线性加速比;灵活支持多种深度学习编译器、自动混合精度;中立开放,合作面广;持续完善的算子集、模型库。OneFlow 的安装也非常简单,只需要以下的命令即可:

```
conda create -n oneflow_env python=3.8
conda activate oneflow_env
python3 -m pip install -f https://release.oneflow.info oneflow==0.9.0+cu116
```

这里我们选择了 cu116 版本的 OneFlow,也可以选择其他版本的 OneFlow。教程以及使用方法参考官方教程¹⁶即可。

4.1.10 MXNet 环境安装（选看）

MXNet 是一个高效且灵活的开源深度学习框架,由 Apache 软件基金会支持。它提供了一个可扩展的计算图模型,允许用户在多种计算设备上运行深度学习模型,包括 CPU、GPU 和多个机器。MXNet 的设计注重效率和性能,可以处理大规模的数据集和复杂的模型。它具有用户友好的 API,使得构建、训练和部署模型变得简单。MXNet 还提供了丰富的工具和库,用于数据加载、模型评估和可视化。总之, MXNet 是一个功能强大且易于使用的开源框架,适用于各种深度学习任务和应用。

对于 Python 版本的 MXNet,可以通过以下的命令进行安装:

```
conda create -n mxnet_env python=3.8
conda activate mxnet_env
pip install mxnet-cu112
```

¹⁴<https://www.mindspore.cn/docs/zh-CN/r2.2/index.html>

¹⁵<https://www.oneflow.org/a/chanpin/oneflow/>

¹⁶<https://docs.oneflow.org/master/index.html>

这里安装的是 CUDA11.2 版本的 MXNet，也可以选择其他版本的 MXNet。

使用以下 Python 代码测试 MXNet 是否安装成功：

```
import mxnet as mx
a = mx.nd.ones((2, 3), mx.gpu())
b = a * 2 + 1
c = b.asnumpy()
print(c)
```

4.1.11 Chainer 环境安装（选看）

Chainer 是一个开源的深度学习库¹⁷，它提供了一种简单而灵活的方式来构建和训练神经网络模型。Chainer 的特点之一是它采用了动态图计算的方式，这意味着用户可以在模型构建和训练过程中使用 Python 的控制流和条件语句，使得模型的构建更加灵活和可读性更强。Chainer 还提供了丰富的优化算法和损失函数，以及可视化工具来帮助用户分析和调试模型。同时，Chainer 还支持多种硬件加速器，如 GPU 和 FPGA，以加快模型的训练和推理速度。总之，Chainer 是一个功能强大且易于使用的深度学习库，适用于各种机器学习任务。

Chainer 的安装非常简单，只需要以下的命令即可：

```
conda create -n chainer_env python=3.8
conda activate chainer_env
pip install chainer
```

可以在官方文档¹⁸中看多个例子学会此深度学习库的用法。

4.1.12 Theano 环境安装（选看）

Theano 是一个开源的机器学习库，它主要用于高效地定义、优化和评估数学表达式。它可以在多种硬件平台上运行，包括 CPU 和 GPU，并且能够自动优化代码以提高性能。同时提供了一个符号计算框架，其中可以定义各种数学表达式，如张量和矩阵操作。它还提供了许多高级的优化技术，以确保计算过程的高效性和准确性。Theano 还支持自动求导，这对于训练机器学习模型非常重要。通过定义损失函数并使用 Theano 的自动求导功能，可以方便地计算梯度并进行模型的参数更新。Theano 还具有与其他常用机器学习库（如 NumPy 和 SciPy）的良好集成，可以方便地进行数据处理和模型评估。

Theano GPU 环境的底层设计主要是依靠 pygpu 模块进行神经网络的加速。所以第一步首先要安装好 pygpu 环境。首先创建一个虚拟环境：

```
conda create -n theano_env python=3.8
conda activate theano_env
```

有以下的两种方式进行安装 pygpu：

- 第一种是在 conda 环境下安装。直接输入以下的命令就可以进行安装：

```
conda install pygpu
```

这个也在 conda-forge 包中进行安装

```
conda install -c conda-forge pygpu
```

然后再安装 theano 神经网络库的环境

¹⁷<https://chainer.org/>

¹⁸<https://docs.chainer.org/en/stable/>

```
pip install theano # 或者是 conda install theano
```

- 第二种方法是通过源码进行安装。首先编译 libpygpuarray, 需要 cmake,c99,cython 等依赖环境。cython 环境可以直接通过 pip 安装:

```
pip install cython
```

安装 cmake 可以直接这样安装

```
sudo apt install cmake
```

下面是对 libpygpuarray 进行安装。首先从 theano 的 github 上下载 libpygpuarray 的源码文件:

```
git clone https://github.com/Theano/libgpuarray.git
cd libgpuarray
```

创建编译目录进行编译。如果想用 libgpuarray 进行开发, 可以这样有如下编译方法:

```
cd libgpuarray
mkdir Build
cd Build
# you can pass -DCMAKE_INSTALL_PREFIX=/path/to/somewhere to install to an alternate
  location
cmake .. -DCMAKE_BUILD_TYPE=Release # or Debug if you are investigating a crash
make
make install
cd ..
```

如果只是想安装 pygpu, 只需要先激活虚拟环境 (如果需要的话), 然后在源码文件夹下面进行以下操作

```
python setup.py build
python setup.py install
```

Theano 文档中描述了很多 Theano 的使用方法, 可以参考 Theano 官网¹⁹。

theano GPU 配置文件的设置 在用户文件夹下创建.theanorc 文件:

```
nano ~/.theanorc
```

并且写入以下的配置文件

```
[global]
device = cuda
floatX=float32
root = /usr/local/cuda
[nvcc]
fastmath=True
compiler_bindir=/usr/local/cuda-10.1/bin
[blas]
ldflags = -lopenblas
[cuda]
root = /usr/local/cuda-10.1
[dnn]
```

¹⁹<https://docs.huihoo.com/theano/0.9/>

```
enabled=True
inclue_path=/usr/local/cuda/include
library_path=/usr/local/cuda/lib64
```

由于 Theano 目前已经不再更新，所以尽量使用 Docker 容器运行 Theano 环境，这里我列举了 CUDA-10.1 作为 Theano 的 GPU 加速环境。

theano GPU 配置文件的测试 创建以下的 python 文件

```
from theano import function, config, shared, tensor
import numpy
import time

vlen = 10 * 30 * 768 # 10 x #cores x # threads per core
iters = 1000

rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], tensor.exp(x))
print(f.maker.fgraph.toposort())
t0 = time.time()
for i in range(iters):
    r = f()
    t1 = time.time()
    print("Looping %d times took %f seconds" % (iters, t1 - t0))
    print("Result is %s" % (r,))
    if numpy.any([isinstance(x.op, tensor.Elemwise) and
        ('GPU' not in type(x.op).__name__) for x in f.maker.fgraph.toposort()]):
        print('Used the cpu')
    else:
        print('Used the gpu')
```

显示 Used the GPU 即配置成功。

关于 Theano 的一些深度学习库

- deepnet 深度学习库
- lasagne 深度学习库

这里注意对一些文件中有个地方的卷积函数进行修改：import lasagne 的时候发现下列错误：

```
ImportError: cannot import name 'downsample'
```

解决方案如下：

- 1). 找到下列发生错误的文件 \lasagne\layers\pool.py;
- 2). 将其中的 from theano.tensor.signal import downsample 修改为
from theano.tensor.signal.pool import pool_2d
- 3). 将文件中的downsample.max_pool_2d 改为 pool_2d。

4.2 Julia 深度学习环境配置（选看）

Julia 是一种高级、动态和高性能的编程语言，主要用于科学计算、数据分析和数值计算。它的设计目标是提供一种易于使用的语言，同时保持与传统科学计算语言（如 Matlab 和 Python）相似的表达能力。

Julia 的特点之一是它的速度，它具有接近于传统编译语言（如 C、C++）的性能。这得益于 Julia 的即时编译（JIT）技术，它能够将代码实时编译为机器码，从而提供高效的执行。此外，Julia 还支持多线程和并行计算，使得它在处理大规模数据和复杂计算任务时表现出色。

Julia 的语法简洁、灵活，易于学习和使用。它支持面向对象和函数式编程，并提供了丰富的内置函数和库，用于处理数值计算、线性代数、统计分析、优化等任务。此外，Julia 还具有强大的数据处理能力，可以直接操作数组、矩阵和其他数据结构。

Julia 是一个开源项目，拥有活跃的社区支持。它可以在多个操作系统上运行，并且可以与其他编程语言（如 Python、C、R）进行无缝集成，使得它成为科学计算和数据分析领域的一种强大工具。所以在深度学习环境中，Julia 也是一个非常好的选择。

4.2.1 Julia 环境安装

4.2.2 Flux ML 安装

4.2.3 其他一些包环境

4.3 Rust 深度学习环境配置（选看）

4.3.1 Rust 环境安装

4.3.2 Candle 深度学习环境安装

4.3.3 Burn 深度学习环境安装

Burn²⁰是一个基于 Rust 语言的深度学习库。

4.3.4 Tch-rs 深度学习环境安装

4.3.5 Neuronika 深度学习环境

<https://github.com/neuronika/neuronika>

4.3.6 其他一些包环境

4.4 Go 深度学习环境配置（选看）

4.4.1 Go 语言环境安装

4.4.2 Gorgonia 深度学习环境安装

4.5 lua 深度学习环境配置（选看）

4.5.1 torch7 深度学习环境安装

Torch7 是一种用于深度学习的开源框架，它基于 Lua 编程语言。它提供了一个强大且灵活的工具集合，用于构建和训练深度神经网络模型。Torch7 具有高效的计算能力和易于使用的接口，使得模型的开发和实验变得更加简单。它支持各种常见的深度学习任务，包括图像分类、目标检测、语音识别等。此外，Torch7 还提供了丰富的扩展库，使得用户可以方便地进行数据处理、可视化和模型优化。在后续开发中，facebook 将 Torch7 开发转为 PyTorch，torch7 也不再进行开发。总之，Torch7 是一个功能强大且易于使用的深度学习框架，适用于各种深度学习应用。

²⁰<https://github.com/tracel-ai/burn>

torch7 环境安装： 以 Ubuntu 系统为例子，首先安装一些编译工具和依赖包：

```
sudo apt-get install gcc g++ git cmake libreadline-dev
```

接下来，我们开始安装 torch7 环境。CUDA10.0 以下的环境安装选择：

```
git clone https://github.com/torch/distro.git ~/torch --recursive
```

CUDA10.0 以上的环境安装选择：

```
git clone https://github.com/nagadomi/distro ~/torch --recursive
```

目前在 CUDA 11.x 环境编译 torch7 会出现错误。

然后进入 torch 目录，执行以下的命令：

```
cd torch
bash install-deps # 检查并且安装一些依赖项环境
```

下一步对 torch7 进行编译安装：

```
bash install.sh
```

编译完成之后，需要对环境变量进行配置，会有以下的询问，选择 yes：

```
Do you want to automatically prepend the Torch install location to PATH and LD_LIBRARY in your
/home/asus/.bashrc?(yes/no)
[yes] >>>
yes
```

通过以下的命令使得环境变量生效：

```
source ~/.bashrc
```

最后，我们可以通过输入 th 命令测试 torch7 环境是否安装成功，如图13：



图 13: Torch7 命令行界面

在调用 torch.inverse(tensor) 过程中，出现了一个问题，即 getrf: Lapack library not found in compile time。出现这个问题的主要原因是由于没有安装 openBLAS 库。openBLAS 库是一个基于 C 和 fortran 的矩阵运算库，可以加速矩阵运算。首先确定操作系统中必须有 gfortran 编译环境。编译 openBLAS 需要较低版本的 gfortran 变异环境。如果没有，则以下命令进行安装：

```
sudo apt-get install gfortran
```

对于 Ubuntu22.04 操作系统，只有最低版本的 gfortran-8，所以安装了 gfortran-8 版本：

```
apt-get search gfortran
sudo apt-get install gfortran-8
sudo mv /usr/bin/gfortran /usr/bin/gfortran.bak #进行备份处理
sudo ln -s gfortran gfortran-8 #进行软链接操作
```

接下来安装 openBLAS:

```
git clone https://github.com/xianyi/OpenBLAS.git
cd ~/OpenBLAS
make NO_AFFINITY=1 USE_OPENMP=1 -j4
sudo make PREFIX=/usr/local/OpenBLAS install #默认是/opt/OpenBLAS.
sudo ln -s /opt/OpenBLAS/lib/libopenblas.so.0 /usr/lib/libopenblas.so.0
```

安装成功后, 设置一下路径

```
CMAKE_LIBRARY_PATH=/usr/local/OpenBLAS/include:/usr/local/OpenBLAS/lib:$CMAKE_LIBRARY_PATH
# 或者是在bashrc文件中定义变量CMAKE_LIBRARY_PATH, 或者是在安装torch之前定义此变量值
```

然后再进行 torch 的编译安装。

4.5.2 其他一些包环境

Lua 语言中有一个包管理器 luarocks, 可以通过 luarocks 安装一些必要的包。目前可以需要安装的包有以下几个:

- **gplots**: 数据可视化包;
- **nn**: 神经网络包;
- **cunn**: CUDA 神经网络加速包;
- **cutorch**: CUDA 神经网络加速包;
- **cuda**: CUDA 神经网络加速包;
- **iTorch**: Jupyter Notebook 的 Lua 版本。

上述的安装方式均可以用 luarocks 进行安装, 如下所示:

```
luarocks install gplots
```

其他包的安装方式同上述类似。

4.6 Java 深度学习环境配置 (选看)

4.6.1 Java 以及 Maven 环境安装

4.6.2 TensorFlow 在 Android 上的使用

4.6.3 Pytorch 在 Android 上的使用

4.6.4 DeepLearning4j 深度学习环境安装

4.6.5 其他一些环境

5 远程登录工具使用

5.1 公网远程连接桌面工具

公网远程桌面连接工具有很多, 我们这里列举几种较为常见的桌面连接工具。

- **ToDesk²¹**: ToDesk 远程控制软件是一款稳定流畅的远程控制电脑手机连接软件, 可远程桌面办公, 远程协助运维. 采用端对端加密, 让每一次远程访问都安全可靠。
对于 Linux 安装方式, 选择对应的架构和安装包下载安装即可。我们这里对应的是 amd64(x86-64) 架构, 所以选择对应的 Debian/Ubuntu/Mint 的 deb 包进行安装。

```
wget https://newdl.todesk.com/linux/todesk-v4.3.1.0-amd64.deb
sudo dpkg -i todesk-v4.3.1.0-amd64.deb
```

当然对于 Fedora 操作系统来说, 可以选择对应的 rpm 包进行安装。

```
wget https://newdl.todesk.com/linux/todesk-v4.3.1.0-x86_64.rpm
sudo rpm -ivh todesk-v4.3.1.0-x86_64.rpm
```

- **向日葵**: 向日葵远程控制软件是一款拥有多年远控技术经验的远程控制软件, 可远程控制手机, 远程桌面连接, 远程开机, 远程管理等, 并深入各行各业提供企业远程办公、企业 IT 运维、技术支持等企业远程解决方案。
对于 Linux 安装方式, 选择对应的架构和安装包下载安装即可。这里我们选择 Ubuntu/Deepin 的 amd64(x86-64) 架构安装。

```
wget https://down.oray.com/sunlogin/linux/SunloginClient_11.0.1.44968_amd64.deb
sudo dpkg -i sunloginclient_11.0.1.44968_amd64.deb
```

对于 Fedora 系列的操作系统, 选择对应的 rpm 包进行安装。

```
wget https://down.oray.com/sunlogin/linux/SunloginClient_11.0.1.44968_amd64.rpm
sudo rpm -ivh SunloginClient_11.0.1.44968_x86_64.rpm
```

Linux 系统可能会出现以下依赖包缺少情形, 并通过对应的安装命令安装即可

- "libXss.so" 依赖包缺少:

```
sudo apt-get install libxss1 # Debian系
sudo yum install libXScrnSaver # Fedora系
```

- "webkitgtk3" 依赖包缺少:

```
sudo apt-get install libwebkitgtk-3.0-0 # Debian系
sudo yum install libappindicator-gtk3 # Fedora系
```

- **RustDesk RustDesk²²**是一款开源的远程桌面软件, 它是由 Rust 语言开发的, 所以它的运行效率非常高, 同时它也是跨平台的远程桌面软件, 支持 Windows、Linux、MacOS 等操作系统。RustDesk 可以有效代替 TeamViewer、向日葵、AnyDesk 等远程桌面软件, 它的安装也非常简单, 只需要下载对应的安装包即可。下载地址是 Github 地址²³, 从发行版中选择自己的合适的架构和操作系统, 这里我们选择 Debian 系的 x86-64.deb 包。安装如下所示

```
wget https://github.com/rustdesk/rustdesk/releases/download/1.2.3/rustdesk-1.2.3-x86_64.deb
sudo dpkg -i rustdesk-1.2.3-x86_64.deb
```

对于 Fedora 系列的操作系统安装步骤如下所示

²¹<https://www.todesk.com/download.html>

²²<https://rustdesk.com/zh/>

²³<https://github.com/rustdesk/rustdesk/releases>

```
wget https://github.com/rustdesk/rustdesk/releases/download/1.2.3/rustdesk-1.2.3-0.x86_64.rpm
sudo rpm -ivh rustdesk-1.2.3-0.x86_64.rpm
```

服务器端安装成功之后，可以设置自己的密码，在客户端安装对应的 RustDesk，然后就可以远程控制电脑了。

- **AnyDesk（选看）** AnyDesk²⁴是一种远程桌面控制软件，它允许你远程访问电脑，或与其他人远程协作。通过 AnyDesk，你可以直接从你的电脑连接到另一台电脑，并在其上执行各种操作。AnyDesk 具有高度安全性和相对较快的连接速度，支持跨平台，包括 Windows，Mac OS，Android 和 iOS。而且 AnyDesk 支持多语言设置，你可以用你想用的语言来使用 AnyDesk。安装方式有两种，一种是基于源安装的方式，添加一下的源

```
wget -q0 - https://keys.anydesk.com/repos/DEB-GPG-KEY | sudo apt-key add -
echo "deb http://deb.anydesk.com/ all main" > /etc/apt/sources.list.d/anydesk-stable.list
sudo apt update
sudo apt install anydesk
```

这样就可以使用 AnyDesk 了。如果是 rpm 系列的包方式（Fedora 系列、SUES 系列等），可以查看官方源²⁵导入方式。另外一种是基于二进制包的安装方式，安装方式如下所示

```
wget https://download.anydesk.com/linux/anydesk_6.1.1-1_amd64.deb
sudo dpkg -i anydesk_6.1.1-1_amd64.deb
```

对于 RPM 系列的操作系统，安装方式如下所示

```
wget https://download.anydesk.com/linux/anydesk-6.1.1-1.x86_64.rpm
sudo rpm -ivh anydesk-6.1.1-1.x86_64.rpm
```

5.2 文件传输工具

文件传输工具首选的是 FileZilla²⁶，它是一个开源的 FTP 工具，可以在 Windows、Linux、MacOS 等操作系统上运行。它支持 FTP、FTPS 和 SFTP 协议，可以在客户端和服务端之间传输文件，并且支持 IPv6，可以在 IPv6 网络上运行。选择对应的操作系统以及处理器架构即可以安装，打开之后界面如图14所示：

²⁴

²⁵<http://rpm.anydesk.com/howto.html>

²⁶<https://filezilla-project.org/download.php>

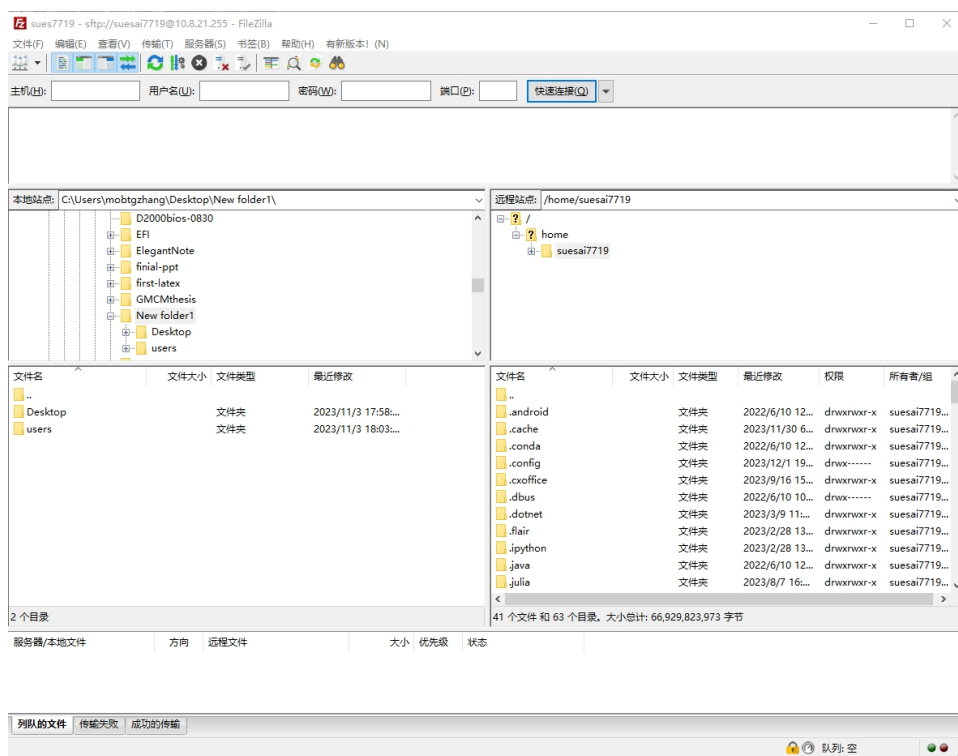


图 14: FileZilla 界面

5.3 VSCode 配置 SSH 远程连接

安装 VSCode²⁷之后，打开本地的 VSCode 开发插件菜单，在扩展程序中搜索 Remote-SSH 并安装，如图15所示。

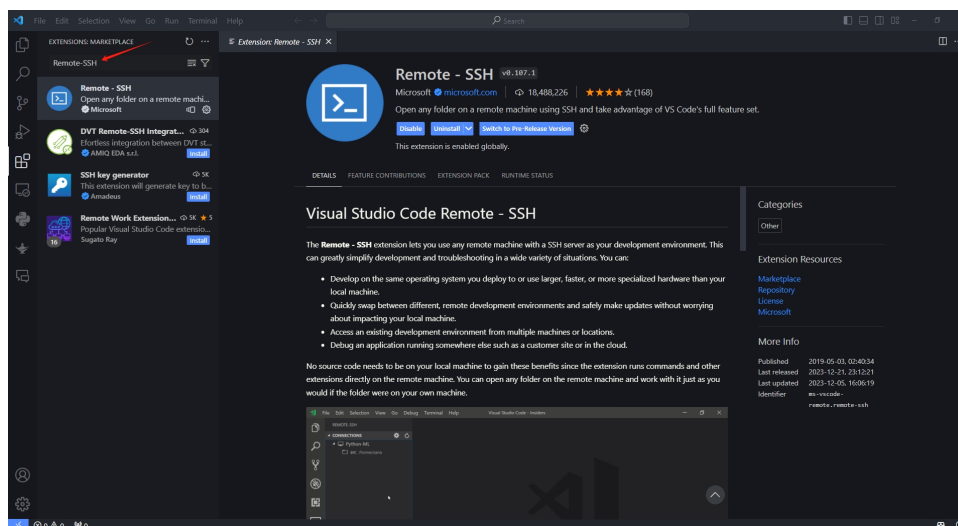


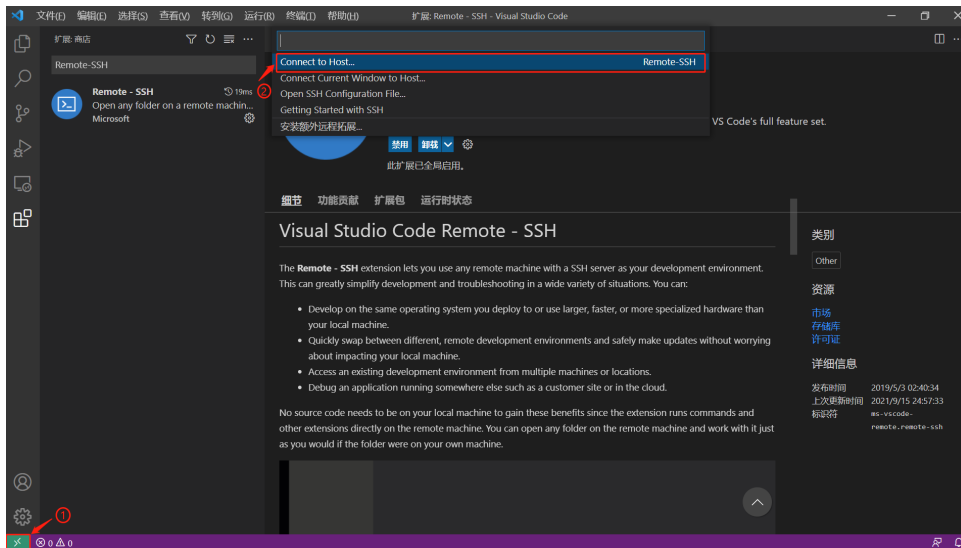
图 15: VSCode 界面

按照图示进行点击，完成添加 SSH 主机。

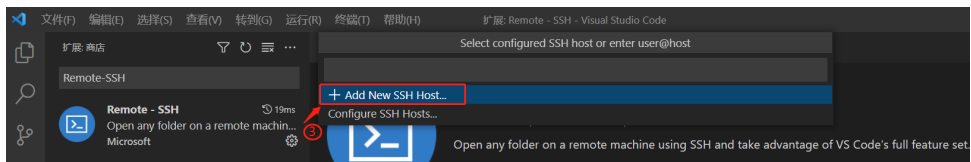
添加登录的命令，例如登录命令为

```
ssh -p ip_port username@ip_address
```

²⁷<https://code.visualstudio.com/>



(a) 找到主机连接面板



(b) 找到连接 IP 地址

图 16: VSCode 添加 SSH 主机

ip_port 为 SSH 端口号，默认为 22，username 为用户名，ip_address 为 IP 地址。

回车后会弹出以下自定义 SSH config 文件的弹窗，不需要修改，直接回车即可。选择直接回车即可。马上可能会弹出选择远程服务器是 Windows、Linux 和 Mac 系统的选项，请选择 Linux。然后就可以像在自己 PC 机上一样使用 VSCode 进行远程开发了。

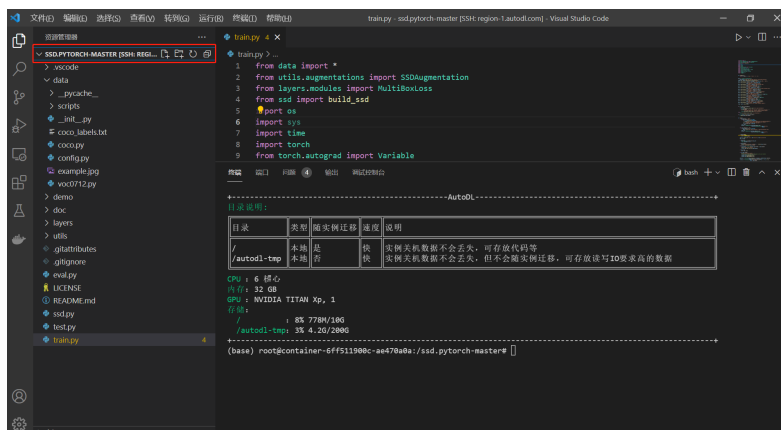


图 17: VSCode 远程开发

5.4 PyCharm 配置 SSH 远程连接

确认您安装的 PyCharm 是社区版还是专业版，只有专业版才支持远程开发功能。

配置 **PyCharm**: [File] -> [Settings]，打开以下设置弹窗，搜索 interpreter 找到 [Python interpreter] 设置项：

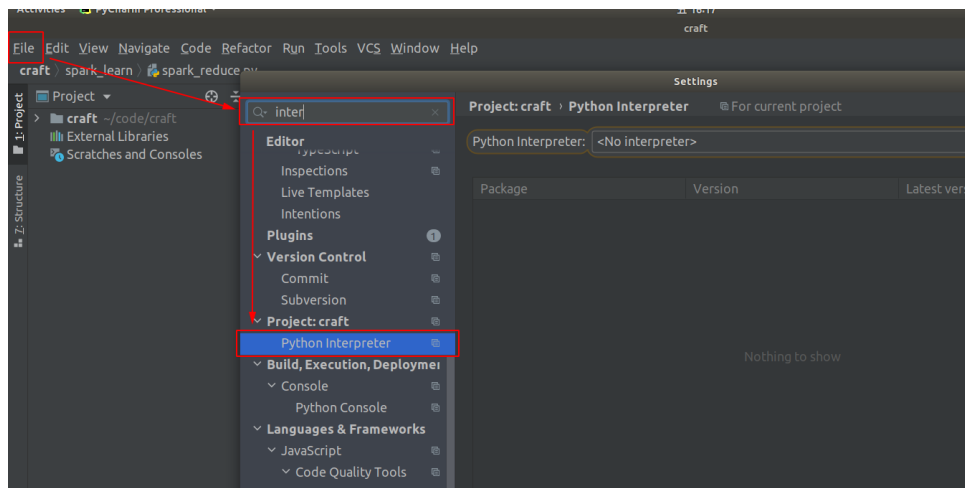


图 18: PyCharm 设置

点击 Add Interpreter，选择 On SSH 并点击 (PyCharm 社区版本无该选项)

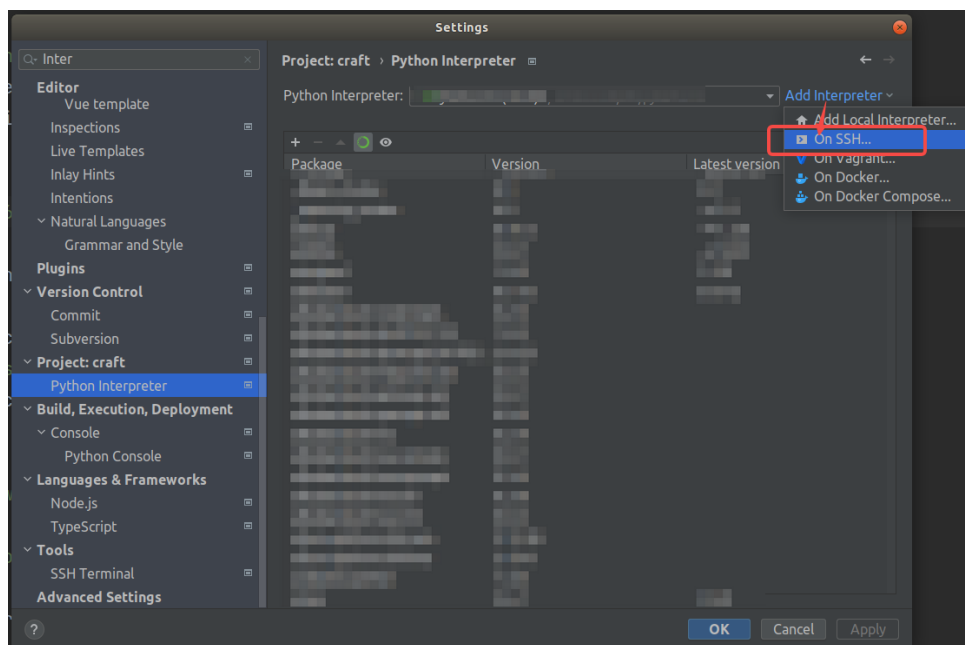


图 19: PyCharm 设置

将实例 SSH 指令中的 Host、Port 与 Username 进行匹配和填写，然后输入密码。

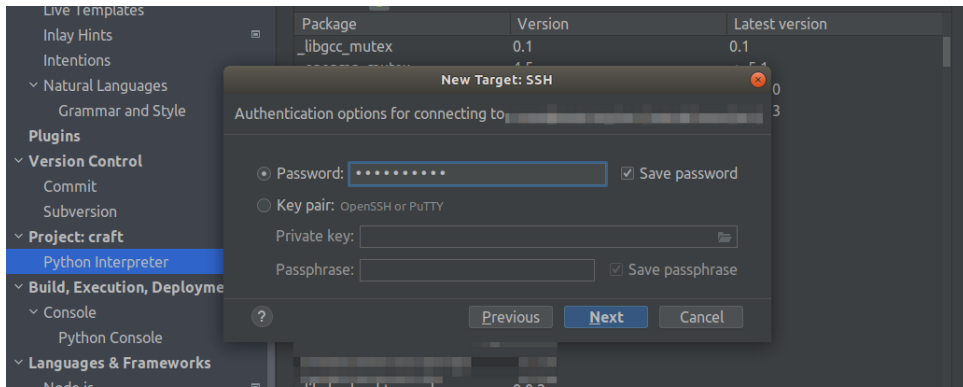


图 20: PyCharm 设置

继续下一步，选择 **System Interpreter**，Python 或虚拟环境则根据实际情况填写。配置同步目录，意思是本地项目和远程实例中的哪个目录进行关联。

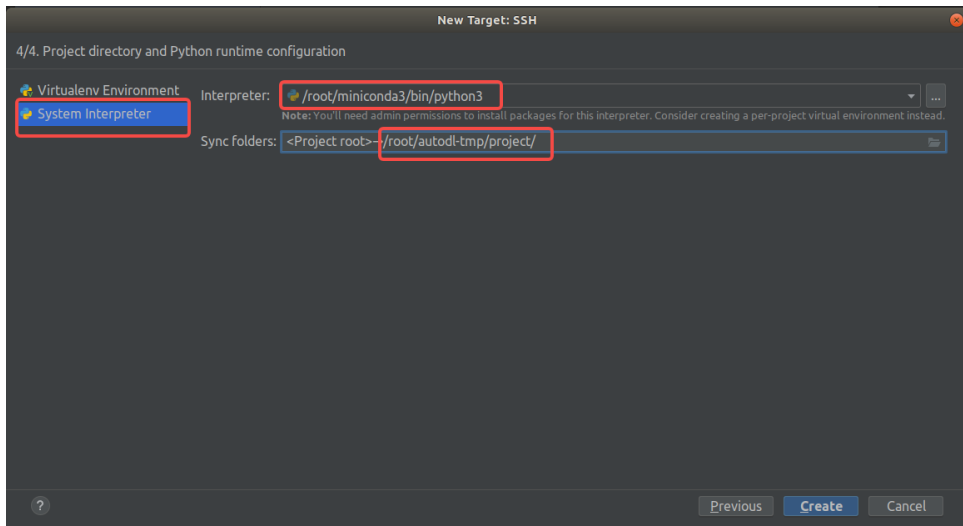


图 21: PyCharm 设置

如果您在运行时找不到 Python 文件，可能是没有自动同步代码，那么可以选择手动同步：

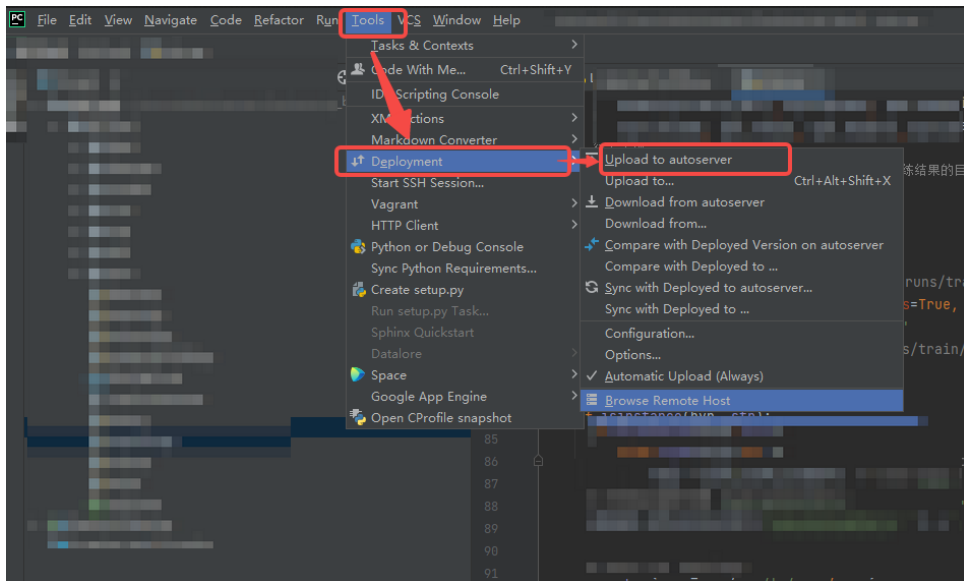


图 22: PyCharm 设置

配置好 PyCharm 远程开发后，可以在 PyCharm 的终端中下拉找到远程服务器打开远程终端：

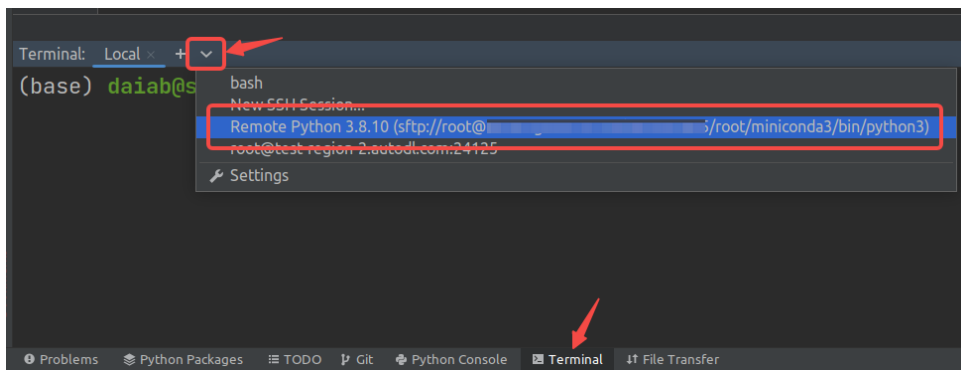


图 23: PyCharm 设置

也可以查看官方教程进行配置²⁸。

5.5 内网穿透（选看）

6 Docker 使用方法

Docker²⁹是一种开源的容器化平台，它可以帮助开发人员和运维团队更高效地构建、打包、分发和运行应用程序。Docker 使用轻量级的容器技术，将应用程序及其依赖项打包到一个可移植的容器中，使应用程序可以在任何环境中运行。通过使用 Docker，开发人员可以快速部署应用程序，减少环境配置和依赖项管理的复杂性。Docker 还提供了强大的容器管理工具，使运维团队能够轻松地管理和扩展容器化的应用程序。

²⁸<https://www.jetbrains.com/help/pycharm/configuring-remote-interpreters-via-ssh.html>

²⁹<https://www.docker.com/>

6.1 Docker 安装

对于 Ubuntu22.04 可以通过以下的方式安装 Docker，首先卸载与 Docker 冲突的软件包

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd
    runc;
do
    sudo apt-get remove $pkg;
done
```

有两种安装方式，一种是基于源安装的方式，另外一种是基于二进制包的安装方式。

基于 apt 源安装的方式 通过以下的方式添加 docker 源：

```
# 添加 Docker 的官方 GPG 公钥：
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/
    keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# 添加 apt 源：
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://
        download.docker.com/linux/ubuntu \
    $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

然后我们直接添加 Docker 的最新版本的源即可，按照以下的命令进行安装：

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
    compose-plugin
```

用下面的命令可以验证 Docker 是否安装成功：

```
sudo docker run hello-world
```

基于二进制包安装的方式 如果你不能使用 Docker 的 apt 存储库来安装 Docker 引擎，可以通过下载发布的 deb 文件并手动安装。每次升级 Docker 引擎时，都需要下载一个新文件，安装步骤如下所示：

- (1). 打开 Docker 官方下载页面网址³⁰；
- (2). 选择 Ubuntu 版本，这里我们选择 Ubuntu22.04 版本；
- (3). 到 pool/stable/ 文件夹下，由于我们服务器处理器为 x86-64 架构，所以这里选择 amd64 作为安装包文件；
- (4). 下载 Docker Engine、CLI、containerd 和 Docker Compose 包的以下 deb 文件：
 - containerd.io_<version>_<arch>.deb;
 - docker-ce_<version>_<arch>.deb;
 - docker-ce-cli_<version>_<arch>.deb;
 - docker-buildx-plugin_<version>_<arch>.deb;

³⁰<https://download.docker.com/linux/ubuntu/dists/>

- `docker-compose-plugin_<version>_<arch>.deb`

安装.deb 包，将以下示例中的路径更新为下载 Docker 软件包的位置。

```
sudo dpkg -i ./containerd.io_<version>_<arch>.deb \
./docker-ce_<version>_<arch>.deb \
./docker-ce-cli_<version>_<arch>.deb \
./docker-buildx-plugin_<version>_<arch>.deb \
./docker-compose-plugin_<version>_<arch>.deb
```

Docker 守护进程会自动启动。通过运行 `helloworld` 映像验证 Docker 引擎安装是否成功。

```
sudo service docker start
sudo docker run hello-world
```

此命令下载测试映像并在容器中运行。当容器运行时，它会打印一条确认消息并退出。

NVIDIA Docker 安装 NVIDIA Docker³¹是一个开源的项目，它可以让用户在 Docker 容器中运行 GPU 应用程序。NVIDIA Docker 包括一个运行时库和一个 Docker 命令行实用程序。NVIDIA Docker 运行时库是一个动态库，它提供了一个 Docker API 的实现，该实现可以将 Docker 容器中的 GPU 设备映射到主机上的 NVIDIA 设备。

通过下面的方法可以添加对应的 apt 源：

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

更显源，并安装 NVIDIA Container Toolkit

```
sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit
sudo systemctl restart docker
```

这样就安装好了对应的 NVIDIA Docker。

Docker 镜像加速 国内从 DockerHub 拉取镜像有时会遇到困难，此时可以配置镜像加速器。Docker 官方和国内很多云服务商都提供了国内加速器服务，建议根据运行 `docker` 的云平台选择对应的镜像加速服务。下面列出国内常用的加速站点，排名不分先后，总体来说阿里云速度较稳定。

- docker 中国区官方镜像加速：<https://registry.docker-cn.com>;
- 网易镜像加速：<http://hub-mirror.c.163.com>;
- 中国科技大学镜像加速：<https://docker.mirrors.ustc.edu.cn>;
- 腾讯云镜像加速：<https://mirror.ccs.tencentyun.com>;
- 阿里云镜像加速：<https://ung2thfc.mirror.aliyuncs.com>。

Linux 系统下，可以通过修改 `daemon` 配置文件 `/etc/docker/daemon.json` 来使用加速器。默认没有 `daemon` 文件，先创建。

```
nano /etc/docker/daemon.json
```

³¹<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/index.html>

添加如下内容：

```
{
  "registry-mirrors": [
    "https://ung2thfc.mirror.aliyuncs.com",
    "https://registry.docker-cn.com",
    "http://hub-mirror.c.163.com",
    "https://docker.mirrors.ustc.edu.cn"
  ]
}
```

然后加载重启 Docker：

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

6.2 Docker 使用

Docker 常用到的命令如表1所示：

表 1: Docker 容器命令

| 命令 | 说明 |
|----------------|---------------------|
| docker run | 创建一个新的容器并运行一个命令 |
| docker start | 启动一个或多个已经被停止的容器 |
| docker stop | 停止一个运行中的容器 |
| docker restart | 重启容器 |
| docker kill | 杀掉一个运行中的容器 |
| docker rm | 删除一个或多个容器 |
| docker pause | 暂停容器中所有的进程 |
| docker unpause | 恢复容器中所有的进程 |
| docker create | 创建一个新的容器但不启动它 |
| docker exec | 在运行的容器中执行命令 |
| docker attach | 连接到正在运行的容器 |
| docker wait | 阻塞运行直到容器停止 |
| docker ps | 显示所有容器 |
| docker images | 显示所有镜像 |
| docker load | 从一个 tar 包中加载一个镜像 |
| docker save | 将一个镜像保存成 tar 包 |
| docker export | 将一个容器保存成 tar 包 |
| docker import | 从一个 tar 包中创建一个容器 |
| docker commit | 创建一个新的镜像基于一个容器的改变 |
| docker build | 从 Dockerfile 构建一个镜像 |
| docker pull | 从 docker 仓库中拉取镜像或仓库 |
| docker rmi | 删除本地一个或多个镜像 |
| docker top | 显示容器的进程信息 |
| docker stats | 显示容器的统计信息 |
| docker tag | 标记本地镜像，将其归入某一仓库 |

Docker 容器的使用 我们可以从 Docker Hub 上拉取镜像，例如我们拉取一个 Ubuntu22.04 的镜像：

```
docker pull ubuntu:22.04
```

其中镜像名称一般命名为 < 仓库名 > : < 标签 >，如果不指定标签，默认为 latest。

可以通过以下的命令创建一个容器：

```
docker run -it --name ubuntu22.04 ubuntu:22.04 /bin/bash
```

其中 -it 参数表示创建一个交互式 (-i 参数) 终端 (-t 参数) 的容器，--name 参数表示容器的名称，ubuntu:22.04 表示使用的镜像，/bin/bash 表示容器启动后执行的命令。然后就可以进入刚创建的 Docker 容器环境。要退出终端，直接输入 exit。

创建容器并且后台运行：特别地，加了 -d 参数默认不会进入容器。

```
docker run -itd --name ubuntu22.04 ubuntu:22.04 /bin/bash
```

如果使用到 Nvidia GPU，可以通过以下的命令创建一个 GPU 容器：

```
docker run --gpus all -it --name ubuntu22.04 ubuntu:22.04 /bin/bash
```

-gpus all 表示的是使用所有的 GPU 设备，如果只想使用部分 GPU 设备，可以使用以下的命令：

```
docker run --gpus '"device=0,1"' -it --name ubuntu22.04 ubuntu:22.04 /bin/bash
```

创建好之后，上述方式就可以在容器中和正常操作系统一样使用了。

特别地，如果创建容器时候，需要指定容器的端口映射，可以使用以下的命令：

```
docker run -it --name ubuntu22.04 -p 8080:80 ubuntu:22.04 /bin/bash
```

其中 -p 参数表示容器的端口映射，8080 表示主机的端口，80 表示容器的端口，也就是将容器的 80 端口映射到主机的 8080 端口。

以下的命令用于显示容器命令信息：

```
docker ps -a # 显示所有的命令
```

停止容器命令：

```
docker stop <id_tag> # <id_tag>为容器ID或者容器名称
```

停止之后的容器可以通过以下的命令启动：

```
docker restart <id_tag> # <id_tag>为容器ID或者容器名称
```

删除容器命令：

```
docker rm <id_tag> # <id_tag>为容器ID或者容器名称
```

进入容器之后，在使用 -d 参数时，容器启动后会进入后台。此时想要进入容器，可以通过以下指令进入：

```
docker attach <id_tag> # <id_tag>为容器ID或者容器名称
```

或者是使用以下命令在容器中执行命令：

```
docker exec -it <id_tag> /bin/bash # <id_tag>为容器ID或者容器名称
```

注意：如果从这个容器退出，容器不会停止，这就是为什么推荐使用 docker exec 的原因。

如果要导出本地某个容器，可以使用 docker export 命令。

```
docker export <id_tag> > ubuntu.tar # <id_tag>为容器ID或者容器名称
```

这样就将容器导出为 ubuntu.tar 文件了。如果要导入容器快照到镜像，可以使用 docker import 命令。

```
docker import ubuntu.tar < test/ubuntu:v1.0
```

此外，也可以通过指定 URL 或者某个目录来导入，例如：

```
docker import http://example.com/exampleimage.tgz example/imagerepo
docker import - example/busybox < /home/ubuntu/test.tar.gz
```

其实可以通过 rootfs 来制作 Docker 镜像，后续再添加这段内容。

Docker 容器文件传输可以通过以下的命令进行：

```
docker cp <id_tag>:/file/path/within/container /host/path/target # <id_tag>为容器ID或者容器名称
```

传输可以是文件和文件夹。

Docker 镜像的使用 当运行容器时，使用的镜像如果在本地中不存在，docker 就会自动从 docker 镜像仓库中下载，默认是从 Docker Hub 公共镜像源下载。

如果想要查看本地镜像，可以使用以下的命令：

```
docker images
```

这样会列举出多个镜像，其中 REPOSITORY 表示镜像的仓库名称，TAG 表示镜像的标签，IMAGE ID 表示镜像的 ID，CREATED 表示镜像的创建时间，SIZE 表示镜像的大小。同一个仓库源可以有多个 TAG，表示这个仓库源的多个不同版本，使用 REPOSITORY:TAG 的方式来指定具体的镜像。所以在创建容器的时候，指定镜像的时候，可以使用 REPOSITORY:TAG 的方式来指定具体的镜像。

获取一个新的镜像，可以通过以下的命令：

```
docker pull <repository>:<tag> # <repository>为镜像仓库名称，<tag>为镜像标签
```

下载完成之后，镜像会缓存到本地，可以通过 docker images 查看。

如果想在仓库源上查找一个镜像，可以从 Docker Hub 上来搜索镜像，也可以通过以下的命令来搜索：

```
docker search <repository> # <repository>为镜像仓库名称
```

这样会通过正则表达式搜索镜像仓库名称，然后列出所有的镜像仓库名称。其中 NAME 表示镜像仓库名称，DESCRIPTION 表示镜像仓库的描述，STARS 表示镜像仓库的收藏数，OFFICIAL 表示是否为官方镜像，AUTOMATED 表示是否为自动构建的镜像。

删除镜像使用以下的命令进行删除操作：

```
docker rmi <repository>:<tag> # <repository>为镜像仓库名称，<tag>为镜像标签
```

创建镜像有两种方式，一种是将已有的容器提交为新的镜像，另外一种是通过 Dockerfile 来创建镜像。下面这条命令表示将容器提交为新的镜像，并且保存在本地：

```
docker commit <id_tag> <repository>:<tag> # <id_tag>为容器ID或者容器名称，<repository>为镜像仓库名称，<tag>为镜像标签
```

另外一种方式是通过 DockerFile 来创建镜像。Dockerfile 是一个文本文件，用来配置镜像的构建过程，Dockerfile 中包含了一条条的指令，每一条指令构建一层，因此每一条指令的内容，就是描述该层应当如何构建。DockerFile 使用到的命令如表2所示：

表 2: DockerFile 命令

| 命令 | 说明 |
|------------|---------------------------|
| FROM | 指定基础镜像 |
| MAINTAINER | 镜像作者信息 |
| RUN | 镜像构建时需要运行的命令 |
| EXPOSE | 容器对外暴露的端口 |
| WORKDIR | 指定工作目录 |
| ENV | 设置环境变量 |
| ADD | 将宿主机目录下的文件拷贝到镜像中 |
| COPY | 类似 ADD，将宿主机目录下的文件拷贝到镜像中 |
| VOLUME | 定义匿名卷 |
| CMD | 指定容器启动时要运行的命令 |
| ENTRYPOINT | 指定容器启动时要运行的命令 |
| ONBUILD | 当构建一个被继承 DockerFile 时运行命令 |

利用 DockerFile 构建镜像，下面一个例子说明了如何构建一个 Docker 镜像：创建一个 DockerFile 文件，内容如下所示：

```
FROM ubuntu:22.04
MAINTAINER mobtgzhang <
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Hi, I am in your container' \
    >/usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

其中 FROM 表示基础镜像，MAINTAINER 表示镜像作者信息，RUN 表示镜像构建时需要运行的命令，EXPOSE 表示容器对外暴露的端口，CMD 表示容器启动时要运行的命令。然后通过以下的命令创建一个自定义的镜像：

```
docker build -t nginx:v1.0 .
```

其中 -t 参数表示镜像的名称，nginx:v1.0 表示镜像的仓库名称和标签，. 表示 Dockerfile 所在的目录。

当然也可以通过 -f 来指定 Dockerfile 的路径，例如：

```
docker build -t nginx:v1.0 -f /path/to/Dockerfile .
```

修改设置镜像标签

我们可以使用 docker tag 命令，为镜像添加一个新的标签。

```
docker tag nginx:v1.0 nginx:latest
```

Docker 容器的连接 Docker 容器可以通过 ssh、sftp 等工具进行远程访问，也可以通过 VSCode、PyCharm 等工具进行远程开发。其远程访问配置方式和正常的 Linux 系统一样，这里不再赘述。

7 WSL2 使用方法

WSL 是 Windows Subsystem for Linux 的缩写，即 Windows 的 Linux 子系统，它是 Windows 10 的一个组件，允许用户在 Windows 上运行 GNU/Linux 环境的二进制程序。WSL2 是 WSL 的第二代，它是一个完全重新设计的版本，基于虚拟机技术，它使用了轻量级的虚拟机管理 Linux 内核，这样就可以在 Windows 上运行 Linux 的二进制程序。

7.1 WSL2 安装

首先，启用适用于 Linux 的 Windows 子系统。以管理员身份打开 powershell 或者 CMD 并运行：

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

然后更新 windows 系统至 1903 或更高版本。

然后，启用虚拟机功能。以管理员身份打开 powershell 或者 CMD 并运行：

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

然后，下载 WSL2 Linux 内核更新包。根据自己的系统选择对应的更新包，下载地址为https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi。

然后，将 WSL2 设置为默认版本。以管理员身份打开 powershell 或者 CMD 并运行：

```
wsl --set-default-version 2
```

最后，安装 Linux 发行版。打开 Microsoft Store，搜索 Linux，选择自己喜欢的 Linux 发行版，例如 Ubuntu，点击获取即可。

7.2 WSL2 使用方法

一些常用的命令如下所示：

查看 WSL 分发版本

```
wsl -l --all -v
```

导出分发版为 tar 文件到 d 盘

```
wsl --export Ubuntu-20.04 d:\wsl-ubuntu20.04.tar
```

注销当前分发版

```
wsl --unregister Ubuntu-20.04
```

重新导入并安装 WSL 在 d:\wsl-ubuntu20.04:

```
wsl --import Ubuntu-20.04 d:\wsl-ubuntu20.04 d:\wsl-ubuntu20.04.tar --version 2
```

设置默认登陆用户为安装时用户名

```
ubuntu2004 config --default-user Username
```

这里不再做过多介绍，服务器主要以 Ubuntu 等 Linux 系统为主，Docker 功能比 WSL2 强大很多，所以不再赘述。

8 LaTeX 安装教程

8.1 LaTeX 安装与配置

8.2 在线 LaTeX 编辑器

Overleaf、TexPage、<https://math-editor.online/>

<https://www.codecogs.com/latex/eqneditor.php>

<https://www.tablesgenerator.com/>

8.3 Typst 安装与配置（选看）

9 大模型部署方法

9.1 普通 CPU 部署方法（全平台）

9.2 使用 Intel 处理器部署（Intel 10 代以上 CPU）

9.3 使用 NVIDIA 处理器部署（显存至少 16GB 以上）

9.4 使用华为 Atlas300I 推理卡部署（选看）

9.5 使用摩尔线程 S80 部署（选看）