第 5 章 仓颉 TensorBoost 的环境搭建

工欲善其事，必先利其器。在详细进入仓颉 TensorBoost 编程体系前，首先带领读者搭建开发环境。仓颉 TensorBoost 需要 MindSpore 的支持，并且需要在 MindSpore 编译时打上仓颉 TensorBoost 的补丁程序。鉴于 MindSpore 的编译较为复杂，以至于本章绝大多数内容是在介绍 MindSpore 的编译方法，所以建议开发者严格按照流程进行操作，并且不要轻易试图改变各个软件的版本号，以免导致编译错误，即使操作流程全部正确，由于网络环境和系统环境的不同，在编译时还可能会遇到各种各样的异常问题，建议读者利用互联网资源，当遇到问题时耐心认真排查错误，相信各种问题都会解决。

本章的核心知识点如下：

- 搭建 MindSpore 编译环境。
- MindSpore 的下载、补丁配置和编译。
- 仓颉 TensorBoost 的环境配置。
- 测试仓颉 TensorBoost 是否安装正确。

5.1 准备工作

在搭建仓颉 TensorBoost 的环境前，需要进行硬件和软件方面的准备。

1. 硬件准备

仓颉 TensorBoost 支持 NVIDIA GPU 平台，也支持昇腾处理器平台。两者在搭建仓颉 TensorBoost 开发环境上大同小异，本章以 NVIDIA GPU 平台为例介绍仓颉 TensorBoost 开发环境搭建的全过程。

搭建仓颉 TensorBoost 的主机需要一定的算力基础，其中，NVIDIA GPU 算力要在 5.3 以上，读者可以在 NVIDIA 官网中查询各款显卡的算力。另外，编译过程需要消耗大量的 CPU 算力，所以应尽可能选择多核心高主频的 CPU，以便于提高编译速度，避免系统卡死。此外，建议磁盘预留 100GB 以上的存储空间。

注意 仓颉 TensorBoost 不能在虚拟机中正常使用。

2. 软件准备

仓颉 TensorBoost 的版本需要与 MindSpore、CUDA 版本严格一致，否则会导致

MindSpore 的编译错误，或者导致仓颉 TensorBoost 无法使用。本书以仓颉 TensorBoost 的 0.39.4 版本为例，其所对应的各个软件的版本如表 5-1 所示。

表 5-1 仓颉 TensorBoost 0.39.4 版本所对应的各个软件版本

软件名称	软件版本
Ubuntu 操作系统	18.04 LTS
仓颉语言和仓颉 TensorBoost	0.39.4
MindSpore	1.8.1 GPU
CUDA	11.1
cuDNN	8.0.X

准备好相应的硬件和软件之后，就可以在 Ubuntu 18.04 版本中搭建 MindSpore 编译环境了。建议在刚刚安装完毕的“干净”的 Ubuntu 操作系统中进行操作，以避免软件冲突导致编译错误。

3. Ubuntu 操作系统的相关准备工作

确认 Ubuntu 操作系统环境是否正常，并安装 git、vim 等基础软件。

1）确认 Ubuntu 的版本

通过 lsb_release 命令可以查询当前 Linux 系统的发行版信息，命令如下：

```
lsb_release -a
```

对于 Ubuntu 18.04.5 操作系统，输出的类似结果如下：

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:        18.04
Codename:       bionic
```

开发者可根据输出内容判断操作系统版本的正确性。

注意 LSB 是 Linux 标准基础(Linux Standard Base)的缩写，是 Linux 标准化的重要里程碑，用于提高 Linux 操作系统所使用代码的兼容性。

通过 uname 命令查询操作系统是否为 64 位，命令如下：

```
uname -a
```

对于 64 位操作系统，输出的类似结果如下：

```
Linux ms-gpu 5.4.0-56-generic #62~18.04.1-Ubuntu SMP Tue Nov 24 10:07:50 UTC 2020 x86_
64 x86_64 x86_64 GNU/LINUX
```

开发者可根据输出内容判断操作系统位数的正确性。

2）确认当前用户是否具有管理员权限

由于在环境搭建过程中需要安装依赖软件，通常需要用到 root 权限。幸运的是，对于

Ubuntu，默认安装时所创建的用户具有 root 权限。当需要用到 root 权限时，可以在命令前添加 sudo 命令(super user do)，即所有以 sudo 开头的命令都需要 root 权限。读者可以根据下一步中的 sudo apt upgrade 命令来判断当前用户是否具有 root 权限。

3) 通过 APT 安装基础软件

Ubuntu 可以通过高级包管理工具(Advanced Packaging Tool，APT)管理软件，其命令为 apt。apt 命令可以在线下载并安装绝大多数软件。在安装基础软件之前，首先需要更新 apt 源并更新现有的软件包，命令如下：

```
sudo apt update          # 更新 apt 源
sudo apt upgrade         # 更新软件包
```

注意　由于 Ubuntu 默认的 apt 源服务器布设在境外，所以可能更新较为缓慢。此时可以尝试使用国内的 apt 镜像源来安装基础软件。

随后，就可以通过 apt 命令安装 git、vim 等基础软件了，命令如下：

```
sudo apt install git vim -y
```

完成之后即可进入 MindSpore 编译环境的搭建流程了。

5.2　搭建 MindSpore 编译环境

本节按照显卡驱动与 CUDA 的安装、编译工具的安装、编译依赖的安装共 3 部分介绍 MindSpore 编译环境的搭建。部分软件之间存在依赖关系，读者不要盲目更换操作顺序，以防止产生不必要的编译错误。

在 MindSpore 的官网上提供了自动搭建环境的脚本(可参考 https://www.mindspore.cn/install 的相关说明)，主要包括两个步骤。

(1) 安装显卡驱动，命令如下：

```
sudo apt-get update
sudo apt-get install Ubuntu-drivers-common
sudo Ubuntu-drivers autoinstall
```

随后，重启计算机，通过以下命令检查显卡驱动是否安装成功：

```
nvidia-smi
```

(2) 通过脚本完成编译工具和依赖的配置，主要包括以下几方面：

- 将软件源配置为华为云源。
- 安装 MindSpore 所需的编译依赖，如 GCC 和 CMake 等。
- 通过 APT 安装 Python 3 和 pip 3，并设为默认。
- 下载 CUDA 和 cuDNN 并安装。
- 如果将 OPENMPI 设置为 on，则安装 Open MPI。

■ 如果将 LLVM 设置为 on，则安装 LLVM。

命令如下：

```
#下载自动脚本
wget https://gitee.com/mindspore/mindspore/raw/r1.8/scripts/install/ubuntu-gpu-source.sh
#运行自动脚本
OPENMPI=on bash -i ./ubuntu-gpu-source.sh
```

执行完毕后，重启计算机，即可完成本节的全部配置，但是，这种配置方法也经常因为网络和系统环境问题而出错，因此建议读者使用手动配置的方法搭建 MindSpore 编译环境。

5.2.1 显卡驱动与 CUDA 的安装

本节以 GeForce GTX 1050 Ti 显卡为例，介绍显卡驱动和 CUDA 的安装方法。其他显卡驱动的安装方法与此类似，只是显卡驱动的版本可能不同。

1. 显卡驱动的安装

Ubuntu 默认使用 Nouveau(开源驱动项目)作为 NVIDIA 显卡的默认驱动，所以需要替换为 NVIDIA 的官方驱动，否则无法正常使用 CUDA。Ubuntu 显卡驱动的安装方法有很多种，可以从 NVIDIA 官方下载对应显卡的 deb 包进行安装，也可以通过 APT 的方式进行安装等。

注意 在 NVIDIA 官方网站 https://www.nvidia.cn/Download/index.aspx?lang=cn 中可以下载合适的显卡驱动 deb 安装包。

使用 APT 安装的方式更加简便快捷，所以后文使用 APT 的方式安装 NVIDIA 显卡驱动。首先需要查询当前显卡的可用驱动，命令如下：

```
Ubuntu-drivers devices
```

此时，Ubuntu 会自动查找并列出当前显卡的可用驱动。对于 GeForce GTX 1050 Ti 显卡，该命令输出的信息如下：

```
== /sys/devices/pci0000:00/0000:00:03.0/0000:04:00.0 ==
modalias : pci:v000010DEd00001C82sv00001043sd0000862Abc03sc00i00
vendor   : NVIDIA Corporation
model    : GP107 [GeForce GTX 1050 Ti]
driver   : nvidia-driver-515 - distro non-free
driver   : nvidia-driver-470 - distro non-free recommended
driver   : nvidia-driver-470-server - distro non-free
driver   : nvidia-driver-510 - distro non-free
driver   : nvidia-driver-450-server - distro non-free
driver   : nvidia-driver-510-server - distro non-free
driver   : nvidia-driver-390 - distro non-free
driver   : nvidia-driver-418-server - distro non-free
driver   : nvidia-driver-515-server - distro non-free
driver   : xserver-xorg-video-nouveau - distro free builtin
```

由此可见，该显卡可用的驱动版本包括 515、470、510、390 等。由于新版本的显卡驱动是向上兼容的，因此这里选择最新驱动 nvidia-driver-515。

随后，即可通过 APT 的方式安装 nvidia-driver-515 驱动，命令如下：

```
sudo apt install nvidia-driver-515 -y
```

安装完成后，需要重启 Ubuntu 操作系统以便生效。通过 nvidia-smi 命令可以判断 NVIDIA 驱动是否安装成功。该命令执行后如果提示 NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver，则说明安装失败。安装成功后该命令的输出结果如图 5-1 所示。

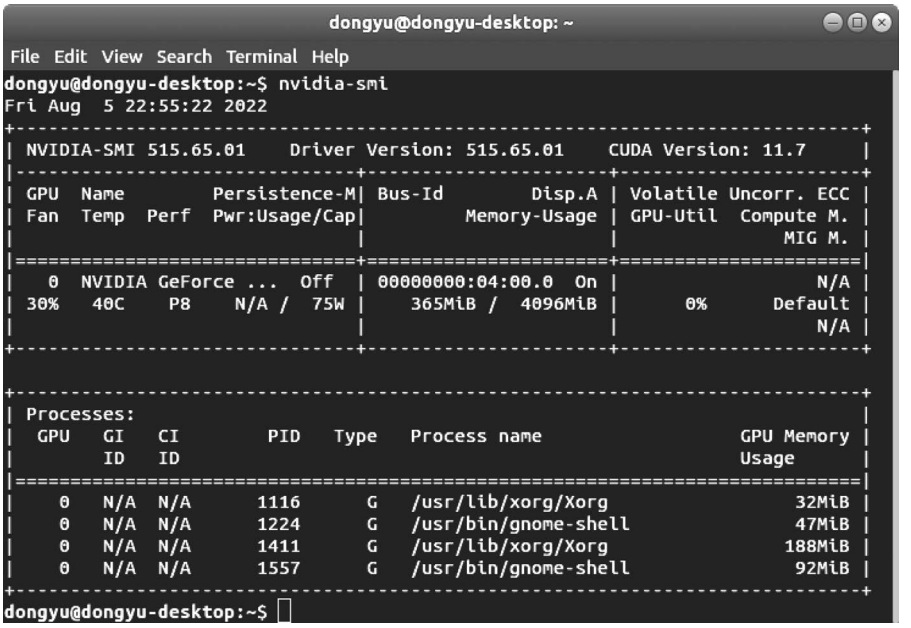

图 5-1　查询 NVIDIA 显卡是否安装成功

另外，还可在 Ubuntu 的设置(Settings)中的关于选项(About)中查询到显卡信息。如果在图形(Graphics)能够正常显示当前显卡的信息(如 NVIDIA GeForce GTX 1050 Ti/PCIe/SSE2)，则说明显卡驱动安装成功，如图 5-2 所示。

2. CUDA 的安装

CUDA 是 NVIDIA 提供的运算平台，名为统一计算设备架构(Compute Unified Device Architecture)。通过 CUDA 可以大大提高 GPU 的计算性能。MindSpore 1.8.1 支持最新的 CUDA 版本为 11.1。读者可以在 NVIDIA 官方网站(https://developer.nvidia.com/cuda-downloads)中下载并安装包 cuda_11.1.1_455.32.00_linux.run，也可以在本书的附带资源中找到。

定位到该文件所在目录后，通过以下命令开始安装驱动：

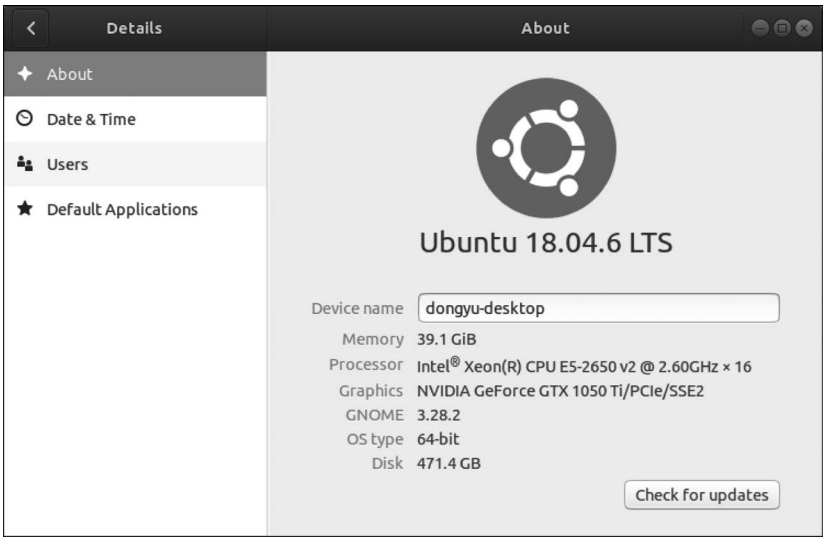

图 5-2　在设置中查询显卡信息

```
sudo ./cuda_11.1.1_455.32.00_linux.run
```

稍等片刻后，安装程序会提示 Existing package manager installation of the driver found. It is strongly recommended that you remove this before continuing，此时选择 Continue 继续，然后在随后的 CUDA 安装许可页面中输入 accept 后确定安装，如图 5-3 所示。

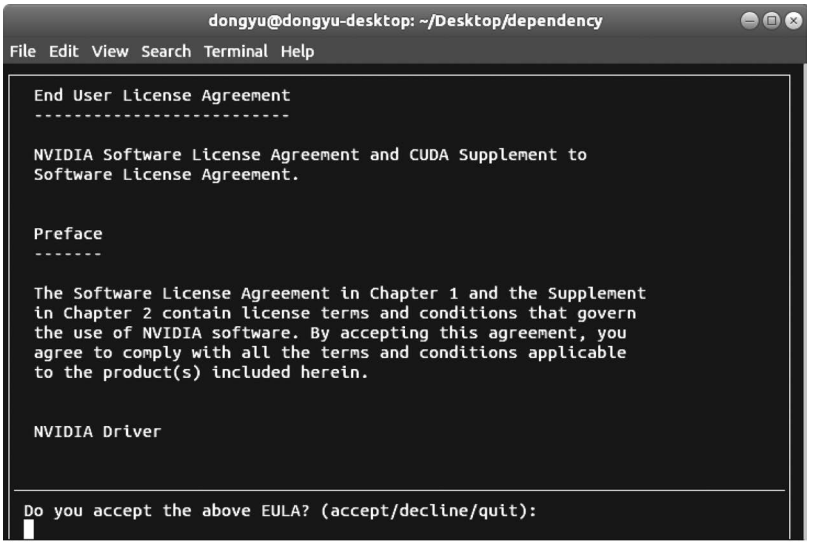

图 5-3　CUDA 的安装许可

在弹出的 CUDA Installer 页中，取消选中显卡驱动(Driver)选项，如图 5-4 所示(通过方向键可以定位到相应的项，通过 Enter 键可以选择相应的项)。这是因为在上一个步骤中

已经安装了当前显卡的最新驱动了，无须再次安装驱动。

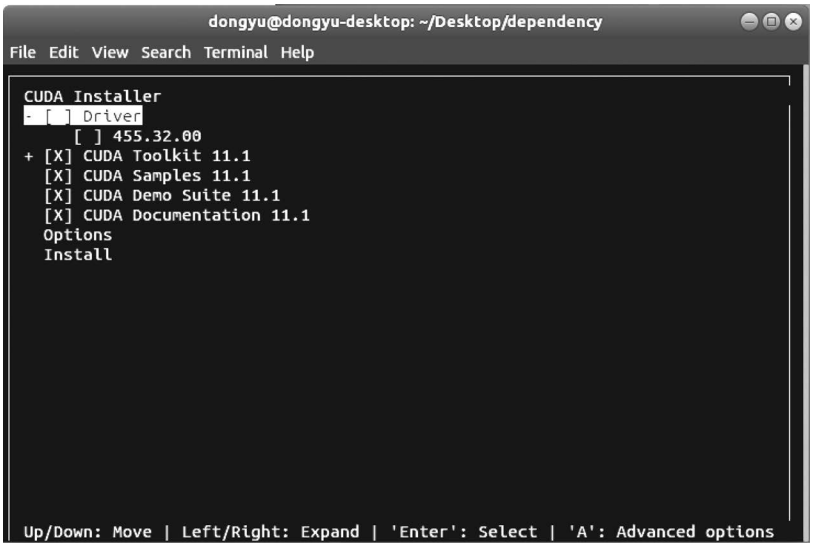

图 5-4　取消选中显卡驱动选项

选中安装(Install)选项后开始安装，这可能需要一段时间。在这段时间内，可能终端没有任何回显提示，需耐心等待。安装结束后，会提示如图 5-5 所示的信息。

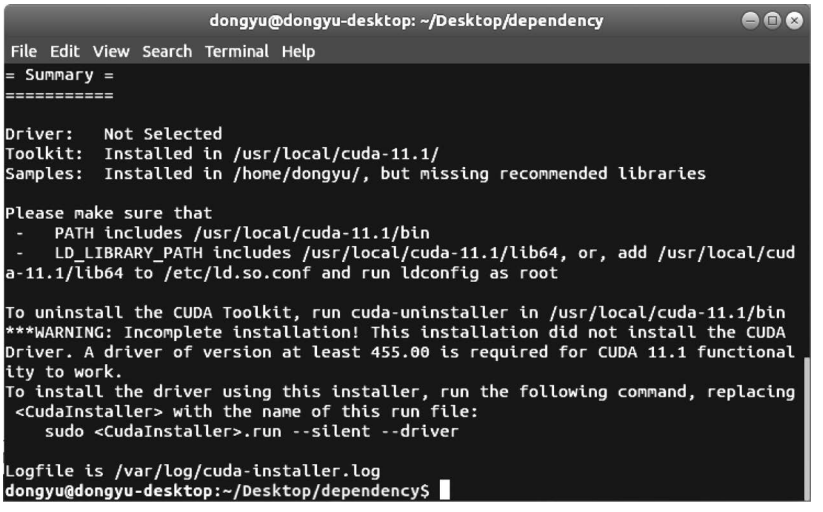

图 5-5　CUDA 安装成功

在～/.bashrc 文件进行 CUDA 的环境变量配置，在末尾添加的命令如下：

```
export PATH = /usr/local/cuda/bin: $ PATH
export LD_LIBRARY_PATH = /usr/local/cuda/lib64:\
/usr/local/cuda/extras/CUPTI/lib64: $ LD_LIBRARY_PATH
export LIBRARY_PATH = /usr/local/cuda/lib64: $ {LIBRARY_PATH}
```

保存该文件后，通过以下命令生效：

```
source ~/.bashrc
```

在随后的安装流程中，凡是修改.bashrc文件都需要在修改完毕后运行一遍上述命令使其生效，后文将不再赘述。

查询CUDA是否安装正确，命令如下：

```
nvcc -V
```

如果提示以下信息，则说明CUDA安装正确：

```
nvcc: NVIDIA (R) CUDA compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
CUDA compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
```

3. cuDNN的安装

cuDNN(CUDA Deep Neural Network Library)是CUDA的深度神经网络库，用于优化深度神经网络的计算能力。MindSpore 1.8.1和CUDA 11.1需要cuDNN 8.0.X版本配套，这里使用8.0.5版本的cuDNN进行安装介绍。

注意 读者可以在https://developer.nvidia.com/cudnn网站下载各个版本的cuDNN的库文件或安装文件。

下载libcudnn8_8.0.5.39-1+cuda11.1_amd64.deb和libcudnn8-dev_8.0.5.39-1+cuda11.1_amd64.deb文件之后，通过APT进行安装，命令如下：

```
sudo apt install ./libcudnn8_8.0.5.39-1+cuda11.1_amd64.deb
sudo apt install ./libcudnn8-dev_8.0.5.39-1+cuda11.1_amd64.deb
```

5.2.2 编译工具的安装

本节安装gcc、Python、cmake、automake等编译时需要的工具。由于在编译Python时需要gmp和OpenSSL的支持，所以gmp和OpenSSL依赖的安装在本节中一并介绍。本节需要安装的编译工具和依赖如表5-2所示。

表5-2 安装编译工具时所需的软件及其版本

软件名称	软件版本
gcc	7.3.0
gmp	6.1.2
OpenSSL	1.1.1
Python	3.7.5
wheel	0.32.0

续表

软 件 名 称	软 件 版 本
cmake	3.18.3
autoconf	2.70
automake	1.16

本节所有需要下载的编译工具均可在本书的附带资源中找到。

1. 安装 gcc 7.3.0

gcc(GNU Compiler Collection)全称为 GNU 编译器套件,是 Linux 操作系统中常用的编译器,可以用于编译 C、C++、Objective-C 等语言的源代码。MindSpore 1.8.1 编译需要使用的 gcc 版本为 7.3.0。在安装 gcc 7.3.0 之前,首先需要安装其依赖库,命令如下:

```
sudo apt install build-essential m4 libgmp-dev libmpfr-dev libmpc-dev -y
```

下载 gcc 7.3.0 源代码,命令如下:

```
wget http://ftp.gnu.org/gnu/gcc/gcc-7.3.0/gcc-7.3.0.tar.gz
```

注意　包括 gcc-7.3.0.tar.gz 在内,本章所涉及的所有编译工具和依赖均可在本书的配套资源中找到,后文将不再单独说明。

解压并进入 gcc 7.3.0 源代码目录,命令如下:

```
tar zxvf gcc-7.3.0.tar.gz
cd gcc-7.3.0
```

通过 configure 工具进行编译前配置,命令如下:

```
./configure --enable-checking=release \
            --enable-languages=c,c++\
            --disable-multilib
```

其中,--enable-checking=release 参数用于声明编译的目标为发行(非调试用途),--enable-languages=c,c++参数用于声明编译的支持语言为 C 和 C++,--disable-multilib 参数用于取消多目标编译。

随后,即可分别通过 make 和 make install 命令进行编译构建和安装了,命令如下:

```
make
sudo make install
```

通过-j 参数指定参与构建和安装的 CPU 核数,用于加速整个流程。例如,对于 16 核 CPU,可以使用-j16 参数将所有的 CPU 核心参与构建和安装,命令如下:

```
make -j16
sudo make install -j16
```

读者可以根据 CPU 核数将 16 替换为当前机器的 CPU 核数。在后文中,凡是涉及

make 和 make install 命令都可以通过-j 参数加速构建编译和安装过程。

当然,可以通过终端符号 && 依次执行构建和安装流程,命令如下:

```
make -j16 && sudo make install -j16
```

gcc 的编译时间较长,读者需耐心等待编译完成。编译和安装完成后,通过软连接将 gcc 7.3.0 设置为默认的 gcc 编译器,命令如下:

```
sudo ln -sf /usr/local/bin/gcc /usr/bin/gcc
```

执行 gcc --version 命令检查 gcc 7.3.0 是否安装成功。如果安装成功,则输出的结果如下:

```
gcc (GCC) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO warranty; not even for
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

2. 安装 gmp 6.1.2

gmp 是开源数学运算库(GNU MP Bignum Library),用于高精度的数学运算。下载 gmp 6.1.2 源代码,命令如下:

```
wget https://gmplib.org/download/gmp/gmp-6.1.2.tar.xz
```

gmp 的安装流程与 gcc 比较类似,包括解压、进入安装目录、配置、构建、安装这几个过程,命令如下:

```
tar xvf gmp-6.1.2.tar.xz          #解压
cd gmp-6.1.2/                     #进入安装目录
./configure                       #配置
make && sudo make install         #构建和安装
```

3. 安装 OpenSSL 1.1.1

OpenSSL 是 SSL 加密的开源实现,用于互联网数据的安全传输。下载 OpenSSL_1_1_1g.tar.gz 源代码文件,命令如下:

```
wget https://github.com/openssl/openssl/archive/OpenSSL_1_1_1g.tar.gz
```

解压并进入 OpenSSL 的源代码目录,命令如下:

```
tar zxvf OpenSSL_1_1_1g.tar.gz
cd openssl-OpenSSL_1_1_1g
```

配置、构建并安装 OpenSSL,命令如下:

```
./config --prefix=/usr/local/openssl
make && sudo make install
```

其中,--prefix=/usr/local/openssl 参数用于指定 OpenSSL 的安装位置。最后,在

～/.bashrc 文件中进行 OpenSSL 的环境变量配置，在末尾添加的命令如下：

```
export OPENSSL_ROOT_DIR = /usr/local/openssl
```

OpenSSL 的安装主要是为确保下一步安装 Python 3.7.5 后，能够通过 SSL 的方式安装 pip 库依赖。

4. 安装 Python 3.7.5

MindSpore 源代码编译时需要使用 Python 脚本，并且部分依赖需要通过 pip 工具安装。在 Ubuntu 18.04 中默认的 Python 版本为 3.6，需要替换安装为 Python 3.7.5。另外，需要更新 pip，并通过 pip 安装 wheel 0.32.0。

1）安装 Python 3.7.5

在安装 Python 3.7.5 之前，首先需要安装其依赖库，命令如下：

```
sudo apt install -y libssl-dev zlib1g-dev libbz2-dev \
                        libreadline-dev libsqlite3-dev wget \
                        curl llvm libncurses5-dev libncursesw5-dev \
                        xz-utils tk-dev libffi-dev liblzma-dev python-openssl
```

下载 Python 3.7.5 源码包：

```
wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
```

解压并进入 Python 的源代码目录，命令如下：

```
tar zxvf Python-3.7.5.tgz
cd Python-3.7.5
```

运行配置脚本，准备构建环境，命令如下：

```
./configure --prefix=/usr/local/python3.7.5 \
            --with-openssl=/usr/local/openssl \
            --enable-shared
```

其中，--prefix=/usr/local/python3.7.5 参数用于指定其安装目录，--with-openssl=/usr/local/openssl 参数用于指定上一步 OpenSSL 的安装目录，--enable-shared 用于启用 Python 库的共享。

编译、构建并安装 Python 3.7.5，命令如下：

```
make && sudo make install
```

安装完成后，在～/.bashrc 文件中进行 Python 的环境变量配置，在末尾添加的命令如下：

```
export PATH=/usr/local/python3.7.5/bin:～/.local/bin:$PATH
export LIBRARY_PATH=/usr/local/python3.7.5/lib:${LIBRARY_PATH}
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:${LD_LIBRARY_PATH}
```

为了方便使用 Python 3.7.5，这里通过软连接的方式将 Python 3.7.5 设置为系统默认

的 Python 程序，命令如下：

```
sudo ln -sf /usr/local/python3.7.5/bin/python3 /usr/bin/python
sudo ln -sf /usr/local/python3.7.5/bin/python3 /usr/bin/python3
```

检查 Python 是否安装正确，命令如下：

```
python --version
```

如果安装正确，则输出的结果如下：

```
Python 3.7.5
```

2）更新 pip

更新 pip 软件版本，命令如下：

```
sudo python -m pip install --upgrade pip
```

如果安装过程中提示 subprocess.CalledProcessError: Command '('lsb_release', '-a')' returned non-zero exit status 1 错误，则可以将/usr/bin/lsb_release 文件删除（或者将 lsb_release 重命名为其他名称），命令如下：

```
sudo rm -rf /usr/bin/lsb_release
```

如果安装过程中提示 python: error while loading shared libraries: libpython3.7m.so.1.0: cannot open shared object file: No such file or directory 错误，则可将 libpython3.7m.so.1.0 文件复制到/usr/lib 目录，命令如下：

```
sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib
```

安装完成后，检查 pip 是否安装正确，命令如下：

```
pip --version
```

如果 pip 安装成功，则输出的结果如下：

```
pip 22.2.2 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
```

随着 pip 版本的更新，版本号可能不同，但是输出结果与此类似。

注意 由于后文需要安装依赖，建议将 pip 修改为国内源，以便加快环境搭建的速度。

3）安装 wheel 0.32.0

wheel 是 Python 中的打包工具，提供跨平台和跨机器的一致性安装。安装 wheel 0.32.0，命令如下：

```
pip install wheel==0.32.0
```

安装过程中输出的信息如下：

```
Defaulting to user installation because normal site-packages is not writeable
Collecting wheel == 0.32.0
  Downloading wheel-0.32.0-py2.py3-none-any.whl (21 kB)
Installing collected packages: wheel
Successfully installed wheel-0.32.0
```

如果提示 Successfully installed wheel-0.32.0，则表示安装成功。

5. 安装 cmake 3.18.3

cmake(cross-platform make)是跨平台的构建编译和构建工具，建立在已有的 make 等构建工具之上，可以将构建编译的过程依据平台生成对应的 makefile、project 等文件。这些文件再通过本机的构建编译工具生成具体的二进制产品。首先下载 cmake 3.18.3，命令如下：

```
wget https://cmake.org/files/v3.18/cmake-3.18.3.tar.gz
```

解压并进入 cmake 的源代码目录，命令如下：

```
tar zxvf cmake-3.18.3.tar.gz
cd cmake-3.18.3
```

通过 Bootstrap 命令运行配置脚本，并分别通过 make 和 make install 命令构建并安装 cmake，命令如下：

```
./Bootstrap
make -j16 && sudo make install -j16
```

cmake 的构建编译过程也较慢，需耐心等待。安装完成后，检查 cmake 是否安装正确，命令如下：

```
cmake --version
```

输出的结果如下：

```
cmake version 3.18.3
CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

6. 安装 autoconf 2.70

autoconf 是自动化配置脚本生成工具，通常和 automake 配合使用。下载 autoconf，命令如下：

```
wget http://ftp.gnu.org/gnu/autoconf/autoconf-2.70.tar.gz
```

解压并进入 autoconf 的源代码目录，命令如下：

```
tar zxvf autoconf-2.70.tar.gz
cd autoconf-2.70
```

通过 configure 命令运行配置脚本，分别通过 make 和 make install 命令构建并安装

autoconf，命令如下：

```
./configure
make -j16 && sudo make install -j16
```

安装完成后，检查 autoconf 是否安装正确，命令如下：

```
autoconf --version
```

如果安装正确，则输出的结果如下：

```
autoconf (GNU Autoconf) 2.70
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+/Autoconf: GNU GPL version 3 or later
<https://gnu.org/licenses/gpl.html>, <https://gnu.org/licenses/exceptions.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by David J. MacKenzie and Akim Demaille.
```

7. 安装 automake 1.16

automake 和 cmake 类似，是一种高层级的构建工具，可以用于生成 makefile。首先下载 automake，命令如下：

```
wget https://ftp.gnu.org/gnu/automake/automake-1.16.tar.gz
```

解压并进入 automake 的源代码目录，命令如下：

```
tar zxvf automake-1.16.tar.gz
cd automake-1.16
```

通过 configure 命令运行配置脚本，并分别通过 make 和 make install 命令构建并安装 automake，命令如下：

```
./configure
make -j16 && sudo make install -j16
```

检查 automake 是否安装正确，命令如下：

```
automake --version
```

如果安装正确，则输出的结果如下：

```
automake (GNU automake) 1.16
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv2+: GNU GPL version 2 or later <https://gnu.org/licenses/gpl-2.0.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Tom Tromey <tromey@redhat.com>
        and Alexandre Duret-Lutz <adl@gnu.org>.
```

5.2.3　编译依赖的安装

本节需要安装的依赖包括 patch、Libtool 等，如表 5-3 所示。除了 NUMA 通过 APT 的方式安装以外，其余依赖均使用 make 构建源代码的方式进行安装，其方法大同小异。

表 5-3　需要安装的依赖及其版本

软 件 名 称	软 件 版 本
patch	2.6
Libtool	2.4.6
flex	2.6.4
OpenMPI	4.0.3
NUMA	2.0.11

1. 安装 patch 2.6

patch 是常用的代码维护工具，可以用于对代码的更新。首先下载 patch 2.6，命令如下：

```
wget https://ftp.gnu.org/gnu/patch/patch-2.6.tar.gz
```

解压并进入 patch 的源代码目录，命令如下：

```
tar zxvf patch-2.6.tar.gz
cd patch-2.6
```

通过 configure 命令运行配置脚本，分别通过 make 和 make install 命令构建并安装 patch，命令如下：

```
./configure
make -j16 && sudo make install -j16
```

检查 patch 是否安装正确，命令如下：

```
patch --version
```

如果安装正确，则输出的结果如下：

```
patch 2.6
Copyright (C) 1988 Larry Wall
Copyright (C) 2003 Free Software Foundation, Inc.

This program comes with NO WARRANTY, to the extent permitted by law.
You may redistribute copies of this program
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.

written by Larry Wall and Paul Eggert
```

2. 安装 Libtool 2.4.6

Libtool 是大型软件编译中常用的库依赖管理工具。首先下载 Libtool 2.4.6，命令

如下：

```
wget https://ftpmirror.gnu.org/libtool/libtool-2.4.6.tar.gz
```

解压并进入 Libtool 的源代码目录，命令如下：

```
tar zxvf libtool-2.4.6.tar.gz
cd libtool-2.4.6
```

通过 configure 命令运行配置脚本，分别通过 make 和 make install 命令构建并安装 Libtool，命令如下：

```
./configure
make -j16 && sudo make install -j16
```

检查 Libtool 是否安装正确，命令如下：

```
libtool --version
```

如果安装正确，则输出的结果如下：

```
libtool (GNU libtool) 2.4.6
Written by Gordon Matzigkeit, 1996

Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3. 安装 flex 2.6.4

flex 是快速语法分析生成器(Fast Lexical Analyzer Generator)。首先下载 flex 2.6.4，命令如下：

```
wget https://github.com/westes/flex/releases/download/v2.6.4/flex-2.6.4.tar.gz
```

解压并进入 flex 的源代码目录，命令如下：

```
tar zxvf flex-2.6.4.tar.gz
cd flex-2.6.4
```

通过 configure 命令运行配置脚本，分别通过 make 和 make install 命令构建并安装 flex，命令如下：

```
./configure
make -j16 && sudo make install -j16
```

检查 flex 是否安装正确，命令如下：

```
flex --version
```

如果安装正确，则输出的结果如下：

```
flex 2.6.4
```

4. 安装 OpenMPI 4.0.3

OpenMPI 是高性能信息传递库，是 MPI(Message Passing Interface)的开源实现。在高性能计算领域，OpenMPI 能够显著提高并发性能。首先下载 OpenMPI 4.0.3，命令如下：

```
wget https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-4.0.3.tar.gz
```

解压并进入 OpenMPI 的源代码目录，命令如下：

```
tar zxvf openmpi-4.0.3.tar.gz
cd openmpi-4.0.3
```

通过 configure 命令运行配置脚本，分别通过 make 和 make install 命令构建并安装 OpenMPI，命令如下：

```
./configure
make -j16 && sudo make install -j16
```

检查 OpenMPI 是否安装正确，命令如下：

```
whereis openmpi
```

如果安装正确，则输出的结果如下：

```
openmpi: /usr/local/lib/openmpi
```

将/usr/local/lib 目录添加到 LD_LIBRARY_PATH 环境变量中，修改～/.bashrc，添加以下内容：

```
export LD_LIBRARY_PATH=/usr/local/lib:${LD_LIBRARY_PATH}
```

如果这一步设置错误，则在运行仓颉 TensorBoost 时会弹出以下提示：

```
./demo: error while loading shared libraries: libmpi.so.40: cannot open shared object file: No
such file or directory
```

5. 安装 NUMA 2.0.11

NUMA(Non Uniform Memory Access)是非统一内存访问技术，常用于分布式计算系统中的内存管理。通过 APT 安装 NUMA，命令如下：

```
sudo apt install libnuma-dev -y
```

检查 NUMA 是否安装正确，命令如下：

```
whereis numa
```

如果安装一切正常，则输出的结果如下：

```
numa: /usr/include/numa.h /usr/share/man/man3/numa.3.gz /usr/share/man/man7/numa.7.gz
```

安装完成上述软件以后，就可以编译 MindSpore 了。

5.3 编译并安装 MindSpore

首先，通过 git 工具下载 MindSpore 的代码仓，命令如下：

```
git clone https://gitee.com/mindspore/mindspore.git
```

进入 MindSpore 源代码目录后，可通过 git 命令将其切换到 1.8.1 版本，命令如下：

```
git checkout v1.8.1
```

开始编译 MindSpore，命令如下：

```
./build.sh -e gpu -S on -j16
```

其中，“-e gpu”参数指明了编译后 MindSpore 的运行平台为 NVIDIA GPU；“-S on”参数表示使用 gitee 镜像站下载相关依赖，在国内可以加速下载速度；“-j16”参数用于指定编译时应用 CPU 的核心数，需要根据设备的具体情况进行修改。如果运行平台为昇腾平台，则编译命令如下：

```
./build.sh -e ascend -S on -j16
```

MindSpore 编译过程较为缓慢，可能需要数小时，建议读者耐心等待。编译伊始，会下载编译 MindSpore 的依赖，此时应保持网络通畅。编译完成后，输出的结果如图 5-6 所示。

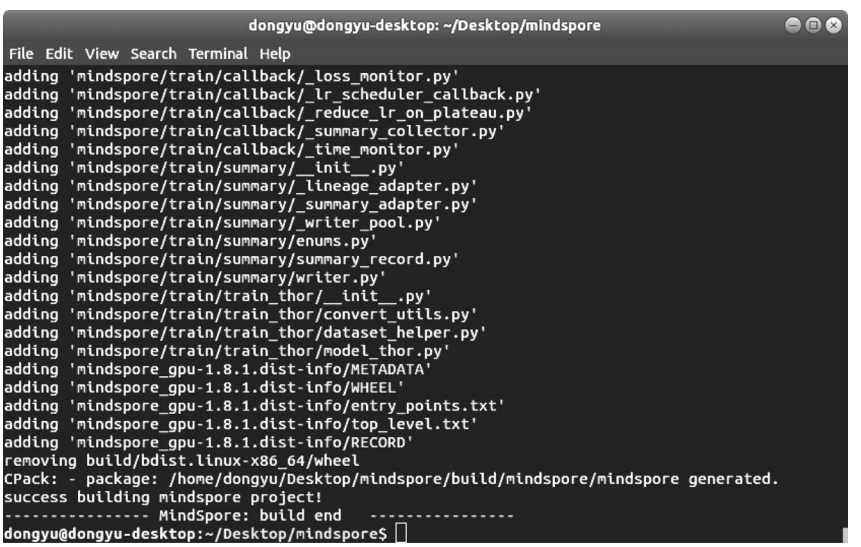

图 5-6 MindSpore 编译完成

进入 MindSpore 源代码目录中的 build/package 子目录，通过 pip 命令安装 MindSpore，命令如下：

```
#进入 MindSpore 的安装包目录，安装包也同时存在于 output 目录中
cd build/package
#安装 MindSpore
pip install mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl
```

安装期间会自动下载 NumPy、psutil、pillow、SciPy 等相关依赖，如果下载速度较慢，则可以使用国内的 pip 源。例如，使用清华大学的 pip 源的命令如下：

```
pip install mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl -i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装完成后整个 MindSpore 的环境搭建就完成了。通过 Python 脚本导入 MindSpore 包的方法可以判断 MindSpore 安装是否正确，命令如下：

```
python -c "import mindspore;mindspore.run_check()"
```

如果 MindSpore 已被正确安装，则输出的信息类似如下：

```
MindSpore version: 1.8.1
The result of multiplication calculation is correct, MindSpore has been installed successfully!
```

5.4　仓颉 TensorBoost 的环境配置

本节介绍仓颉 TensorBoost 的安装方法，并通过一个简单的例子检验仓颉 TensorBoost 是否安装成功。

5.4.1　仓颉 TensorBoost 的安装

在 0.38.2 版本之后的仓颉包已整合仓颉 TensorBoost，之前的版本需要单独解压 AI 包，命令如下：

```
unzip cangjie-v0.38.2-alpha.zip
unzip Cangjie_AI_0.38.2.zip
```

解压后，进入相应目录后通过 tar 命令解压相应架构的软件包，命令如下：

```
tar zxvf Cangjie-AI-0.38.2-x86_64-ubuntu18.04.tar.gz
tar zxvf Cangjie-0.38.2-linux_x64.tar.gz
```

将解压后的 cangjie_ai、cangjie 目录移动到当前用户的主目录（也可以是其他目录）中。如果解压到其他目录下，则应注意在配置～/.bashrc 时需要指定到相应的位置。

注意　如已经配置了仓颉语言的开发环境，则应确认仓颉语言的版本和仓颉 TensorBoost 的版本一致。

在仓颉 TensorBoost 软件包的 lib 子目录中新建 mindspore 目录，并将编译后的 MindSpore 源代码目录中的 build 目录和 config 目录复制到刚刚创建的 mindspore 目录中，命令如下：

```
mkdir -p ./cangjie_ai/lib/mindspore
cd ./cangjie_ai/lib/mindspore
cp -r <编译后的 MindSpore 源代码目录>/build .
cp -r <编译后的 MindSpore 源代码目录>/config .
```

在上述命令中将加粗部分修改为编译后的 MindSpore 源代码目录。

在～/.bashrc 文件中进行仓颉 TensorBoost 的环境变量配置，在每次用户登录时自动配置仓颉 TensorBoost 和仓颉语言环境，需要添加的内容如下：

```
source ～/cangjie/envsetupAI.sh
source ～/cangjie/envsetup.sh
```

此时，仓颉 TensorBoost 的安装已经完成。如果当前用户没有对应 Python 包目录(/usr/local/python3.7.5/lib/python3.7/site-packages)下的写入权限，就会回显如下错误：

```
please install mindspore-1.8 before running CangjieAI
```

这是因为我们没有 Python 包目录的写入权限，所以 MindSpore 包就会安装到当前用户主目录下的如下位置：

```
～/.local/lib/python3.7/site-packages/mindspore
```

此时，通过 pip 安装任意包时都会出现如下警告：

```
Defaulting to user installation because normal site-packages is not writeable
```

为了解决这个问题，可以修改 cangjie_ai 目录中的 envsetup.sh 文件，修改 MindSpore 安装目录变量的定义：

```
~~MINDSPORE_INSTALL_PATH=${PYTHON_BIN_PATH}/../lib/python3.7.5/site-packages/mindspore~~
MINDSPORE_INSTALL_PATH=～/.local/lib/python3.7/site-packages/mindspore
```

再次执行～/cangjie_ai/envsetup.sh 文件，即可解决这一问题。

5.4.2 检查仓颉 TensorBoost 是否安装成功

本节测试仓颉 TensorBoost 的安装是否完成，并介绍堆栈空间大小设置方法。

1. 通过矩阵(张量)的乘法判断仓颉 TensorBoost 是否安装成功

通过一个简单的程序判断仓颉 TensorBoost 是否安装成功。

【实例 5-1】 通过张量的方式实现矩阵的乘法，代码如下：

```
//code/chapter05/example5_1.cj
from CangjieAI import ops.*
from CangjieAI import common.*
main(): Int64
{
```

```
    let input_x = Tensor(Array<Float32>([1.0, 2.0, 3.0, 4.0]), shape:Array<Int64>([2,
2]))
    let input_y = Tensor(Array<Float32>([1.0, 2.0, 3.0, 4.0]), shape:Array<Int64>([2,
2]))
    print(matmul(input_x, input_y))
    print("Cangjie AI run test success!\n")
    return 0
}
```

关于张量的含义和应用在第 6 章中会详细介绍，这里的程序仅用于测试环境配置是否正确。在 examples_1.cj 源代码所在目录中通过以下命令编译：

```
cjc --enable-ad --int-overflow=wrapping -lcangjie_ai \
    -lmindspore_wrapper ./example5_1.cj -o ./main
```

上述命令参数的具体含义如下。

- --enable-ad：启用仓颉的自动微分能力。
- --int-overflow=wrapping：将运算溢出模式设置为 wrapping。
- -lcangjie_ai：使用仓颉 TensorBoost 库。
- -lmindspore_wrapper：使用 MindSpore Wrapper 库。
- ./examples5_1.cj -o ./main：编译 examples5_1.cj 代码，输出为 main 程序。

编译完成后，执行编译后的 main 可执行程序，命令如下：

```
./main
```

如果仓颉 TensorBoost 环境配置正常，则输出的结果如下：

```
Tensor(shape=[2, 2], dtype=Float32, value=
[[7.00000000e+00 1.00000000e+01]
[1.50000000e+01 2.20000000e+01]])
Cangjie AI run test success!
```

2. 仓颉程序的堆栈空间大小设置

仓颉程序默认的栈空间大小为 1MB，堆空间大小为 256MB。由于仓颉 TensorBoost 程序通常伴随着大量的程序，因此可以通过环境变量的配置修改堆栈的空间大小。在～/.bashrc 文件中进行环境变量配置，在末尾添加的命令如下：

```
export CJSTACKSIZE=100MB        #栈空间设置为 100MB
export CJHEAPSIZE=16GB          #堆空间设置为 16GB
```

加粗字体部分可以根据开发者的需求修改。

注意　仓颉程序的栈空间范围可以配置为 64KB～1GB，堆空间范围可以配置为 4MB～16GB，并且配置的数值需要小于物理内存大小。

在之后的学习中，如果出现内存溢出的情况，开发者则可以尝试扩大堆栈空间来保证程序的正常运行。如果栈空间不足，则会在运行时提示 StackOverflowError 错误；如果堆空

间不足，则会在运行时提示 OutOfMemoryError。

当然，上述配置都是针对主板上的物理内存的分配空间的设置，显卡的内存大小也是影响仓颉 TensorBoost 程序运行的重要因素，并且无法升级显卡及通过软件配置的方式提高其分配的空间。建议读者尽可能选用内存更大的显卡，例如 NVIDIA TESLA P40 等计算专用卡。

5.5 环境配置中的常见问题

本节总结了环境配置中可能遇到的常见问题。

5.5.1 更新 Python 版本后终端无法正常打开

安装 Python 3.7.5 后，会遇到无法直接打开终端的情况。这是因为新编译的 Python 3.7.5 中缺少 GI 包。此时，读者可以在任意目录上右击，选择 Open in terminal 打开终端，但是这种方法治标不治本，下面介绍解决这一问题的方法。

1. 确定终端无法打开的原因

为了能够顺利解决问题，首先需要确认是否因为缺少 GI 包引起终端无法打开。在已经打开的终端打开终端，命令如下：

```
gnome-terminal
```

输出的结果如下：

```
Traceback (most recent call last):
  File "/usr/bin/gnome-terminal", line 9, in <module>
    from gi.repository import GLib, Gio
ModuleNotFoundError: No module named 'gi'
```

通过报错信息 ModuleNotFoundError: No module named 'gi'可以发现确实是没有发现 GI 包导致的错误。如果报错信息与上文不同，则说明是其他原因，读者可从互联网上寻求帮助，不要按照本书的解决方法进行处理。

2. 问题的解决方法

打开 GI 包的目录，命令如下：

```
cd /usr/lib/python3/dist-packages/gi/
```

由于原本的 Python 版本为 3.6，所以需要将 GI 包的版本切换为 3.7。直接对两个涉及版本的文件名进行修改即可，命令如下：

```
sudo mv _gi_cairo.cpython-36m-x86_64-linux-gnu.so \
        _gi_cairo.cpython-37m-x86_64-linux-gnu.so
sudo mv _gi.cpython-36m-x86_64-linux-gnu.so \
        _gi.cpython-37m-x86_64-linux-gnu.so
```

注意黑体代码中的版本号的替换，将文件名中的 36 改为 37。如果当前 Python 版本为

其他版本，如 3.5 版本，则将 35 改为 37 即可。

随后，将 GI 包目录复制到新编译后的 Python 3.7.5 的相应安装目录中即可，命令如下：

```
sudo cp -rf /usr/lib/python3/dist-packages/gi/ \
        /usr/local/python3.7.5/lib/python3.7
sudo cp -rf /usr/lib/python3/dist-packages/gi/ \
        /usr/local/python3.7.5/site-packages
```

此时即可正常打开终端了。

5.5.2　编译 MindSpore 时出现 OpenMPI 编译错误

编译 MindSpore 时可能会出现 OpenMPI 编译错误并终止程序，错误信息输出如下：

```
 ================================================
Open MPI autogen: completed successfully. w00t!
 ================================================

checking whether  -- prefix = /home/dongyu/Desktop/dependency/mindspore/build/mindspore/.
mslib/ompi_0c49acaf25c2c0e3d530c0461bc6c004 is declared … ./configure: line 4308: 5: Bad
file descriptor./configure: line 4310: ${+y}: bad substitution
CMake Error at cmake/utils.cmake:179 (message):
  error! when ./configure;CXXFLAGS = -D_FORTIFY_SOURCE=2
  -O2; --prefix = /home/dongyu/Desktop/dependency/mindspore/build/mindspore/.mslib/ompi_
0c49acaf25c2c0e3d530c0461bc6c004
  in /home/dongyu/Desktop/dependency/mindspore/build/mindspore/_deps/ompi-src
Call Stack (most recent call first):
  cmake/utils.cmake:393 (__exec_cmd)
  cmake/external_libs/ompi.cmake:10 (mindspore_add_pkg)
  cmake/mind_expression.cmake:42 (include)
  CMakeLists.txt:54 (include)
```

这是由于下载 OpenMPI 时源文件发现了变更，从而导致出现此问题。此时，可以更改 OpenMPI 的下载源解决整个问题，修改 MindSpore 源代码目录中的 cmake/external_libs/ompi.cmake 文件，按照删除线和黑体部分内容进行修改：

```
if(ENABLE_GITEE)
    ~~set(REQ_URL "https://gitee.com/mirrors/ompi/repository/archive/v4.0.3.tar.gz")~~
    ~~set(MD5 "70f764c26ab6cd99487d58be0cd8c409")~~
    set(REQ_URL "https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-4.0.3.
tar.gz")
    set(MD5 "f4be54a4358a536ec2cdc694c7200f0b")

else()
    ~~set(REQ_URL "https://github.com/open-mpi/ompi/archive/v4.0.3.tar.gz")~~
    ~~set(MD5 "86cb724e8fe71741ad3be4e7927928a2")~~
    set(REQ_URL "https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-4.0.3.
tar.gz")
    set(MD5 "f4be54a4358a536ec2cdc694c7200f0b")
```

```
endif()

set(ompi_CXXFLAGS " - D_FORTIFY_SOURCE = 2  - O2")
mindspore_add_pkg(ompi
        VER 4.0.3
        LIBS mpi
        URL ${REQ_URL}
        MD5 ${MD5}
        PRE_CONFIGURE_COMMAND ./configure
        CONFIGURE_COMMAND ./configure)
include_directories( ${ompi_INC})
add_library(mindspore::ompi ALIAS ompi::mpi)
```

在默认情况下,MindSpore 编译程序会使用 GitHub 源下载依赖源文件。在上述代码中,对 GitHub 源和 gitee 源的相关内容都进行了修改。如果读者确认使用了 gitee 源,则只需对 if(ENABLE_GITEE)分支下的内容进行修改。再次编译 MindSpore 即可解决上述问题。

5.5.3 eigen 包下载失败

前文提到,在编译 MindSpore 时会首先下载并编译相关依赖,包括 grpc、isl、nccl 等。由于网络环境的不确定性,以后还可能会存在类似的问题,读者可以在源代码的 cmake/external_libs 目录下找到相关依赖的配置文件进行定制化修改。这些依赖都会被下载到 build/mindspore/_dep 目录中。读者也可以直接将已经被编译的_dep 目录复制到需要编译的 MindSpore 源代码的相应目录中,用于解决相关问题。这里以 eigen 包为例介绍相应的处理办法。

无论选择 GitHub 还是 gitee 作为 MindSpore 依赖的下载源,在默认情况下,eigen 包是通过 https://gitlab.com/libeigen/eigen/-/archive/3.3.9/eigen-3.3.9.tar.gz 地址下载的,更加容易出现下载失败问题,可能出现的提示如下:

```
[ 22 % ] Performing download step (download, verify and extract) for 'eigen3 - populate'
-- Downloading…
   dst = '/home/dongyu/Desktop/mindspore/build/mindspore/_deps/eigen3 - subbuild/eigen3 -
populate - prefix/src/eigen - 3.3.9.tar.gz'
   timeout = 'none'
-- Using src = 'https://gitlab.com/libeigen/eigen/ - /archive/3.3.9/eigen - 3.3.9.tar.gz'
-- verifying file…
       file = '/home/dongyu/Desktop/mindspore/build/mindspore/_deps/eigen3 - subbuild/eigen3
- populate - prefix/src/eigen - 3.3.9.tar.gz'
-- MD5 hash of
     /home/dongyu/Desktop/mindspore/build/mindspore/_ deps/eigen3  -  subbuild/eigen3  -
populate - prefix/src/eigen - 3.3.9.tar.gz
  does not match expected value
    expected: '609286804b0f79be622ccf7f9ff2b660'
      actual: '4a4778ea63911c79a86ac1eab22076d4'
-- Hash mismatch, removing…
-- Retrying…
```

这就是因为下载失败使 eigen-3.3.9.tar.gz 文件不完整或者损坏，导致了 Hash 校验错误。此时，读者可以自行下载（或者在本书配套资源中找到）eigen-3.3.9.tar.gz 文件并放置到 MindSpore 源码目录下的/build/mindspore/_deps/eigen3-subbuild/eigen3-populate-prefix/src/位置，替换下载错误的 eigen-3.3.9.tar.gz 文件。随后，重新编译即可解决问题。

5.5.4　通过 SSH 和 Samba 服务在 Windows 环境中开发仓颉 TensorBoost 程序

许多开发者习惯于在 Windows 环境中开发仓颉程序，本节介绍在 Windows 系统中使用 Ubuntu 终端及在 Windows 系统中访问 Ubuntu 文件的基本方法。在进行如下操作前，需要保证 Windows 和 Ubuntu 主机在同一个网络下。

1. 在 Windows 系统中使用 Ubuntu 终端

（1）查询 Ubuntu 主机的 IP 地址，命令如下：

```
ip addr
```

此时会弹出当前的网络配置信息，如图 5-7 所示。

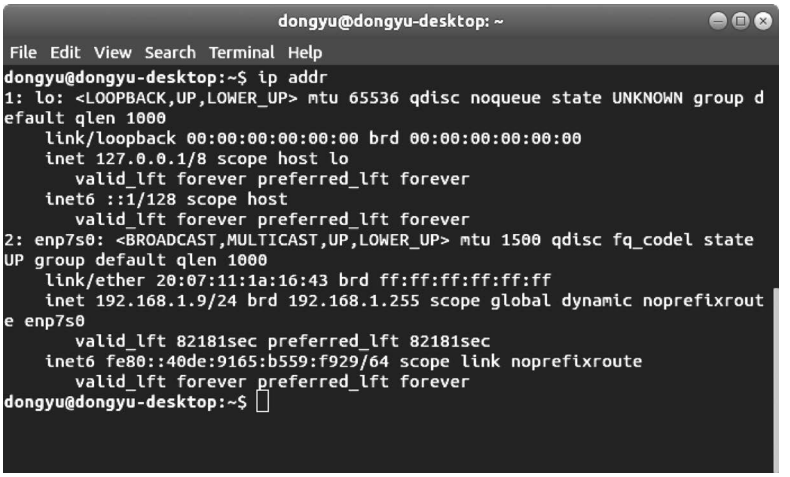

图 5-7　查看主机网络配置信息

图 5-7 中显示了主机的 IP 地址为 192.168.1.9。这个 IP 地址会根据读者使用的网络环境的不同而不同。

（2）安装 SSH 工具。在 Ubuntu 终端中下载并安装 SSH，命令如下：

```
sudo apt install ssh -y
```

输入当前用户名的密码后即可安装 SSH。安装完成后，Ubuntu 会自动启动 SSH 服务，通过以下命令可以查询 SSH 的运行状态：

```
systemctl status sshd
```

如果在回显的信息中找到 active (Running)提示，则说明 SSH 服务正常运行。

(3) 在 Windows 环境中，通过 ssh 命令即可访问 Ubuntu 主机，命令如下：

```
ssh 192.168.1.9 -l <用户名>
```

其中，IP 地址需要按实际情况进行替换，并且把“<用户名>”替换为 Ubuntu 主机的可用用户名。命令执行后，根据提示输入密码即可使用 Ubuntu 中的终端，如图 5-8 所示。

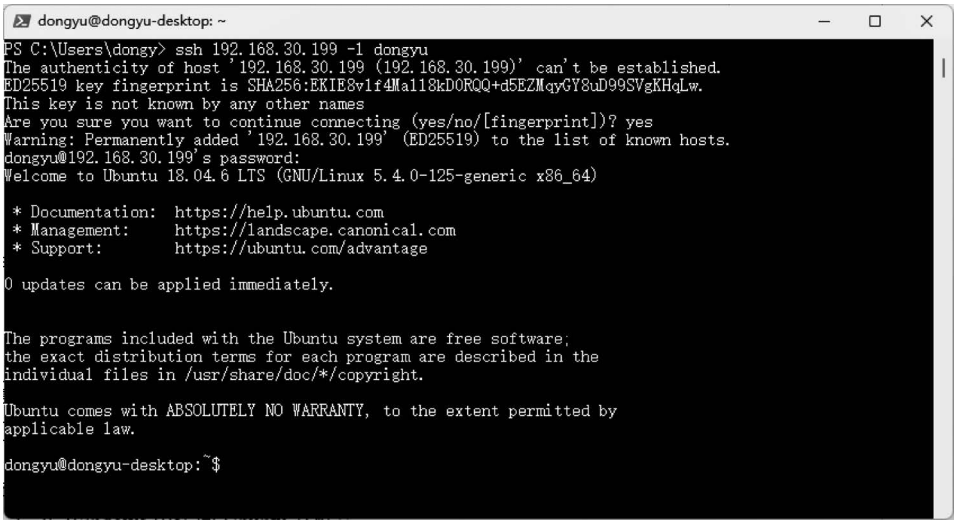

图 5-8 通过 SSH 远程连接 Ubuntu 主机

注意 首次连接时，会提示授权信息为建立，并提示 Are you sure you want to continue connecting，此时输入 yes 后回车即可正常连接。

读者还可以选择在 VSC 等集成开发环境(IDE)中使用 Ubuntu 终端，在同一个 IDE 中进行代码编辑和编译，操作会更加方便。

2. 在 Windows 系统中访问 Ubuntu 文件

为了能够远程访问 Ubuntu 文件，方便进行代码的编辑操作，可通过 Samba 的方式将 Ubuntu 的目录共享出来，以便于在 Windows 系统中访问其目录中的内容。具体步骤如下。

(1) 安装 Samba。在命令行工具中安装 Samba，命令如下：

```
sudo apt install samba -y
```

(2) 查询 Samba 是否正常运行，命令如下：

```
systemctl status smbd
```

如果 Samba 正常运行，则可见 active(running)提示，如图 5-9 所示。

(3) 在/etc/samba/smb.conf 文件的末尾添加当前用户的主目录共享配置，代码如下：

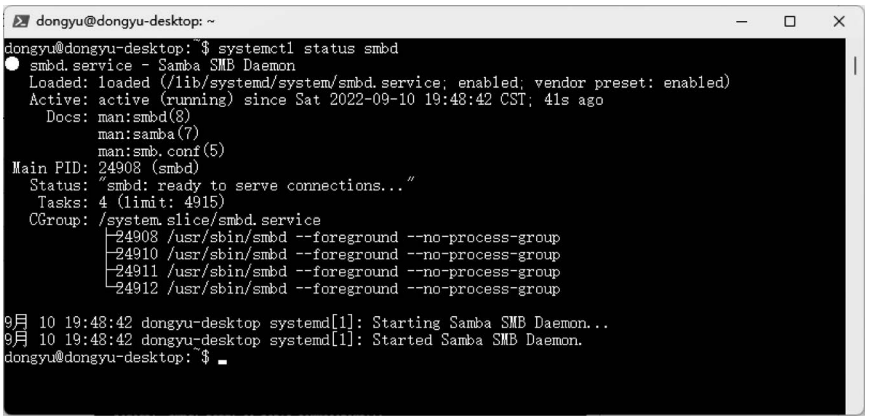

图 5-9　Samba 服务已经启动

```
[cangjie]          //共享目录名称
   comment = cangjie ai directory
   browseable = yes
   path = /home/dongyu        #共享目录位置,可根据实际情况修改
   create mask = 0777
   directory mask = 0777
   valid users = dongyu       #用户名需要和稍后添加的 Samba 用户名相同
   public = yes
   available = yes
   read only = no
   writable = yes
```

这里将共享目录名设置为 cangjie,在 Windows 连接时需要用到该名称。另外,需要注意根据上面的注释对共享目录和用户名进行调整。

(4) 添加一个 Samba 用户。这里的用户名为 dongyu,命令如下:

```
sudo smbpasswd -a dongyu
```

根据提示输入密码后确认即可,如图 5-10 所示。

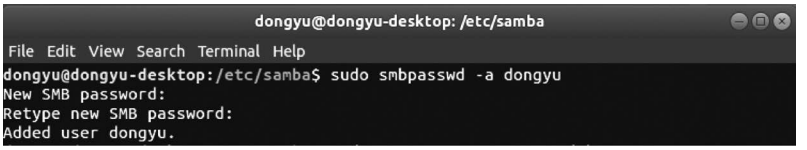

图 5-10　添加 Samba 用户

(5) 重启 Samba 服务,命令如下:

```
sudo systemctl restart smbd
```

(6) 在 Windows 环境中通过 IP 地址访问共享目录。在 Windows 11 中,右击桌面上的“此计算机”,选择“映射网络驱动器”选项,如图 5-11 所示。

图 5-11　映射网络驱动器

在弹出的映射网络驱动器对话框中，在“驱动器”选项中选择合适的驱动器盘符(这里选择 Z:)，在“文件夹”中输入\\192.168.1.199\cangjie，如图 5-12 所示。注意这里的 IP 和目录名要和之前的 IP 和配置信息一致；共享目录名 cangjie 需要和配置时一致，其他选项保持默认，单击“完成”按钮。

在弹出的“输入网络凭据”对话框中键入 Samba 用户名和密码后，即可访问 Ubuntu 的共享目录，如图 5-13 所示。

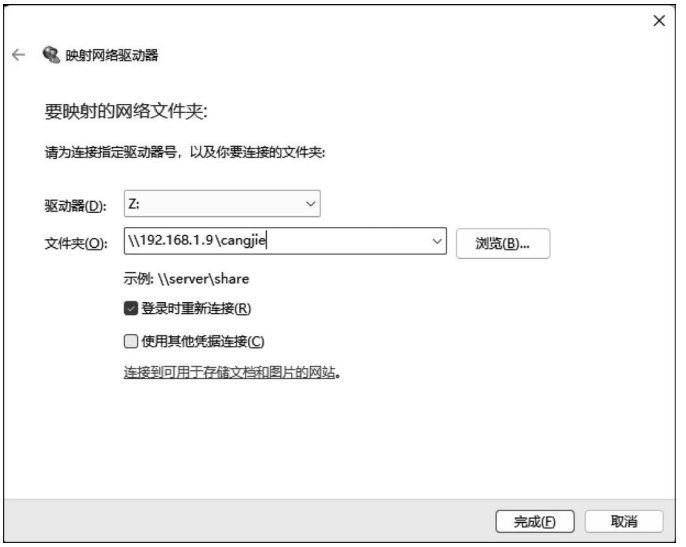

图 5-12　指定网络驱动器的位置

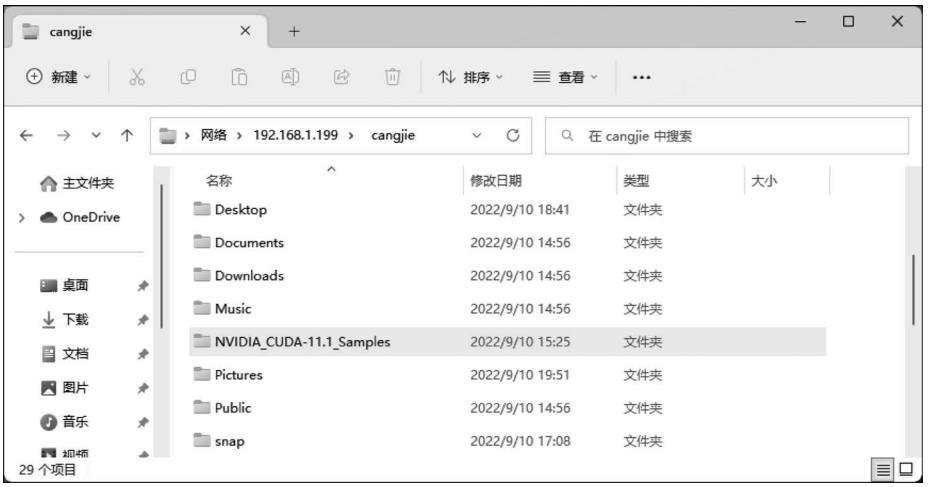

图 5-13　Ubuntu 的共享目录

此时，就可以通过一个固定的盘符（Z:）传递数据和文件了，可以通过 VSC 打开 Ubuntu 共享目录中的代码进行编辑。

5.6　本章小结

本章介绍了仓颉 TensorBoost 的完整配置方法，并列举了编译过程中几个可能出现的常见问题。编译的过程可能比较枯燥，可能需要若干小时的努力。对于初学者，利用几天时间研究编译过程也是正常的。

在随后的几章中都会利用这个仓颉 TensorBoost 开发环境。建议读者保持计算机的独立环境，防止开发环境遭到破坏。同时建议对全盘进行备份，至少对 MindSpore 的编译目录进行备份，以便于在系统出现问题后可以快速地恢复开发环境。

5.7　习题

1. 在 Ubuntu 18.04 操作系统中，配置仓颉 TensorBoost 的开发环境。
2. 通过矩阵（张量）的乘法程序判断仓颉 TensorBoost 的开发环境是否配置成功。
3. 查阅 MindSpore 的相关资料，思考 MindSpore 和仓颉 TensorBoost 之间的关系。