Tutorial-XSCOW

人工智能（AI）的迅猛发展正深刻影响着学术界和工业界。AI技术的进步依赖于处理海量数据和复杂模型的能力，因此，高性能计算（HPC）平台成为推动AI研究和应用的关键工具。尽管HPC在技术上提供了强大的支持，其使用的复杂性却给学术研究人员和工业从业者带来了挑战。许多研究人员和工程师面临着平台配置繁琐、资源调度复杂以及编程模型不友好的问题，这些因素可能延缓AI项目的开发进程。

XSCOW 是基于 SCOW 和 CraneSched 研发的算力网络融合与交易平台，通过XSCOW，超算用户无需安装任何软件，只需使用现代浏览器即可高效利用超算资源完成计算任务。用户可以通过网页界面进行作业提交、文件管理、终端调用、用户管理等多项操作，极大降低了使用门槛。

本教程通过一系列在XSCOW上运行AI的案例，帮助用户快速掌握在HPC环境中进行AI学习和研究的方法，助力学术界和工业界更高效地利用高性能计算资源。

下面我们首先介绍如何在 XSCOW 平台上申请计算资源，然后通过简单案例 Tutorial 0 介绍如何在 XSCOW 平台上进行计算，最后是 AI 相关的教程介绍。

XSCOW 平台申请计算资源

1. 登陆

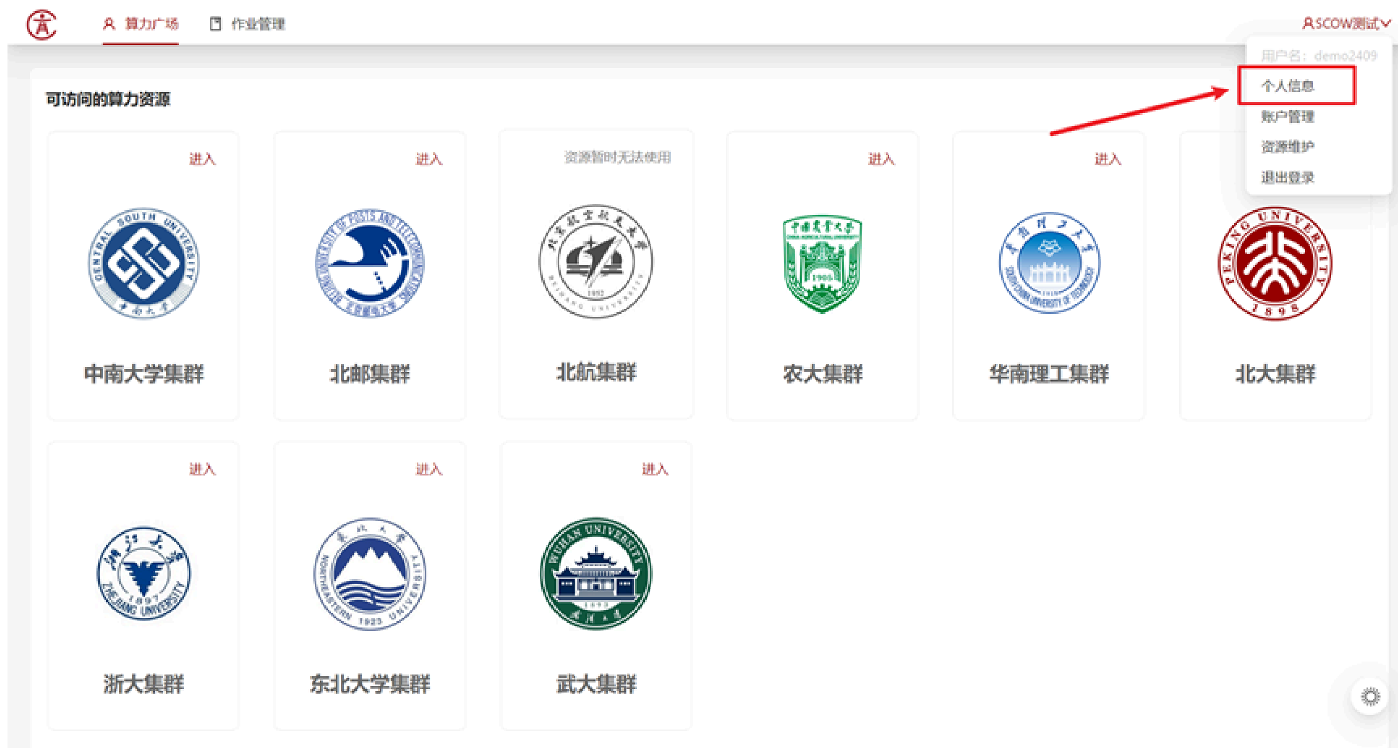
我们可以通过如下网址访问：

https://aigc.emic.edu.cn/

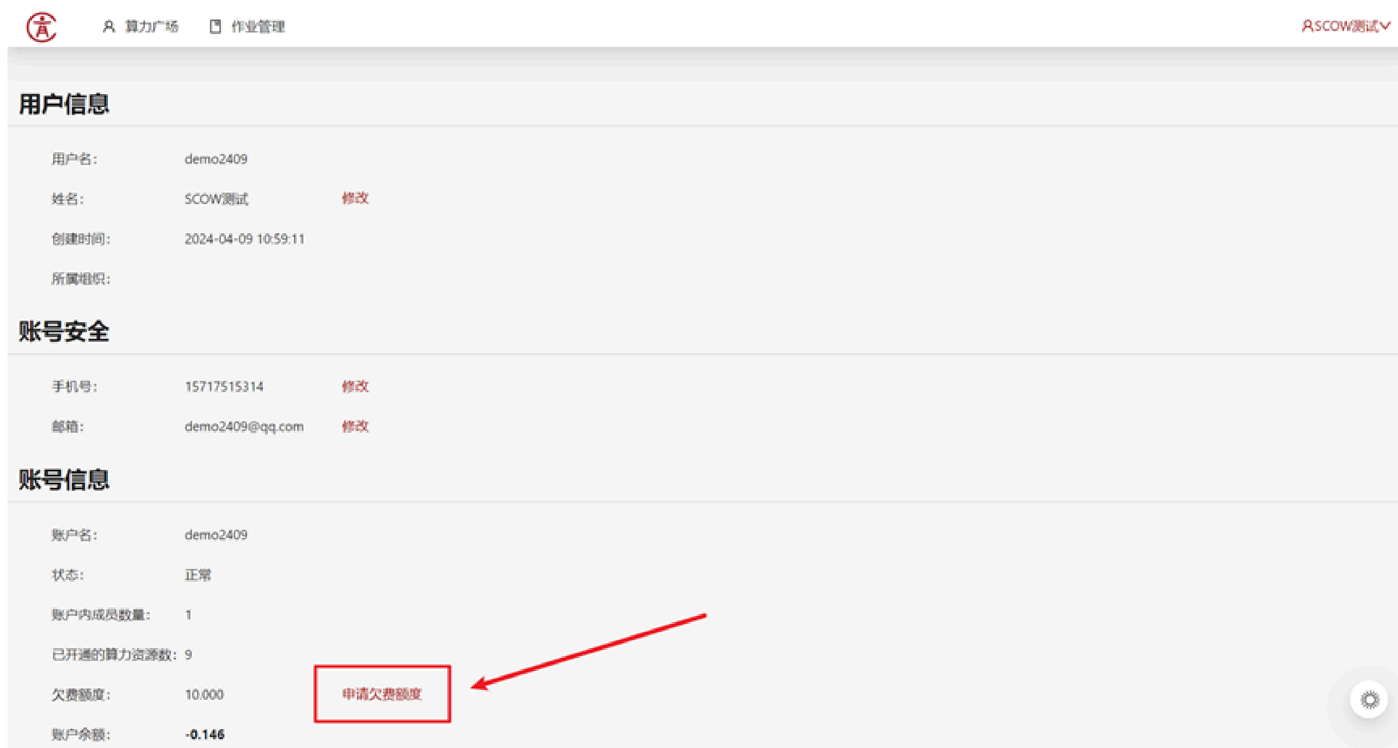
登陆后，从页面最下方的 “智算合作” 可进入资源访问，可选择任一合作节点。

在教育部大模型公共服务平台中，普通用户首次登录时，需要完成一个“申请欠费额度”的简单步骤，这个步骤完成后，便可以开始免费使用平台中列出的所有集群资源了。

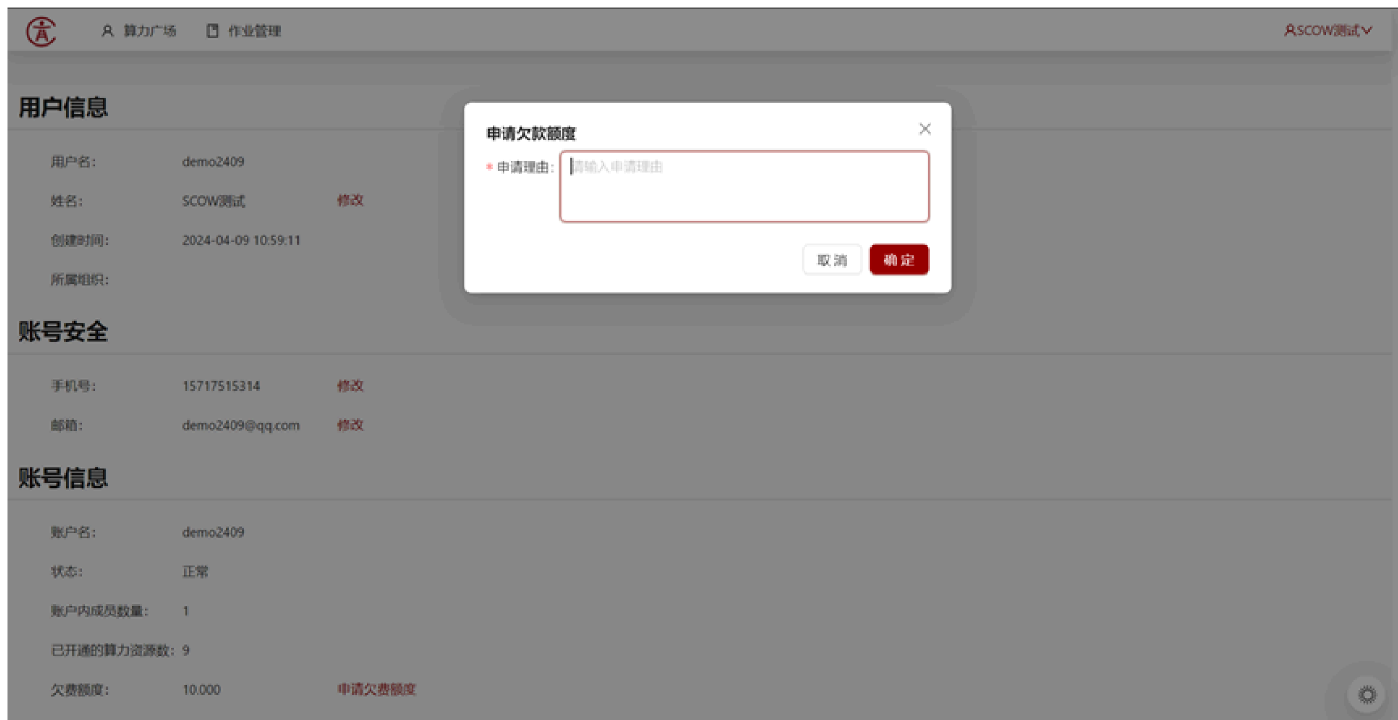
普通用户在登录教育部大模型公共服务平台平台后，点击右上角的账户名，会出现下拉菜单。选择下拉菜单中的“个人信息”，点击进入。



点击“个人信息”后，能看到如下的页面。点击“申请欠费额度”：

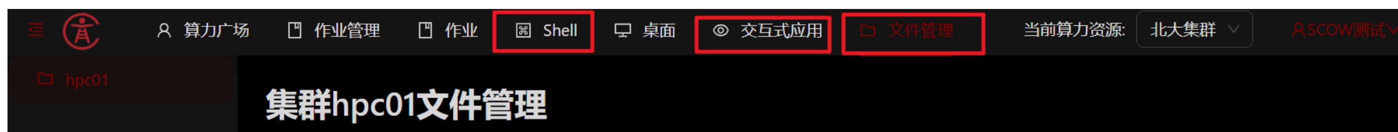


点击“申请欠费额度”后，将出现以下的弹窗。在弹窗中，填写申请欠费额度的理由后，点击“确定”，则成功提交了欠费申请。



提交了欠费额度申请后，等管理员批准申请。成功批准后，用户就可以开始免费试用平台上的集群计算资源了！

我们以 “北京大学高性能计算中心” 为例，进入后显示 dashboard 界面：



其中 “shell” 可用于打开命令行窗口：在 login 节点的命令行窗口可用于直接通过队列系统提交任务；在 data 节点的命令行窗口可用于数据传输和连接网络。

“交互式应用” 可以打开桌面窗口、Matlab、RStudio、Jupyter notebook、Jupter Lab 等交互式应用。

“文件管理” 可用于管理、上传、下载文件。

我们的教程是使用 Jupyter Lab 运行的，所以需要在 “交互式应用” 中创建 Jupter Lab 应用。但在创建 Jupter Lab 应用之前，需要先在 “shell” 中安装 jupyter，才能成功创建。

2. 安装 Conda

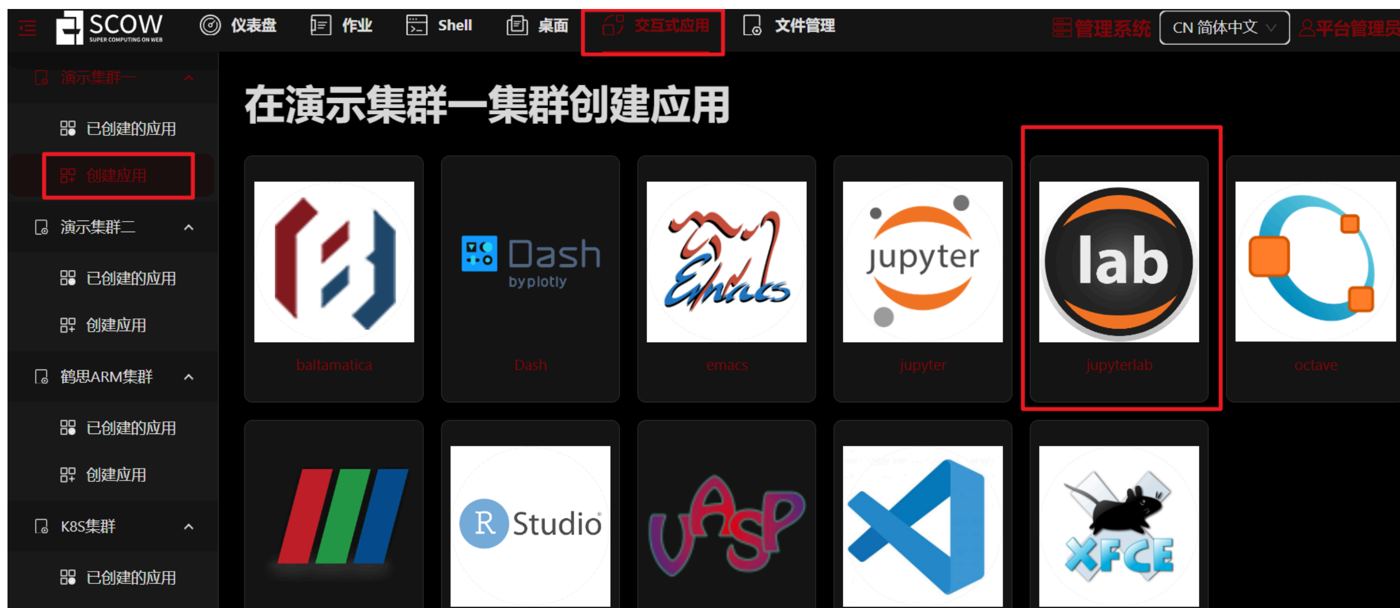
在联网的数据节点运行下面命令安装 conda

```
# 1. 获得最新的miniconda安装包；
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

# 2. 安装到自己的HOME目录下software/miniconda3中，这个目录在安装前不能存在，否则会报错；
sh Miniconda3-latest-Linux-x86_64.sh -b -p ${HOME}/software/miniconda3

# 3. 安装成功后可以删除安装包，节省存储空间
rm -f Miniconda3-latest-Linux-x86_64.sh

# 4. 将环境变量写入~/.bashrc文件中；(下面这句话，添加到~/.bashrc文件中)
export PATH=${HOME}/software/miniconda3/bin:$PATH

# 5. 退出重新登录或者执行以下命令，即可导入 conda 环境
source ~/.bashrc

# 6. 检查是否安装成功
conda --version
```

创建conda环境并安装 Jupyter

```
conda create -n tutorial0 python
conda activate tutorial0
pip install notebook jupyterlab
```

3. 创建 Jupyter Lab 应用 点击 dashboard 上的 “交互式应用”，点击 “创建应用”, 点击 “JupyterLab”，填写相应的资源，点击最下方的 “提交”，进行创建。



SCOW 仪表盘 作业 Shell 桌面 交互式应用 文件管理

演示集群一
已创建的应用
创建应用
演示集群二
已创建的应用
创建应用
鹤思ARM集群
已创建的应用
创建应用
K8S集群
已创建的应用
创建应用
统信集群
已创建的应用

创建jupyterlab

* 作业名：jupyterlab-20240920-123824
* 账户：a_admin
* 分区：gpu
* QOS：normal
* 节点数：1
* 单节点GPU卡数：1
* 最长运行时间：60 分钟

其他sbatch参数：比如：--gpus gres:2 --time 10

总GPU卡数：1　　总CPU核心数：4　　总内存容量：

取 消　提 交

创建成功后显示 “Running”，点击 “连接” 进入

在 XSCOW 平台上运行 Tutorial 0

1. 获取教程所需文件

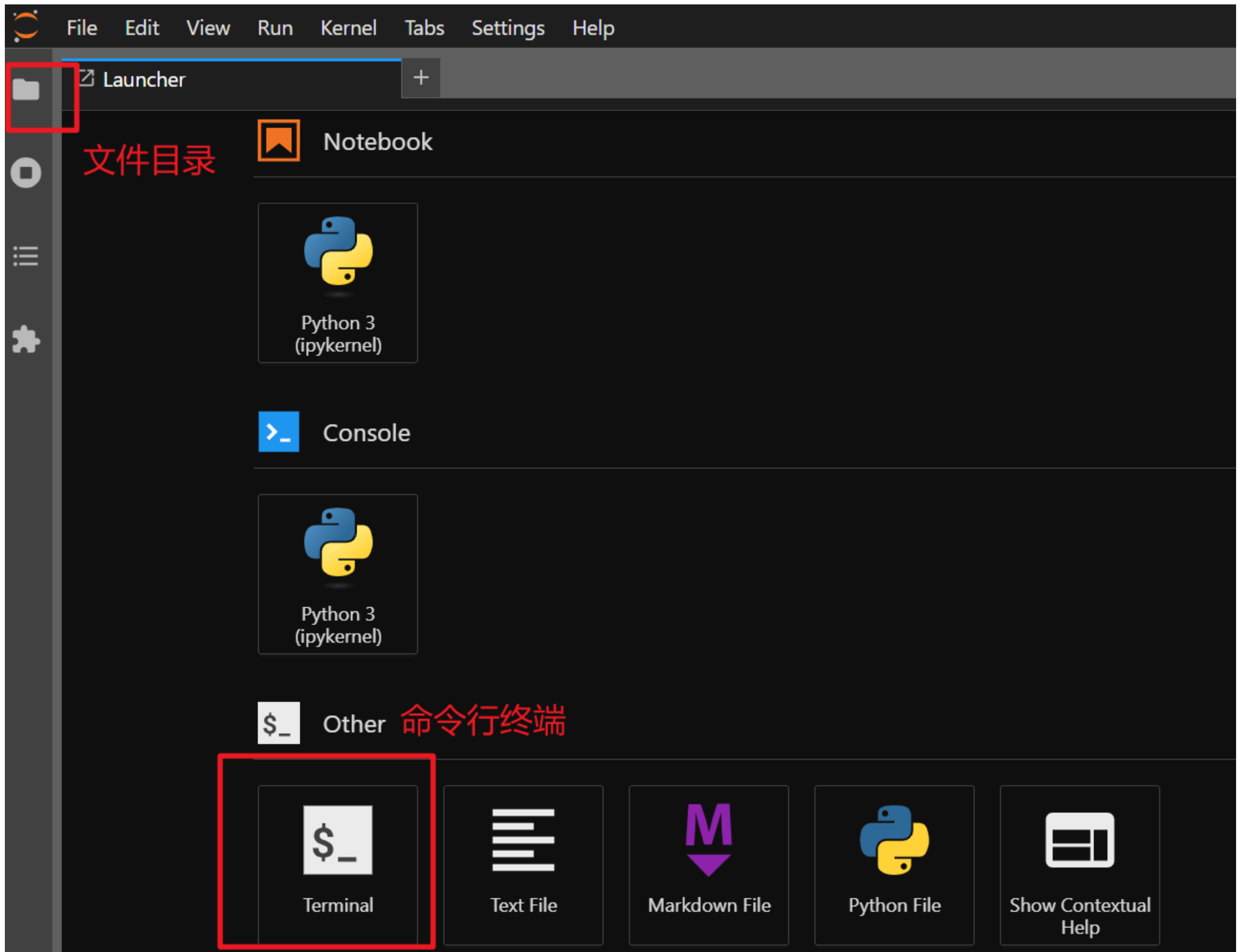
```
wget https://www.pkuscow.com/tutorial/scow/tutorial.tar.gz
tar -xzf tutorial.tar.gz
```

2. 运行 Tutorial 0

在所下载的教程文件夹中打开 tutorial/Tutorial0_hello_world/tutorial0_hello_world.ipynb 文件。可以看到文件中有 markdown 单元格和 python 代码单元格。用鼠标点击选中单元格后，“Ctrl + Enter” 可运行单元格，markdown 在运行后起到渲染的效果，python 会在下方输出结果。注意：苹果电脑使用的快捷键会有所不同。

3. Jupyter Lab 中使用命令行

除了在 SCOW 中使用 shell 外，还可以使用 Jupyter Lab 提供的命令行终端。



数据与模型路径

部分集群中, 数据和模型已经提前下载好在公用存储下：

- 数据 `/lustre/public/tutorial/data`
- 模型 `/lustre/public/tutorial/models`

如果您所用的集群中不含上述目录，则需按后续教程中的提示进行下载

教程内容

教程目前由 9 个独立的案例构成：

Pytorch 基础

- **Tutorial1**: 通过预测房价这一简单案例展示如何使用全连接神经网络解决回归问题，并在单机单显卡上运行案例。

CV 相关

- **Tutorial2**: 通过MNIST数据集和一个规模较小的简单CNN网络展示使用CNN进行图像分类的简单案例。
- **Tutorial3**: 实际应用和研究中通常会使用大型数据集和多卡并行，这部分使用著名的ResNet50网络和ImageNet数据集，展示在多张显卡上并行的图像分类任务。

大模型相关

- **Tutorial4**: 通过在 SCOW 平台上运行 bge-m3 模型，展示 embedding 模型的推理任务。
- **Tutorial5**: 在 SCOW 平台上运行 bge-reranker-v2-m3。
- **Tutorial6**: 通过 Qwen2-7B-Instruct 模型，展示大模型的推理、微调、合并。
- **Tutorial7**: Qwen2-72B-Instruct-GPTQ-Int4 模型的推理。
- **Tutorial8**: 在多张显卡上运行 Qwen2-72B-Instruct 模型。
- **Tutorial9**: stable-diffusion-3-medium 文生图任务，通过 prompt 提示生成对应的图片。

作者: 黎颖; 龙汀汀

联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

测试由 褚苙扬（cly2412307718@stu.pku.edu.cn）同学完成

Tutorial0: Hello World

本教程运行简单的 hello world 命令。

In [1]:
```
print("Hello World")
```

```
Hello World
```

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial1: 房价预测

本节旨在通过 kaggle 房价预测竞赛 这一简单案例展示如何使用全连接神经网络解决回归问题。

分以下几步来实现：

1. 环境安装与应用创建
2. 分步运行本文件

   2.1 数据加载和预处理

   2.2 构建网络

   2.3 训练与评估模型

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial1 python=3.9
conda activate tutorial1
pip install notebook jupyterlab torch==2.1.0 numpy==1.26.4
matplotlib==3.8.4 pandas==2.2.2 scikit-learn==1.5.0
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial1` , 硬件资源建议至少4个CPU核心。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 分步运行本文件

2.1 数据预处理

作为简化模型的案例，这里使用的是 kaggle 房价预测竞赛中的训练数据集。在后面的处理中，我们只使用了数值部分的特征，并把全部数据分为训练集和测试集两部分。

实验所用数据: train.csv

In [1]:
```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch

# 读取数据
# VAR_PLACEHOLDER
data = pd.read_csv('./data/train.csv')

# 去掉第一列编号
data = data.iloc[:, 1:]

# 只保留数值类型的数据
numeric_features = data.select_dtypes(include=[np.number])

# 处理缺失值
numeric_features.fillna(numeric_features.mean(), inplace=True)

# 分离特征和目标变量
X = numeric_features.drop('SalePrice', axis=1).values
y = numeric_features['SalePrice'].values

# 切分数据为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# 标准化特征
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 转换为torch张量
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)
y_test = torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)
```

2.2 构建模型

使用一个全连接层加 relu 激活层作为示例，更复杂的网络可以通过修改 Net 类来实现。

In [2]:
```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self, input_features):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_features, 128)
        self.fc2 = nn.Linear(128, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x) # 回归任务不用激活函数
        return x
```

2.3 训练与评估

由于房价之间的差异巨大，评估预测是否准确时应该考虑相对值的变化，所以使用下面的评估函数：

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log \hat{y}_i)^2}$$

训练模型与评估模型都在这部分

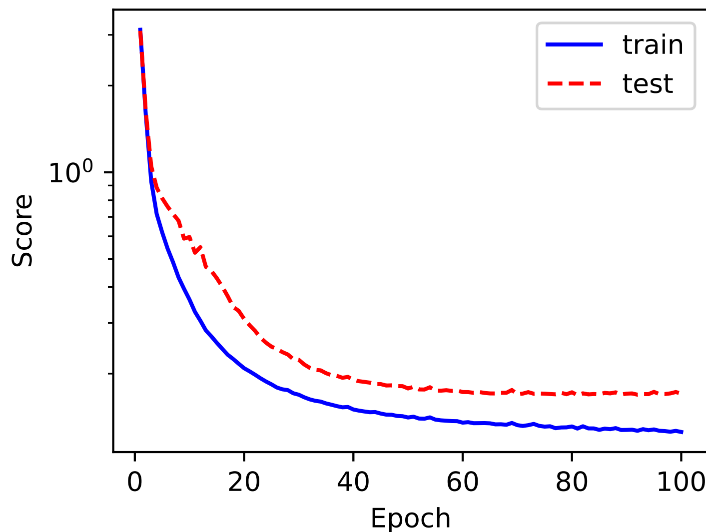
In [ ]:
```
from torch.utils.data import DataLoader, TensorDataset

# 参数设置
learning_rate, weight_decay, epochs, batch_size = 0.1, 5, 100, 32

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 实例化模型
model = Net(X_train.shape[1]).to(device)

# 损失函数
criterion = nn.MSELoss()

# 优化器
optimizer = torch.optim.Adam(model.parameters(), lr= learning_rate, weight_c

# loss 评估
def score(model, X, y):
    pred = torch.clamp(model(X), 1, float('inf'))
    score = torch.sqrt(criterion(torch.log(pred), torch.log(y)))

    return score.item()

# 训练模型
train_dataset = TensorDataset(X_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True

train_ls, test_ls = [], []
for epoch in range(epochs):
    model.train()
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        # 前向传播
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # 反向传播和优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # 模型评估
    model.eval()
```

```
    with torch.no_grad():
        train_ls.append(score(model, X_train, y_train))
        test_ls.append(score(model, X_test, y_test))
```

In [4]:

```
from matplotlib import pyplot as plt
from matplotlib_inline import backend_inline

backend_inline.set_matplotlib_formats('svg')

plt.rcParams['figure.figsize'] = (4, 3)

plt.plot(list(range(1, epochs + 1)), train_ls, 'b', label='train')
plt.plot(list(range(1, epochs + 1)), test_ls, 'r--', label='test')
plt.xlabel("Epoch")
plt.ylabel("Score")
plt.yscale('log')
plt.xlim([1, epochs])

plt.legend()
plt.grid()
```

Out[4]: <matplotlib.legend.Legend at 0x7fefdf252430>



> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial2: 图像分类1

本节旨在展示使用CNN进行图像分类的简单案例，使用MNIST数据集和一个规模较小的简单CNN网络。

分以下几步来实现：

1. 环境安装与应用创建
2. 分步运行本文件

   2.1 数据加载和预处理

   2.2 定义 CNN 模型

   2.3 训练与评估

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial2 python=3.9
conda activate tutorial2
pip install notebook jupyterlab numpy==1.26.4 matplotlib==3.8.4
pip install torch==1.13.0+cu116 torchvision==0.14.0+cu116
torchaudio==0.13.0 --extra-index-url
https://download.pytorch.org/whl/cu116
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial2` , 硬件资源为1个GPU。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 分步运行本文件

2.1 数据预处理

MNIST 是一个手写数字数据集，包含了数字0到9的灰度图像。

In [ ]:
```
%tar -xzf data.tar.gz
```

In [1]:

```
import torchvision.transforms as transforms
import torchvision.datasets as datasets

# 定义数据转换方式: 标准化
transform = transforms.Compose([
    transforms.ToTensor(),  # 将图像转换为 PyTorch 张量
    transforms.Normalize((0.5,), (0.5,))  # 标准化（均值0.5，标准差0.5）
])

# 加载训练集和测试集
train_dataset = datasets.MNIST('./data', train=True, transform=transform)
test_dataset = datasets.MNIST('./data', train=False, transform=transform)
```

2.2 定义 CNN 模型

In [2]:

```
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1) # 卷积层：输入
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.mp = nn.MaxPool2d(kernel_size=2, stride=2) # 池化层：用于减小输出矩
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.mp(self.conv1(x)))
        x = self.relu(self.mp(self.conv2(x)))
        x = x.view(x.size(0), -1)  # 展平
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

2.3 训练与评估

完成模型训练和评估在 V100 显卡上需要约 6 min

In [3]:

```
from torch.utils.data import DataLoader
import torch
import time
from datetime import timedelta

# 参数设置
learning_rate, epochs, batch_size = 0.001, 20, 64
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Using device:", device)


# 实例化模型
model = CNN().to(device)
```

```python
# 损失函数
criterion = nn.CrossEntropyLoss()

# 优化器
optimizer = torch.optim.Adam(model.parameters(), lr= learning_rate)

# 评估函数
def accuracy(model, data_loader, device):
    total = 0
    correct = 0
    model.eval()
    for X, y in data_loader:
        X, y = X.to(device), y.to(device)
        outputs = model(X)
        _, predicted = torch.max(outputs.data, 1)
        total += y.size(0)
        correct += (predicted == y).sum().item()

    return 100 * correct / total

# 数据准备
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuf
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffl

# 训练模型
train_ls = []
train_acc, test_acc = [], []
for epoch in range(epochs):
    start_time = time.time()
    model.train()
    total_loss = 0

    for X, y in train_loader:
        X, y = X.to(device), y.to(device)

        # 前向传播
        outputs = model(X)
        loss = criterion(outputs, y)

        # 反向传播和优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # loss 记录
        total_loss += loss.item()
    average_loss = total_loss / len(train_loader)
    train_ls.append(average_loss)

    # 模型评估
    with torch.no_grad():
        train_acc.append(accuracy(model, train_loader, device))
        test_acc.append(accuracy(model, test_loader, device))

    # 计时
```

```
    end_time = time.time()
    print(f"{epoch}/{epochs}: accuracy: {test_acc[-1]} | time consuming: {ti
```

```
Using device: cuda
0/20: accuracy: 98.07 | time consuming: 0:00:20.484834
1/20: accuracy: 98.48 | time consuming: 0:00:17.870257
2/20: accuracy: 98.69 | time consuming: 0:00:17.827839
3/20: accuracy: 98.94 | time consuming: 0:00:17.885543
4/20: accuracy: 98.99 | time consuming: 0:00:18.206810
5/20: accuracy: 98.7 | time consuming: 0:00:18.379300
6/20: accuracy: 99.03 | time consuming: 0:00:18.142421
7/20: accuracy: 99.04 | time consuming: 0:00:17.860128
8/20: accuracy: 98.9 | time consuming: 0:00:17.888026
9/20: accuracy: 98.94 | time consuming: 0:00:18.103130
10/20: accuracy: 99.17 | time consuming: 0:00:17.779486
11/20: accuracy: 98.67 | time consuming: 0:00:18.168736
12/20: accuracy: 99.0 | time consuming: 0:00:17.885525
13/20: accuracy: 98.97 | time consuming: 0:00:18.084002
14/20: accuracy: 99.06 | time consuming: 0:00:17.851521
15/20: accuracy: 99.11 | time consuming: 0:00:17.829083
16/20: accuracy: 99.16 | time consuming: 0:00:17.777250
17/20: accuracy: 98.88 | time consuming: 0:00:17.824750
18/20: accuracy: 99.01 | time consuming: 0:00:17.837890
19/20: accuracy: 99.13 | time consuming: 0:00:17.837298
```

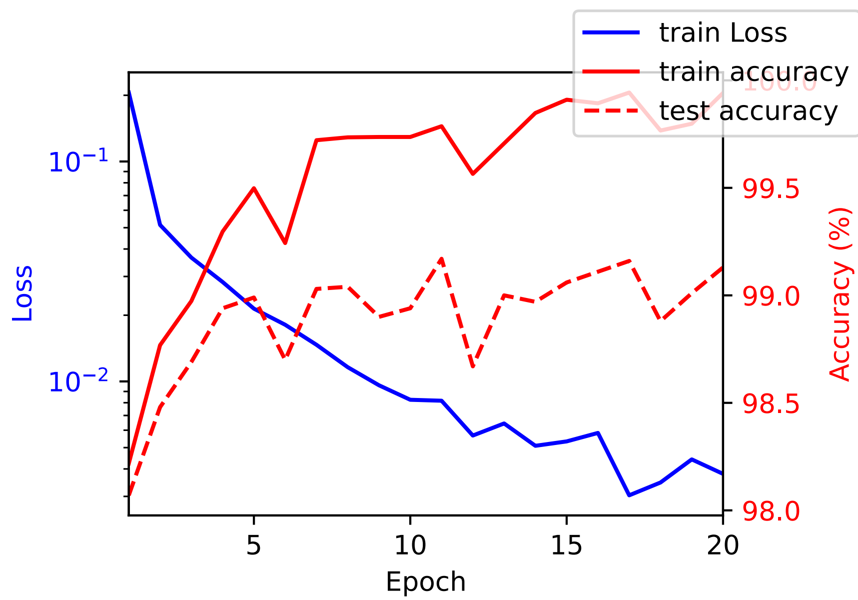
训练过程中 Loss 一直在下降，本次训练中最高能达到 99% 的准确率，实际上通过改进模型和训练过程并进行数据增强，CNN训练MNIST数据集的准确率可以进一步提升。

In [4]:

```
import matplotlib.pyplot as plt
from matplotlib_inline import backend_inline

backend_inline.set_matplotlib_formats('svg')

plt.rcParams['figure.figsize'] = (4, 3)

fig, ax1 = plt.subplots()

# loss
ax1.plot(list(range(1, epochs + 1)), train_ls, 'b-', label='train Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss', color='b')
ax1.tick_params(axis='y', labelcolor='b')
ax1.set_yscale('log')
ax1.set_xlim([1, epochs])

# Accuracy
ax2 = ax1.twinx()
ax2.plot(list(range(1, epochs + 1)), train_acc, 'r-', label='train accuracy'
ax2.plot(list(range(1, epochs + 1)), test_acc, 'r--', label='test accuracy')
ax2.set_ylabel('Accuracy (%)', color='r')
ax2.tick_params(axis='y', labelcolor='r')

fig.legend()
```

Out[4]:

```
<matplotlib.legend.Legend at 0x7f5cd07636d0>
```



> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial3: 图像分类2

本节旨在展示更接近实际的训练场景，使用ResNet50训练ImageNet数据集，在多块显卡上做并行。

分以下几步来实现：

1. 环境安装与应用创建
2. 分步运行本文件

   2.1 数据加载和预处理

   2.2 模型

   2.3 训练与评估

   2.4 加载模型

ImageNet 是一个大型的视觉数据库，由斯坦福大学的李飞飞（Fei-Fei Li）教授及其团队于2009年创建。ImageNet包含了1000个类别总计120万张训练图片，以及5万张验证图片。可以先尝试从公共目录（/lustre/public/tutorial/data/imagenet）加载 ImageNet 数据集，如果公共目录不存在，则需从 ImageNet 官网自行下载：https://image-net.org

（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ；使用参考：https://tmuxcheatsheet.com）

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial3 python=3.9
conda activate tutorial3
pip install notebook jupyterlab numpy==1.26.4 matplotlib==3.8.4
conda install pytorch torchvision torchaudio accelerate pytorch-cuda=12.1 -c pytorch -c nvidia
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用，`Conda环境名` 请填写 `tutorial3`，硬件资源为2个GPU。创建应用后，进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 分步运行本文件

2.1 数据预处理

In [1]:
```
import torchvision.transforms as transforms
import torchvision.datasets as datasets

# 数据预处理
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224), # 随机裁剪并调整为 224x224
    transforms.RandomHorizontalFlip(), # 随机水平翻转
    transforms.RandomRotation(10),  # 随机旋转 10 度
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
])

val_transforms = transforms.Compose([
    transforms.Resize(256), # 缩放到 256 像素
    transforms.CenterCrop(224), # 中心裁剪到 224x224
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
])

# 加载训练集和测试集
# VAR_PLACEHOLDER
train_dataset = datasets.ImageFolder(root='/lustre/public/tutorial/data/imag
val_dataset = datasets.ImageFolder(root='/lustre/public/tutorial/data/imager
```

2.2 模型

ResNet 是由何凯明于 2015 年提出的网络结构，ResNet结构使得网络的层数能够做得更深，对之后的工作有深远的影响。这里我们使用的是PyTorch自带的ResNet-50模型。

In [2]:
```
import torch
import torchvision.models as models

# 加载未经过预训练的 ResNet-50 模型
model = models.resnet50() # 不使用预训练权重
```

2.3 训练与评估

我们需要在多卡上并行训练，这里我们使用的是两张A100显卡。

在训练过程中我们增加了学习率调整策略，以加速收敛。

完成模型训练和评估需要约 13 h

下面版本的代码只能在 GPU 上跑，请申请相应的资源并指定使用的 GPU 数量。

这里还实现了一些常用的函数和类，可用于计时和绘图等。

In [3]:
```
# 常用函数和类实现

# 评估函数
def accuracy(model, data_loader, devices, topk=(1, )):
    model.eval()
    total = 0
    correct = {k: 0 for k in topk}

    with torch.no_grad():
        for X, y in data_loader:
            X, y = X.to(devices[0]), y.to(devices[0])
            outputs = model(X)

            _, predicted = outputs.topk(max(topk), dim=1, largest=True, sort
            correct_k = (predicted == y.view(-1, 1).expand_as(predicted))

            total += y.size(0)
            for k in topk:
                correct[k] += correct_k[:, :k].float().sum().item()

    return  {f'top-{k}': 100 * correct[k] / total for k in topk}


# 训练函数
def train(model, train_loader, loss, optimizer, devices):
    model.train()

    loss_list = []
    time_load = log_time()
    time_train = log_time()

    data_iter = iter(train_loader)
    count = 0
    while True:
        # 加载数据
        try:
            time_load.start()
            X, y = next(data_iter)
            X, y = X.to(devices[0]), y.to(devices[0])
            time_load.stop()
            count += 1
        except StopIteration:
            break

        # 训练
        time_train.start()
        outputs = model(X)
        l = loss(outputs, y)
        optimizer.zero_grad()
        l.backward()
        optimizer.step()
```

```
        time_train.stop()

        # loss 记录
        loss_list.append(l.item())

        # 输出 注意：在 GPU 上使用逻辑语句非常耗时，如果不关心 epoch 中间的训练过程，请
        if (count > 0) and (count % 1000 == 0):
            average_load_time = sum(time_load.times[-1000:]) / 1000
            average_train_time = sum(time_train.times[-1000:]) / 1000
            average_loss = sum(loss_list[-1000:]) / 1000
            print(f"batch: {count} | batch loss: {average_loss} | batch loac

    average_loss = sum(loss_list) / len(loss_list)

    return average_loss, time_load.sum(), time_train.sum()


# 计时器
import time
from datetime import timedelta

class log_time():
    """记录运行过程的时间片段"""
    def __init__(self):
        self.times = []
        self.beg = None
        self.end = None

    def start(self):
        # 开始计时
        self.beg = time.time()

    def stop(self):
        # 停止计时
        self.end = time.time()
        self.times.append(self.end - self.beg)

    def avg(self):
        # 平均时长
        return sum(self.times) / len(self.times)

    def sum(self):
        # 总时长
        return sum(self.times)

# 过程记录和绘图
import matplotlib.pyplot as plt
from matplotlib_inline import backend_inline
from IPython import display

class log_process:
    """在训练过程中动态绘制数据"""
    def __init__(self, epochs, figsize=(4, 3)):
        backend_inline.set_matplotlib_formats('svg')
        self.epochs = epochs
        self.fig, self.ax1 = plt.subplots(figsize=figsize)
```

```python
        self.ax2 = self.ax1.twinx()
        self.epochs = epochs
        self.train_ls = []
        self.test_acc = []

    def update(self, epoch, average_loss, test_acc):
        """向图表中添加数据点并更新图表"""
        self.train_ls.append(average_loss)
        self.test_acc.append(test_acc)

        # 清除当前轴的内容

        self.ax1.cla()
        self.ax2.cla()

        # 绘制损失
        self.ax1.plot(list(range(1, epoch + 2)), self.train_ls, 'b-', label=
        self.ax1.set_xlabel('Epoch')
        self.ax1.set_ylabel('Train Loss', color='b')
        self.ax1.tick_params(axis='y', labelcolor='b')
        #self.ax1.set_yscale('log')
        self.ax1.set_xlim([1, self.epochs])

        # 绘制准确率
        self.ax2.plot(list(range(1, epoch + 2)), self.test_acc, 'r--', label
        #self.ax2.set_ylabel('Test Accuracy (%)', color='r')
        self.ax2.tick_params(axis='y', labelcolor='r')
        self.fig.legend(loc='upper right')

        display.display(self.fig)
        display.clear_output(wait=True)
```

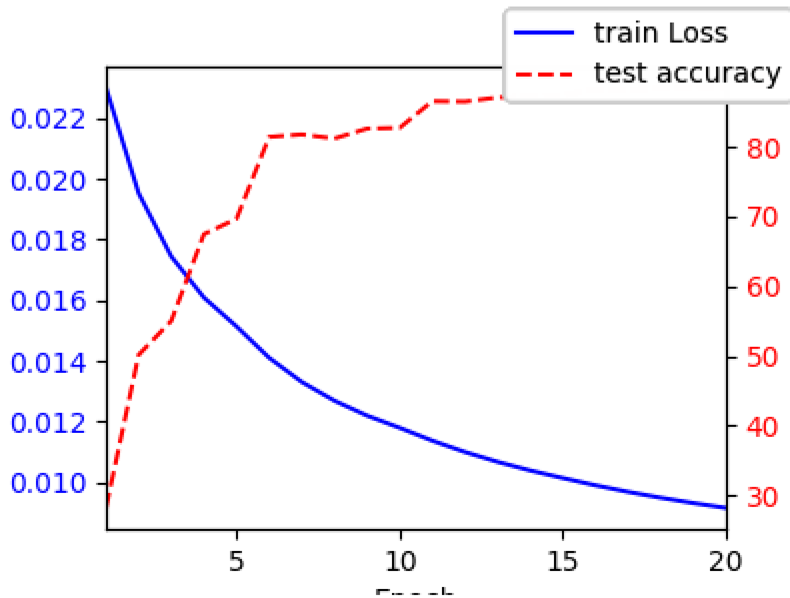
训练过程

In [ ]:

```python
# 训练过程

from torch.utils.data import DataLoader
import torch.nn as nn
import os

# 参数设置
epochs, batch_size, gpu_n = 20, 256, 2
learning_rate, momentum, weight_decay = 0.1, 0.9, 1e-4
topk = (1, 5)
show_top_k = 5

# 查看 GPU 数量
gpu_total = torch.cuda.device_count()
if gpu_total < gpu_n or gpu_total < 1:
    raise ValueError(f"Value gpu_n should <= {gpu_total} and gpu_total shoul

# 指定多GPU训练
model = models.resnet50()
devices = [torch.device(f'cuda:{i}') for i in range(gpu_total)]
model = nn.DataParallel(model, device_ids=devices)  # 指定使用计算使用的GPU
```

```python
model.to(devices[0]) # 移动模型到主设备上

# 损失函数
loss = nn.CrossEntropyLoss().to(devices[0]) # 损失函数也要在主设备上

# 优化器
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=m

# 学习率调整策略
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.

# 数据准备
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, r

# 主训练循环
train_ls = []
train_time = []
load_time = []
val_time = log_time()
train_process = log_process(epochs, figsize=(4, 3))
for epoch in range(epochs):

    # 训练
    batch_average_loss, epoch_load_time, epoch_train_time = train(model, tra
    train_ls.append(batch_average_loss/batch_size)
    load_time.append(epoch_load_time)
    train_time.append(epoch_train_time)

    # 验证
    val_time.start()
    accuracy_dict = accuracy(model, val_loader, devices, topk=topk)
    val_time.stop()

    # 调整学习率
    scheduler.step()

    # 更新和输出
    average_loss = sum(train_ls)/len(train_ls)
    average_load_time = timedelta(seconds=sum(load_time)/len(load_time))
    average_train_time = timedelta(seconds=sum(train_time)/len(train_time))
    average_val_time = timedelta(seconds=val_time.avg())
    print(f"loss: {average_loss} | test accuracy: {accuracy_dict} | load_tim

    train_process.update(epoch, average_loss, accuracy_dict[f'top-{show_top_

    # 保存模型
    os.makedirs('./modles', exist_ok=True)
    torch.save(model.state_dict(), f'./modles/resnet50_epoch_{epoch+1}.pth')
```

```
batch: 1000 | batch loss: 3.147843108892441 | batch load time: 0.36832323312
7594 | batch train time: 0.09844980454444885
batch: 2000 | batch loss: 3.112295998573303 | batch load time: 0.36611182928
08533 | batch train time: 0.08506760716438294
```



2.4 加载模型

之后可以加载模型参数重复使用模型训练的结果

In [ ]:
```
import torch
import torchvision.models as models

# 先确保有一个与之前保存权重相匹配的模型架构
model = models.resnet50(pretrained=False)

# 加载之前保存的权重
state_dict = torch.load(f'./modles/resnet50_epoch_{epoch+1}.pth')
model.load_state_dict({k.replace('module.',''):v for k,v in state_dict.items
```

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial4: Bge Embedding

本节旨在使用 bge-m3 模型给出句子的向量表示，并计算句子的语义相似度。

分以下几步来实现：

1. 环境安装与应用创建
2. 下载模型
3. 模型的使用

   3.1 稠密检索

   3.2 稀疏检索

   3.3 多向量检索

   3.4 加权语义相似度

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial4 python=3.9
conda activate tutorial4
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c
pytorch -c nvidia
pip install peft numpy==1.26.4 matplotlib==3.8.4 ipykernel==6.29.5
transformers==4.42.4
```

（ pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial4` , 硬件资源建议使用1张GPU运行。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 下载模型

建议在联网的命令行中下载模型，命令执行位置在当前文件所在的文件夹。

```
# 如果以下目录存在, 可以直接复制:
cp -r /lustre/public/tutorial/models/models--BAAI--bge-m3/ ./

# 否则请自行下载：
```

```
export HF_ENDPOINT=https://hf-mirror.com
huggingface-cli download --resume-download BAAI/bge-m3 --local-dir
models--BAAI--bge-m3
```

（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ；使用参考：https://tmuxcheatsheet.com）

3. 模型使用

在data shell 中执行：

```
pip install -U FlagEmbedding
```

3.1 稠密检索

稠密检索使用低维、密集的向量表示文本数据，将文本嵌入到连续的向量空间中，能够捕捉语义相似性，适合处理自然语言处理（NLP）任务中的模糊查询和复杂语义关系。运行下方代码。

In [ ]:
```
from FlagEmbedding import BGEM3FlagModel

# 填写模型路径
# VAR_PLACEHOLDER
model = BGEM3FlagModel('models--BAAI--bge-m3',
                       use_fp16=True)

# 待计算的句子
sentences_1 = ["What is BGE M3?", "Defination of BM25"]
sentences_2 = ["BGE M3 is an embedding model supporting dense retrieval, lex
               "BM25 is a bag-of-words retrieval function that ranks a set c

# 计算 Embedding
embeddings_1 = model.encode(sentences_1,
                            batch_size=12,
                            max_length=8192,
                            )['dense_vecs']
embeddings_2 = model.encode(sentences_2)['dense_vecs']

# 计算相似度
similarity = embeddings_1 @ embeddings_2.T
print(similarity)
# 结果应该是：
# [[0.6265, 0.3477], [0.3499, 0.678 ]]
```

3.2 稀疏检索

稀疏检索使用高维、稀疏的向量表示文本，其中大部分特征值为零，其计算效率高，易于解释，适合处理短文本和关键词匹配。运行下方代码。

In [ ]:
```
from FlagEmbedding import BGEM3FlagModel

# 填写模型路径
# VAR_PLACEHOLDER
model = BGEM3FlagModel('models--BAAI--bge-m3',  use_fp16=True)

# 待计算的句子
sentences_1 = ["What is BGE M3?", "Defination of BM25"]
sentences_2 = ["BGE M3 is an embedding model supporting dense retrieval, lex
               "BM25 is a bag-of-words retrieval function that ranks a set c

# 通过 lexical mathcing 计算相似度
output_1 = model.encode(sentences_1, return_dense=True, return_sparse=True,
output_2 = model.encode(sentences_2, return_dense=True, return_sparse=True,

lexical_scores = model.compute_lexical_matching_score(output_1['lexical_weig
print(lexical_scores)
# 0.19554901123046875
print(model.compute_lexical_matching_score(output_1['lexical_weights'][0], c
# 0.0

# 查看每个 token 的 weight：
print(model.convert_id_to_token(output_1['lexical_weights']))
# [{'What': 0.08356, 'is': 0.0814, 'B': 0.1296, 'GE': 0.252, 'M': 0.1702, '3
#  {'De': 0.05005, 'fin': 0.1368, 'ation': 0.04498, 'of': 0.0633, 'BM': 0.25
```

3.3 多向量检索

多向量检索是一种混合方法，结合了稠密和稀疏检索的优点，使用多个向量来表示一个文档或查询。运行下方代码。

In [ ]:
```
from FlagEmbedding import BGEM3FlagModel

# 填写模型路径
# VAR_PLACEHOLDER
model = BGEM3FlagModel('models--BAAI--bge-m3',  use_fp16=True)

# 待计算的句子
sentences_1 = ["What is BGE M3?", "Defination of BM25"]
sentences_2 = ["BGE M3 is an embedding model supporting dense retrieval, lex
               "BM25 is a bag-of-words retrieval function that ranks a set c

# 通过 colbert 计算相似度
output_1 = model.encode(sentences_1, return_dense=True, return_sparse=True,
output_2 = model.encode(sentences_2, return_dense=True, return_sparse=True,

print(model.colbert_score(output_1['colbert_vecs'][0], output_2['colbert_vec
print(model.colbert_score(output_1['colbert_vecs'][0], output_2['colbert_vec
# 0.7797
# 0.4620
```

3.4 加权语义相似度

计算三种检索的加权平均值：

In [ ]:
```
from FlagEmbedding import BGEM3FlagModel

# 填写模型路径
# VAR_PLACEHOLDER
model = BGEM3FlagModel('models--BAAI--bge-m3',  use_fp16=True)

# 待计算的句子
sentences_1 = ["What is BGE M3?", "Defination of BM25"]
sentences_2 = ["BGE M3 is an embedding model supporting dense retrieval, lex
               "BM25 is a bag-of-words retrieval function that ranks a set c

sentence_pairs = [[i,j] for i in sentences_1 for j in sentences_2]

# 计算混合相似度
# w[0]*dense_score + w[1]*sparse_score + w[2]*colbert_score
print(model.compute_score(sentence_pairs,
                          max_passage_length=128,
                          weights_for_different_modes=[0.4, 0.2, 0.4]))
```

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial5: bge-reranker-v2-m3

本节旨在展示使用 bge-reranker-v2-m3 模型来计算语言相似度的流程。

分以下几步来实现：

1. 环境安装与应用创建
2. 下载模型
3. 模型使用

    3.1 通过 FlagEmbedding 运行

    3.2 通过 Huggingface Transformer 运行

reranker 使用 cross similarity 来计算两个句子的相似度，把两个句子共同作为模型的输入，与 embedding 方法相比更耗费计算资源，但是对语言有更好的理解。

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial5 python=3.9
conda activate tutorial5
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c
pytorch -c nvidia
pip install notebook jupyterlab peft numpy==1.26.4
matplotlib==3.8.4 transformers==4.42.4
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial5` , 硬件资源为1个GPU。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 下载模型

在联网的命令行中执行，命令执行位置在当前文件所在的文件夹。

```
# 如果以下目录存在， 可以直接复制:
cp -r /lustre/public/tutorial/models/models--BAAI--bge-reranker-v2-
m3/ ./

# 否则请自行下载：
```

```
export HF_ENDPOINT=https://hf-mirror.com
huggingface-cli download --resume-download BAAI/bge-reranker-v2-m3
--local-dir models--BAAI--bge-reranker-v2-m3
```

（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ；使用参考：https://tmuxcheatsheet.com）

3. 模型使用

3.1 通过 FlagEmbedding 运行

使用 FlagEmbedding， 计算 cross similarity

在data shell 中执行，安装 FlagEmbedding 包：

```
pip install -U FlagEmbedding
```

安装后，运行下方代码：

In [ ]:
```
from FlagEmbedding import FlagReranker

# 填写模型路径
# VAR_PLACEHOLDER
reranker = FlagReranker('models--BAAI--bge-reranker-v2-m3', use_fp16=True)

# 计算相似度
score = reranker.compute_score(['query', 'passage'], normalize=True)
print(score) # 0.003497010252573502

scores = reranker.compute_score([['what is panda?', 'hi'], ['what is panda?'
print(scores) # [0.00027803096387751553, 0.9948403768236574]
```

3.2 通过 Huggingface Transformer 运行

或者使用 Huggingface Transformer 也可以调用模型，运行下方代码：

In [ ]:
```
import torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer

# 填写模型路径
# VAR_PLACEHOLDER
modle_path = 'models--BAAI--bge-reranker-v2-m3'

# 加载模型和分词器
tokenizer = AutoTokenizer.from_pretrained(modle_path)
model = AutoModelForSequenceClassification.from_pretrained(modle_path)
model.eval()
```

```
# 把模型移动到显卡
print(torch.cuda.is_available())
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)


# 计算相似度
pairs = [['what is panda?', 'hi'], ['what is panda?', 'The giant panda (Ailu
with torch.no_grad():
    inputs = tokenizer(pairs, padding=True, truncation=True, return_tensors=

    # 将输入数据移动到GPU
    inputs = {key: value.to(device) for key, value in inputs.items()}

    scores = model(**inputs, return_dict=True).logits.view(-1, ).float()
    print(scores)
```

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial6: qwen2-7b 推理和微调

本章旨在使用 Qwen2-7B-Instruct 模型展示模型对话、微调训练的过程。

分以下几步来实现：

1. 环境安装与应用创建
2. 下载模型
3. 模型推理
4. 模型微调

    4.1 安装 LLaMA Factory

    4.2 使用 LLaMA Factory 微调

5. 合并 Lora 参数
6. 加载合并后的模型
7. 使用 Web UI

Qwen 系列模型是由阿里巴巴开发的。Qwen 模型系列包括不同规模的模型，参数范围从 0.5 到 720 亿，适用于各种应用场景，如文本生成、翻译、问答等。Qwen2-7B-Instruct 支持高达 131,072 个 token 的上下文长度，能够处理大量输入。

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial6 python=3.9
conda activate tutorial6
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c
pytorch -c nvidia
pip install notebook jupyterlab accelerate numpy==1.26.4
matplotlib==3.8.4 transformers==4.42.4
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial6` , 硬件资源建议使用一张A100或V100 32G。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 下载模型

在联网的命令行中下载模型：

执行命令的路径为本文件所在文件夹下。

```
# 如果以下目录存在， 可以直接复制:
cp -r /lustre/public/tutorial/models/models--Qwen--Qwen2-7B-Instruct/ ./
# 否则请下载:
export HF_ENDPOINT=https://hf-mirror.com
huggingface-cli download --resume-download Qwen/Qwen2-7B-Instruct --local-dir models--Qwen--Qwen2-7B-Instruct
```

（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ； 使用参考：https://tmuxcheatsheet.com）

3. 模型推理

运行以下代码：

```python
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# 加载模型和分词器
# VAR_PLACEHOLDER
model_path = "models--Qwen--Qwen2-7B-Instruct"

tokenizer = AutoTokenizer.from_pretrained(model_path)

model = AutoModelForCausalLM.from_pretrained(
    model_path,
    torch_dtype="auto",
    device_map="auto"
)
model.eval()

# 移动模型到 GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# prompt
prompt = "介绍一下大模型"
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": prompt}
]
```

```python
# 对话
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)
model_inputs = tokenizer([text], return_tensors="pt").to(device)

generated_ids = model.generate(
    model_inputs.input_ids,
    max_new_tokens=512
)
generated_ids = [
    output_ids[len(input_ids):] for input_ids, output_ids in zip(model_input
]

response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0

print(response)
```

4. 模型微调

4.1 安装 LLaMA Factory

使用 LLaMA Factory 进行微调。

在联网的命令行中下载并安装LLaMA Factory:

```bash
git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e ".[torch,metrics,deepspeed,qwen]"
```

4.2 使用 LLaMA Factory 微调

运行以下命令解压数据集并创建配置文件，再执行微调:

In [ ]:

```bash
%%bash
tar -xzf data.tar.gz

echo "# 模型相对路径
model_name_or_path: models--Qwen--Qwen2-7B-Instruct

### method
stage: sft
do_train: true
finetuning_type: lora
lora_target: all

### dataset
dataset_dir: data/
dataset: my_identity
template: qwen
```

```
cutoff_len: 1024
max_samples: 1000
overwrite_cache: true
preprocessing_num_workers: 16

### output
output_dir: saves/Qwen2-7B-adapter # 微调参数保存路径
logging_steps: 10
save_steps: 500
plot_loss: true
overwrite_output_dir: true

### train
per_device_train_batch_size: 4
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
num_train_epochs: 70.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
ddp_timeout: 180000000

### eval
val_size: 0.1
per_device_eval_batch_size: 4
eval_strategy: steps
eval_steps: 500
" > qwen_finetune.yaml

llamafactory-cli train qwen_finetune.yaml
```

5. 合并 Lora 参数

创建配置文件并执行模型参数合并

In [ ]:

```
%%bash
echo "model_name_or_path: models--Qwen--Qwen2-7B-Instruct
adapter_name_or_path: saves/Qwen2-7B-adapter # 微调参数路径
template: qwen
finetuning_type: lora

## export
export_dir: outputs/Qwen2-7B-Finetuned # 模型合并后导出路径
export_size: 2
export_device: auto
export_legacy_format: false" > qwen_merge.yaml

llamafactory-cli export qwen_merge.yaml
```

6. 加载合并后的模型

创建配置文件并进行微调后的模型的推理

In [ ]:

```
%%bash
echo "# 模型路径
model_name_or_path: outputs/Qwen2-7B-Finetuned
template: qwen" > qwen-chat.yaml

llamafactory-cli chat qwen-chat.yaml
```

7. 使用 Web UI

上述过程也可以使用图形化的 webui 界面进行。执行下面命令，即可打开 webui 界面。为了避免路径错误，建议在加载模型、导出模型时使用绝对路径。

```
llamafactory-cli webui
```

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial7: qwen2-72b-int4 推理

本节旨在展示如何在 scow 平台上使用单机单卡运行 Qwen2-72B-Instruct-GPTQ-Int4 模型的推理任务。

分以下几步来实现：

1. 环境安装与应用创建
2. 下载模型
3. 模型推理

Qwen 系列模型是由阿里巴巴开发的。Qwen 模型系列包括不同规模的模型，参数范围从 0.5 到 720 亿，适用于各种应用场景，如文本生成、翻译、问答等。Qwen2-72B-Instruct-GPTQ-Int4 支持高达 131,072 个 token 的上下文长度，能够处理大量输入。

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial7 python=3.9
conda activate tutorial7
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c
pytorch -c nvidia
pip install notebook jupyterlab accelerate numpy==1.26.4
matplotlib==3.8.4 optimum>1.13.2 auto-gptq>0.4.2
transformers>=4.32.0,<4.38.0 accelerate tiktoken einops
transformers_stream_generator==0.0.4 scipy
pip install --upgrade pyarrow
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial7` , 硬件资源为1个GPU，建议使用 A100 40G 或者 V100 32G。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 下载模型

在联网的命令行中执行，命令执行位置在当前文件所在的文件夹。

```
# 如果以下目录存在, 可以直接复制:
cp -r /lustre/public/tutorial/models/models--Qwen--Qwen2-72B-
Instruct-GPTQ-Int4/ ./

# 否则请下载:
```

export HF_ENDPOINT=https://hf-mirror.com
huggingface-cli download --resume-download Qwen/Qwen2-72B-Instruct-GPTQ-Int4 --local-dir models--Qwen--Qwen2-72B-Instruct-GPTQ-Int4
（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ；使用参考：https://tmuxcheatsheet.com）

3. 模型推理

[参考链接]

运行下方代码进行模型推理:

In [ ]:
```python
from transformers import AutoModelForCausalLM, AutoTokenizer

# 使用 GPU
device = "cuda"

# 模型路径
# VAR_PLACEHOLDER
model_path = "models--Qwen--Qwen2-72B-Instruct-GPTQ-Int4"

# 加载模型和分词器
model = AutoModelForCausalLM.from_pretrained(
    model_path,
    torch_dtype="auto",
    device_map="auto"
)
tokenizer = AutoTokenizer.from_pretrained(model_path)

# prompt
prompt = "什么是大语言模型"
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": prompt}
]

# 生成回答
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)
model_inputs = tokenizer([text], return_tensors="pt").to(device)

generated_ids = model.generate(
    model_inputs.input_ids,
    max_new_tokens=512
)
generated_ids = [
```

```
    output_ids[len(input_ids):] for input_ids, output_ids in zip(model_input
]

response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0

print(response)
```

推理过程中使用 nvidia-smi 命令可以查看 GPU 运行情况。

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial8: Qwen2-72B-Instruct 单机多卡

本节展示如何在 SCOW 平台上使用单机多卡运行 Qwen2-72B-Instruct 模型的推理任务。

分以下几步来实现：

1. 环境安装与应用创建
2. 下载模型
3. 模型推理

Qwen 系列模型是由阿里巴巴开发的。Qwen 模型系列包括不同规模的模型，参数范围从 0.5 到 720 亿，适用于各种应用场景，如文本生成、翻译、问答等。Qwen2-72B-Instruct 支持高达 131,072 个 token 的上下文长度，能够处理大量输入。

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial8 python=3.9
conda activate tutorial8
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c
pytorch -c nvidia
pip install --upgrade notebook jupyterlab transformers
huggingface_hub accelerate
```

（ pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial8` , 建议的硬件资源为2张 A100。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 下载模型

在联网的命令行中执行，命令执行位置在当前文件所在的文件夹。

```
# 如果以下目录存在， 可以直接复制:
cp -r /lustre/public/tutorial/models/models--Qwen--Qwen2-72B-
Instruct/ ./

# 否则请下载:
export HF_ENDPOINT=https://hf-mirror.com
huggingface-cli download --resume-download Qwen/Qwen2-72B-Instruct
--local-dir models--Qwen--Qwen2-72B-Instruct
```

（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ；使用参考：https://tmuxcheatsheet.com）

3. 模型推理

创建JupyterLab交互应用，进行模型推理：

In [ ]:

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# 使用 GPU
device = "cuda"

# 模型路径
# VAR_PLACEHOLDER
model_path = "models--Qwen--Qwen2-72B-Instruct"

# 加载模型和分词器
model = AutoModelForCausalLM.from_pretrained(
    model_path,
    torch_dtype="auto",
    device_map="auto"
)
tokenizer = AutoTokenizer.from_pretrained(model_path)

# prompt
prompt = "什么是大模型"
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": prompt}
]

# 生成回答
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)
model_inputs = tokenizer([text], return_tensors="pt").to(device)

generated_ids = model.generate(
    model_inputs.input_ids,
    max_new_tokens=512
)
generated_ids = [
    output_ids[len(input_ids):] for input_ids, output_ids in zip(model_input
]

response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0
```

```
print(response)
```

推理过程中使用 nvidia-smi 命令可以查看 GPU 运行情况。

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn

Tutorial9: Stable_diffusion

本节旨在展示使用 stable-diffusion-3-medium-diffusers 模型进行文生图任务。

分以下几步来实现：

1. 环境安装与应用创建
2. 下载模型
3. 文生图

Stable Diffusion 是由 Stability AI 开发的一个开源的深度学习模型，用于生成高质量图像。

1. 环境安装与应用创建

首先在联网的命令行中创建conda环境:

```
conda create -n tutorial9 python=3.9
conda activate tutorial9
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia
pip install notebook jupyterlab sentencepiece protobuf numpy==1.26.4 matplotlib==3.8.4 transformers==4.42.4
pip install -U diffusers
```

（pytorch 版本需与 cuda 版本对应，请查看版本对应网站：https://pytorch.org/get-started/previous-versions ，通过 nvidia-smi 命令可查看 cuda 版本）

然后创建JupyterLab应用, `Conda环境名` 请填写 `tutorial9`, 硬件资源为 1 张 A100 显卡。创建应用后, 进入应用并打开本文件。

CUDA Version: 12.1; Torch Version: 2.3.1

2. 下载模型

在联网的命令行中执行，命令执行位置在当前文件所在的文件夹。

```
# 如果以下目录存在， 可以直接复制:
cp -r /lustre/public/tutorial/models/stable-diffusion-3-medium-diffusers/ ./

# 否则请下载, 其中 "hf_***" 是 huggingface 官网为每个用户提供的 token 序列
export HF_ENDPOINT=https://hf-mirror.com
huggingface-cli download --token hf_*** --resume-download stabilityai/stable-diffusion-3-medium-diffusers --local-dir stable-diffusion-3-medium-diffusers
```

（建议使用 tmux 工具进行数据下载。tmux（Terminal Multiplexer）是一个终端复用器，它允许用户在一个单一终端会话中运行多个终端会话，并且它的会话可以在不同的时间和地点断开和重新连接，非常适合远程工作和需要长时间运行的任务。关于 tmux 的安装和介绍参考：https://tmuxcheatsheet.com/how-to-install-tmux ；使用参考：https://tmuxcheatsheet.com）

3. 文生图

运行以下代码，从文字生成图像：

```
In [ ]: import torch
from diffusers import StableDiffusion3Pipeline

# 加载模型
pipe = StableDiffusion3Pipeline.from_pretrained("stable-diffusion-3-medium-c

# 使用 GPU
pipe = pipe.to("cuda")

# promt 内容，可以使用多个 prompt
# prompt2 = "Photorealistic"
prompt = "Albert Einstein leans forward, holds a Qing dynasty fan. A butterf

# 根据 prompt 生成多张图片
for i in range(10):
    image = pipe(
            prompt=prompt,
            # prompt_2=prompt2,
            negative_prompt="ugly, deformed, disfigured, poor details, bad a
            num_inference_steps=70,
            guidance_scale=7,
            height=1024,
            width=1024,
        ).images[0]

    image.save(f"{i}.png")
```

生成的图像在本地目录下，可点击或下载查看。

> 作者: 黎颖; 龙汀汀
>
> 联系方式: yingliclaire@pku.edu.cn; l.tingting@pku.edu.cn